Database Web Programming

OO - XML

Bipin C. Desai
Arlin L. Kipling

BytePress

Database Web Programming

Open Office Version

Bipin C. Desai
Arlin L. Kipling

Concordia University
Montreal

BytePress

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
ISBN: 978-1-988392-04-2

Dedication

To my family and the next generation:

A, A, C, L, L, R, W

Bipin C. Desai

To my wife, Joan, and daughters, Caroline, Diane and Arlene

Arlin L. Kipling

CONTENTS

Preface

This book is the result of teaching the laboratory component of an introductory course in Database Systems in the Department of Computer Science & Software Engineering, Concordia University, Montreal.. The intent of this part of the course was to have the students create a practical web-based application wherein the database forms the dynamic component of a real life application using a web browser as the user interface.

It was decided to use all open source software, namely, Apache web server, PHP, JavaScript and HTML, and also the open source database which started as MySQL and has since migrated to MariaDB.

The examples given in this book have been run successfully both using MySQL on a Windows platform and MariaDB on a Linux platform without any changes. However, the code may need to be updated as the underlying software systems evolve with time, as functions are deprecated and replaced by others. Hence the user is responsible for making any required changes to any code given in this book.

The readers are also warned of the changing privacy and data usage policy of most web sites. They should be aware that most web sites collect and mine user's data for private profit.

The authors wish to acknowledge the contribution of many students in the introductory database course over the years whose needs and the involvement of one of the authors in the early days of the web prompted the start of this project in the late part of the 20th century. This was the era of the dot com bubble. The corporations that survived this bubble have grown, thanks to data and its mining, to become the most powerful monopolies in the history of mankind. It is hoped that the open source community will replace these with locally controlled alternatives.

Overview

Chapter 1 Introduction
Database web programming combines two fields of computer science: database programming and web programming.
Chapter 2 MySQL Shell
The MySQL shell is used to access a MySQL database.
Chapter 3 SQL Basics
SQL (Structured Query Language) is the language used for relational databases.
Chapter 4 SQL Constraints
SQL constraints restrict the data that can be put into a database.
Chapter 5 SQL Relationships
A relationship in databases is an association or connection between two or more entity sets (tables of data).
Chapter 6 SQL/PSM
SQL/PSM (Structured Query Language/Persistent Stored Modules) is a procedural programming language.
Chapter 7 HTML
HTML (Hyper Text Markup Language) is the language in which web pages are written.
Chapter 8 CSS
CSS (Cascading Style Sheets). is a part of HTML which is used to describe the presentation of web pages.
Chapter 9 PHP
PHP (PHP Hypertext Preprocessor) is a server-side scripting language which is embedded in HTML. It is executed by an interpreter on the server computer and allows dynamic web pages.
Chapter 10 JavaScript
JavaScript is a client-side scripting language which.is embedded in HTML. It is executed by the browser on the client computer.
Chapter 11 State
State is memory retained between HTTP (HyperText Transfer Protocol) requests to a given web server.
Chapter 12 MySQL API
The MySQL API (Application Programming Interface) is a set of declarations of functions in PHP which communicate with a MySQL database.
Chapter 13 Applications
An application is a program or group of programs designed for users. The user of a well-designed application needs relatively little computer expertise.
Chapter 14 Word Wide Web, A Hacker's Heaven
The World Wide Web is not completely safe.
Appendix A Refernces
Appendix B Supplemantal material

1 Introduction

1.1 Database Web Programming

Database web programming combines two fields of computer science: database system programming and web programming. It is difficult to define this part of computer science in a few simple words because there are many software components which are combined to produce a database web application. The software is described later and this will define the scope of database web programming.

There are many examples of database web applications on the Internet. A familiar example is given in the next section.

1.2 Example

A simple and easy-to-understand example of a database web application is the telephone directory for Canada, called Canada411. The database for Canada411 records the name, address and telephone number of everybody in Canada who has a telephone number and that database resides in one or more computers connected to the Internet. Anyone who is able to use the Internet can access this database and retrieve information from the database.

Use a computer which is connected to the Internet, start Firefox, or some other web browser program. Then type the following in the address box, which is near the top of the browser window and press Enter.

http://www.canada411.ca/

Somewhere there is another computer connected to the Internet with the internet address, www.canada411.ca which is for the web server for the Canada411 web site. The forward slash, /, at the end of the name signifies a default web page in the document root directory (the directory on the hard disk in which all web pages are stored) of www.canada411.ca. This computer sends the default web page to the computer with the browser and the browser displays this page on the screen, which is shown in Figure 1.1. This is called the home page for www.canada411.ca.

The home page has has a number of forms (boxes where text could be entered) and are used to search for information in the Canada411 database. Complete one of the forms (Find a Person, for example) and then click the submit button on that form (enter Willams Gates in the Find a Person form). The Search button sends the data back to the web server and requests a result in a new web page. Complete one of the forms (Find a Person, for example) and then click the submit button on that form (Find, for example on the Find a Person form). The Search button sends the data back to the web server and requests a new (second) web page. The web server sends the second web page to the computer which is running the browser and the browser displays that page on the screen. The second web page shows the result of the search specified by the form on the home page. The second web page is called the result page below.

The home page and the result page are described in more detail below.

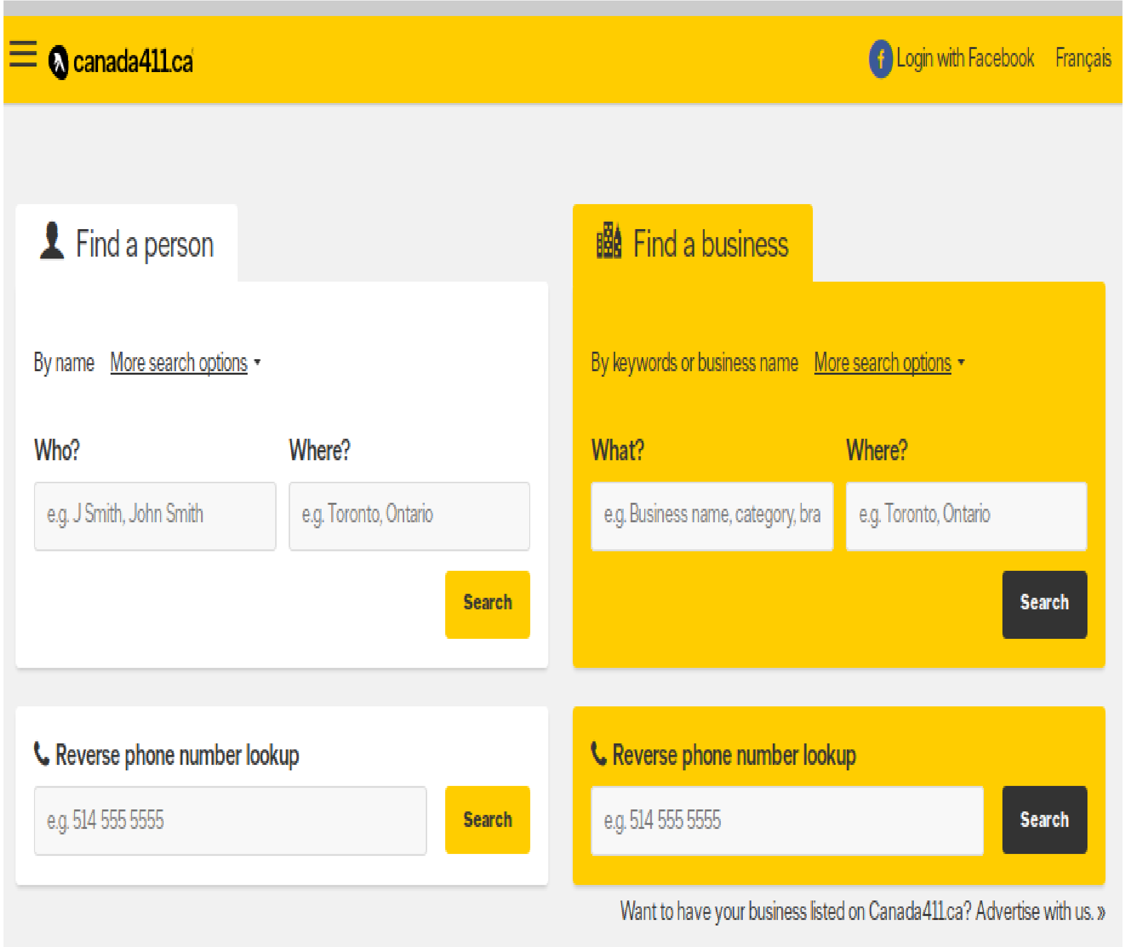


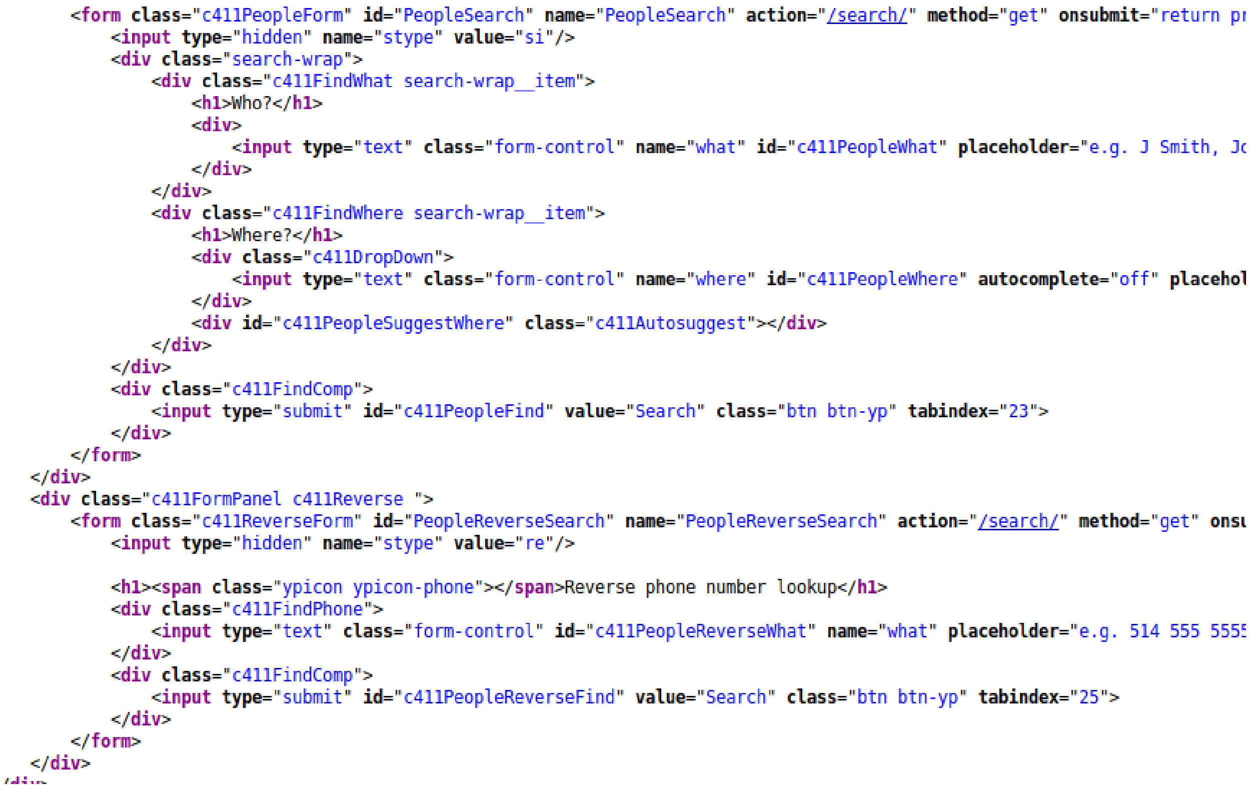
Figure 1.1 Canada411 home page

(1) Home page
There are a number of forms on the home page: one set of forms to find a person, one set of forms to find a business and a set of forms to reverse lookup a person or a business. The home page is shown in Figure 1.1. The purpose of the home page of the Canada411 web site is to receive information from a user about a person or business and then search for people or businesses specified by that information.

The source code for a web page can be viewed in a new window by right clicking on the web page to display a menu and then left clicking on the menu item to view the source code. The source code consists of instructions written in HTML (hypertext markup language) which is executed by the browser to produce the display on the screen.

Each form starts with the opening tag, <form *attributes* > and ends with the closing tag, </form>. In the opening tag, the value of the attribute, action, is the address of the web page which is requested when the submit button of the form is clicked. With the cursor at the top of the page, click Edit/Find to search for the first occurrence of, <form. Find the next occurrence of <form by clicking Find at the bottom of the screen. The source code of the part of the first web page with the first occurrence of <form is shown in Figure 1.2. Note that the form is inside a table,

which starts with the opening tag, <table *attributes* > and ends with closing tag, </table>. Close the source code window before proceeding.



```
        <form class="c411PeopleForm" id="PeopleSearch" name="PeopleSearch" action="/search/" method="get" onsubmit="return pr
            <input type="hidden" name="stype" value="si"/>
            <div class="search-wrap">
                <div class="c411FindWhat search-wrap__item">
                    <h1>Who?</h1>
                    <div>
                        <input type="text" class="form-control" name="what" id="c411PeopleWhat" placeholder="e.g. J Smith, Jo
                    </div>
                </div>
                <div class="c411FindWhere search-wrap__item">
                    <h1>Where?</h1>
                    <div class="c411DropDown">
                        <input type="text" class="form-control" name="where" id="c411PeopleWhere" autocomplete="off" placehol
                    </div>
                    <div id="c411PeopleSuggestWhere" class="c411Autosuggest"></div>
                </div>
            </div>
            <div class="c411FindComp">
                <input type="submit" id="c411PeopleFind" value="Search" class="btn btn-yp" tabindex="23">
            </div>
        </form>
    </div>
    <div class="c411FormPanel c411Reverse ">
        <form class="c411ReverseForm" id="PeopleReverseSearch" name="PeopleReverseSearch" action="/search/" method="get" onsu
            <input type="hidden" name="stype" value="re"/>

            <h1><span class="ypicon ypicon-phone"></span>Reverse phone number lookup</h1>
            <div class="c411FindPhone">
                <input type="text" class="form-control" id="c411PeopleReverseWhat" name="what" placeholder="e.g. 514 555 5555
            </div>
            <div class="c411FindComp">
                <input type="submit" id="c411PeopleReverseFind" value="Search" class="btn btn-yp" tabindex="25">
            </div>
        </form>
    </div>
</div>
```

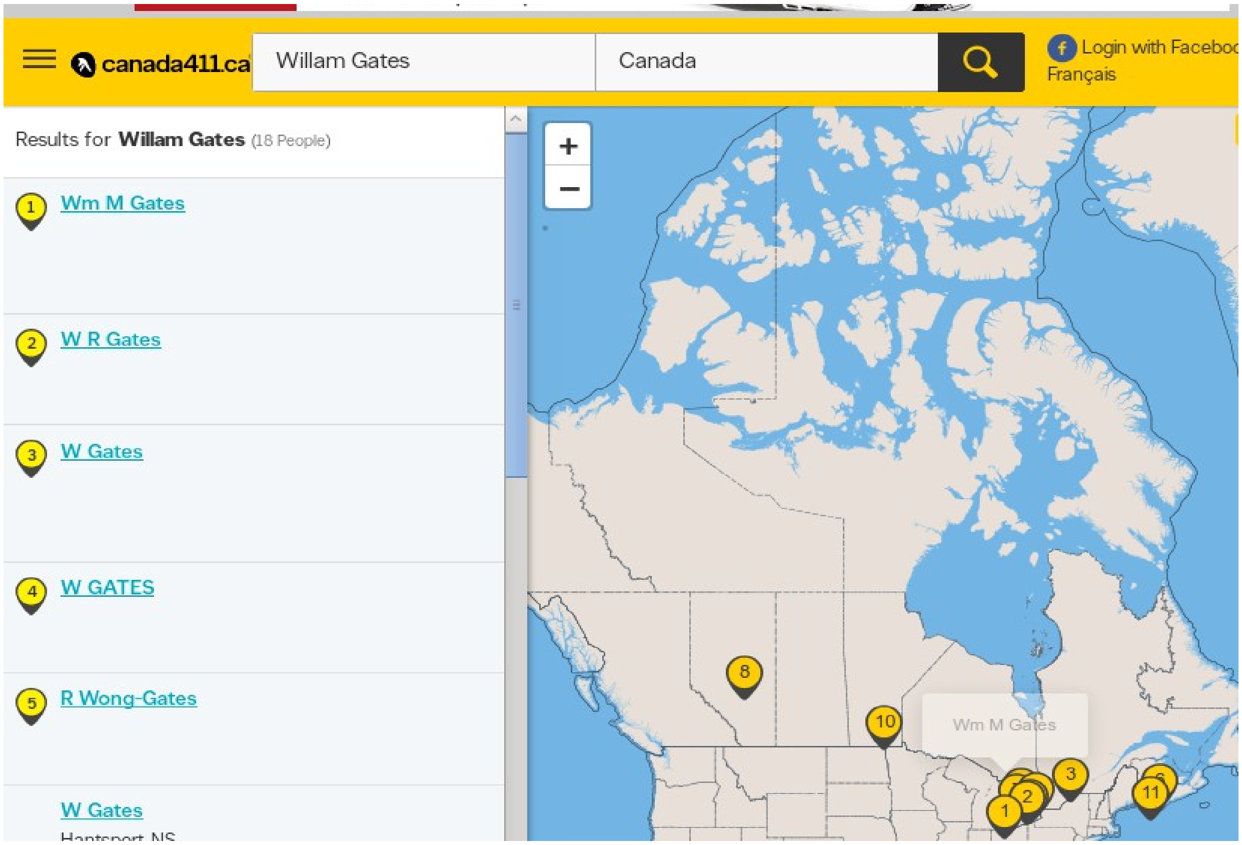
Figure 1.2 Source code of part of home page in previous figure

(2) Result page
The list of all people or businesses specified by the information entered on the home page is given on the result page. This data has been retrieved from the database of Canada411.

For example, if you are William Gates and you want to find all of the people with your name who have a phone in Canada, type the following in the home page. In the Find a Person form, type in Persons Name, William Gates, and in Location, type Canada. Then click, Find.

For this example, the first part of the result page is shown in Figure 1.3, at the time of writing. Note that all people with initial, W, as well as first name, William, are retrieved from the database. This list will eventually change since phones are connected and disconnected as people move, and the phone company updates the database with the changes. The addresses and home numbers in the results have been removed from Figure 1.3 to respect confidentially.

The source code for the result page can be viewed as described for the home page. In the source code, find the beginning of the list of names by typing, Results for, in the Find box of the browser. The beginning of the line, Results for William Gates, is highlighted. The source code with the heading, Results for. . . , near the top and the first part of the result is shown in Figure 1.4.



Willam Gates

Canada

Results for **Willam Gates** (18 People)

1 Wm M Gates

2 W R Gates

3 W Gates

4 W GATES

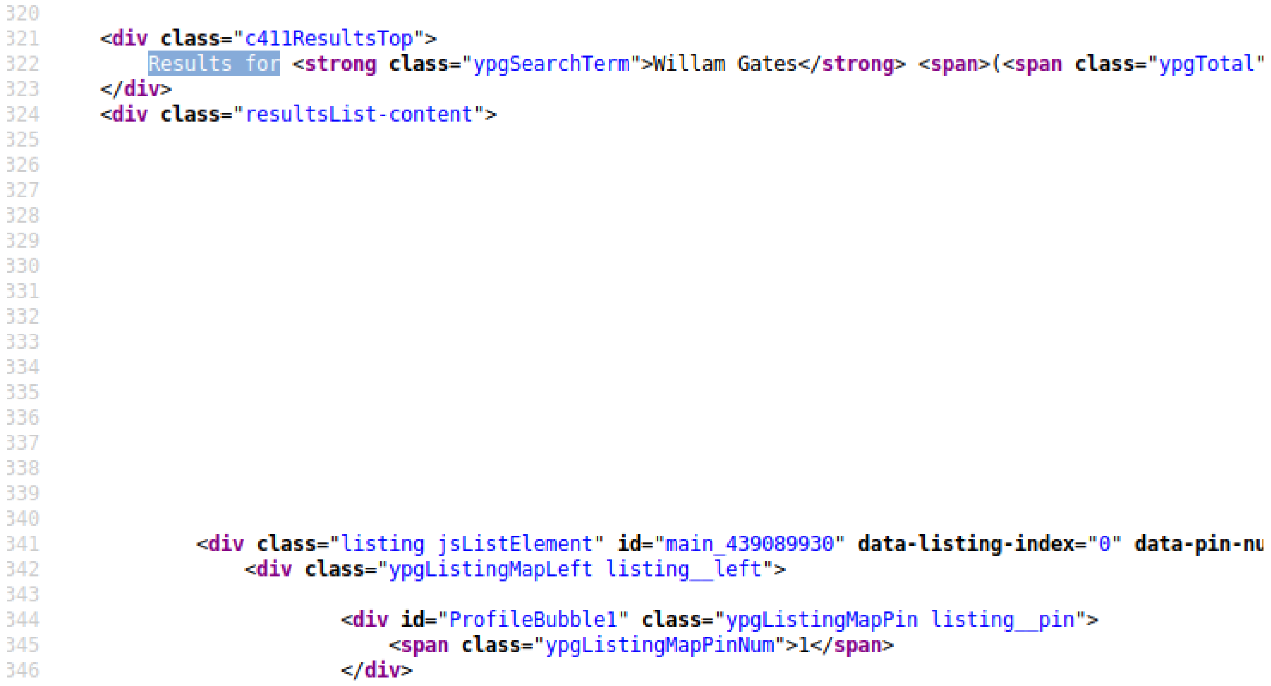
5 R Wong-Gates

W Gates
Hantsport NS

Figure 1.3 Canada411 result web page, first part

There are many links used in the result page. Links start with the opening tag, <a *attributes* > and end with the closing tag, </a>. The value of the attribute, href, of the opening tag is the address of the web site which is requested when the link is clicked. The text between the opening and closing tags is the label for the link that appears on the web page. Find the first link after the heading, by typing, "Results for. , in the Find box.

A user (somebody with a computer connected to the Internet) of the database for Canada411 can only retrieve information from the database. No one, except the organization running the web site, can put information into the database. However, there are many databases on the Internet where users can both insert information into and retrieve information from the database. One example is the ubiquitous dating service offered on many web sites. Another example is the web site that advertises apartments for rent, which has to a great extent replaced advertising in newspapers. Other examples are shopping web sites, such as Alibaba, Amazon, Otto, and auction and online commerce web sites such as, Alibaba, Amazon, Ebay and Shopify. On shopping websites people can buy many kinds of products and anyone can sign up to sell goods and services. On. commerce web sites people can both buy and sell goods and services.



```

    <div class="c411ResultsTop">
        Results for <strong class="ypgSearchTerm">Willam Gates</strong> <span>(<span class="ypgTotal"
    </div>
    <div class="resultsList-content">
















            <div class="listing jsListElement" id="main_439089930" data-listing-index="0" data-pin-nu
                <div class="ypgListingMapLeft listing__left">

                        <div id="ProfileBubble1" class="ypgListingMapPin listing__pin">
                            <span class="ypgListingMapPinNum">1</span>
                        </div>
```

Figure 1.4 Source code of part of result page in previous figure

1.3 Database

1.3.1 Concepts

A database is a place where data is stored. Conceptually, a database is a container and the data is the contents of the container. So, using a rather simple analogy, a database is like a bottle (a container) and the data is like the water (contents of the bottle).

Data is facts or items of information about something. The data in a database is not a random collection of facts, but facts which are connected in some way, for example, facts about students who are enrolled in a computer science program.

Database Structure

The majority of current databases use a database model called the relational model. A relational database consists of one or more tables which are containers for the data. The table is the only data structure used in a relational database.

The terminology used for the same elements of the relational model and a relational database (the implementation of the relational model) is different. The correspondence between the terminologies is as follows.

Relational model	Relational database
relation	table
attribute	column
tuple	row

Other terms are also used for a relational database: field is used as a synonym for column and record is used as a synonym for row. The terminology, table, column and row, will be used exclusively in this chapter.

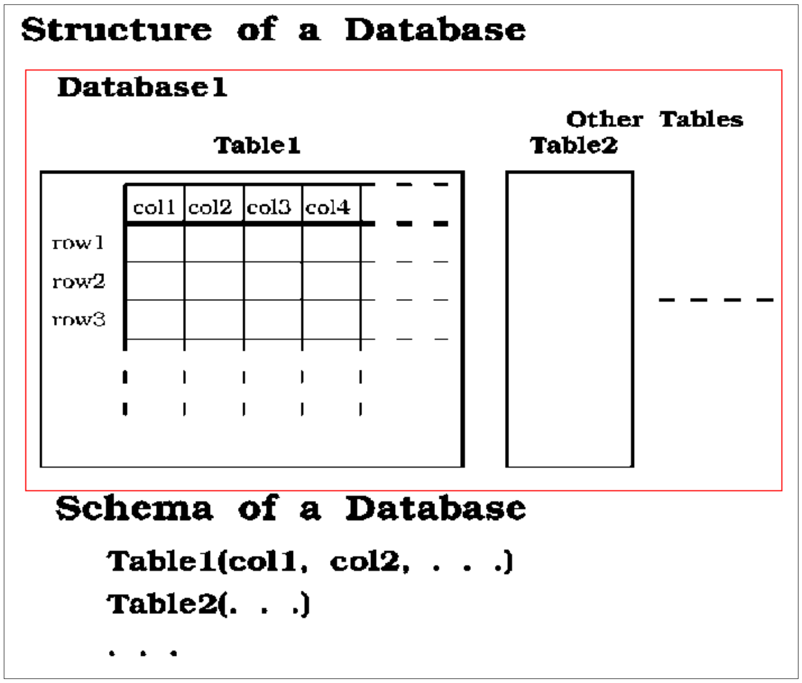


Figure 1.5 Database structure **Database and table**

Database and table

The structure of a relational database is shown in Figure 1.5. Since only the relational model is used, the word database will always imply relational database. The name of the database is Database1, and the names of the tables which constitute the database are Table1, Table2, and so on. A table consists of columns and rows. The names of the columns of Table1 are col1, col2, and so on. The number of columns of a table is fixed for each table. The rows of a table contain the data and are not named since a table can have any number of rows including zero rows. A particular row is identified uniquely by the data in one or more columns of the row. The rows of Table1 are labeled row1, row2, and so on, only to indicate where the rows are in the table.

Domain

There is one mandatory constraint for every column of a table and this is called the domain. In the relational model a domain is the set of values from which an attribute takes a value; in the database the column has a data type which corresponds to the domain. A domain is a constraint (restriction) on the value of data in a column of a table. An example of a domain is a string with a maximum number of characters. Another example of a domain is a number with a maximum number of digits and no digits after the decimal point (an integer). A domain in a database corresponds to a data type in a programming language, except a domain both specifies the data type and restricts the range of values of the data.

Schema
A schema specifies the structure of table. The schema of a table is the name of the table followed by the names of the columns in brackets, and optionally, the domain of each column after the name of the column. The brief schema of a database is shown in Figure 1.5. If the database in Figure 1.5 consists of only one table and the table has two columns, the brief schema is: Table1(col1, col2)

If the domain of col1 is domain1 and the domain of col2 is domain2, the detailed schema is

Table1(col1: domain1, col2: domain2)

Note that the name of the table is capitalized and the name of each column is not capitalized. This is a convention adopted here which helps to distinguish between names of tables and names of columns in a large database.

1.3.2 **Database Contents**

The contents of a database are the data in the rows of the tables of the database at a particular time. The word, instance, is often used to mean contents of a database at a particular time.

The contents of a very simple database are shown in Figure 1.6. The simplest database has only one table and the simplest table has only one column. The simple database in Figure 1.6 consists of one table, named Person. The table, Person, has two columns, with names, name and weight. The brief schema of this table, given in the figure, is

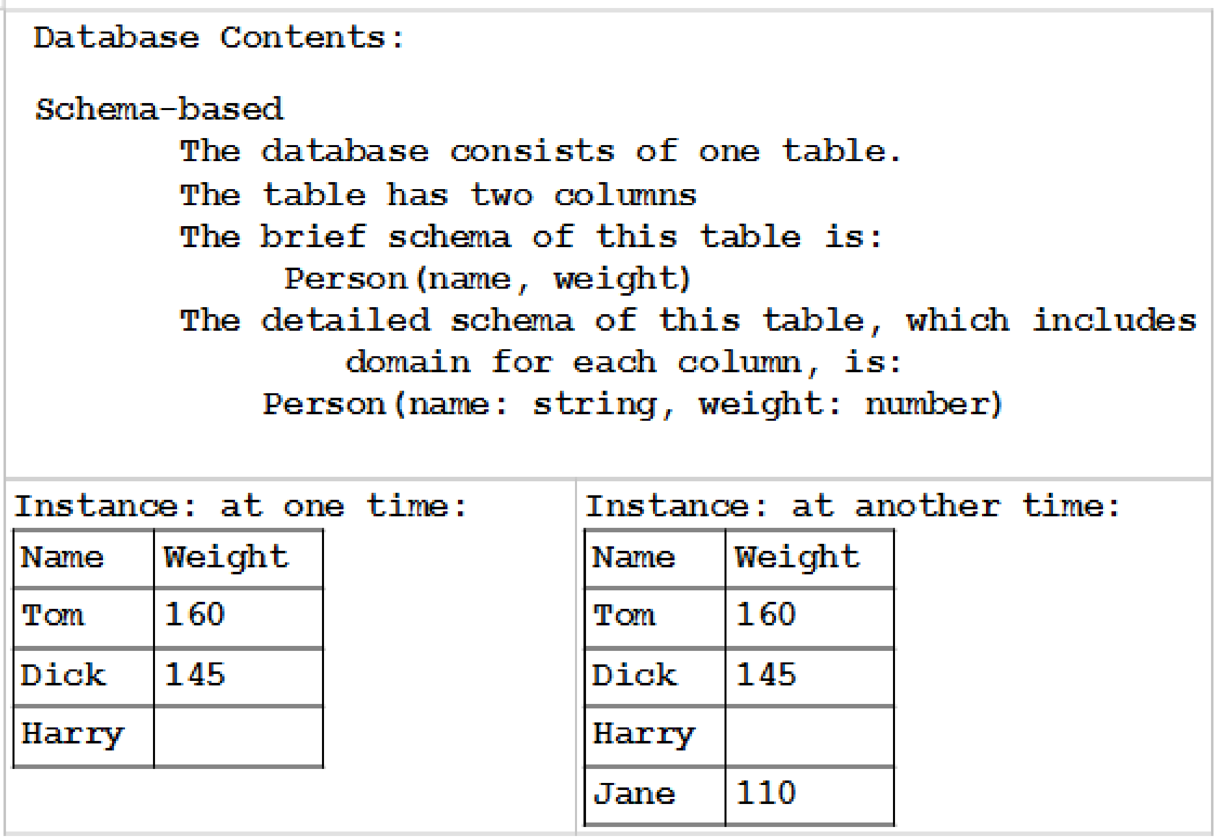
Person (name, weight)

The meaning of the table and the columns of the table is indicated by the names chosen for the table and columns. The data in the table is about a person who could be anyone. The data in column, name, could be the first name, last name or both names of the person. The data in the column, weight, could be the weight of the person in kilograms or pounds. So the name is expressed by a string (a sequence of characters which should be only letters in this case), that is, the domain of name is string. The weight is expressed by a number, that is, the domain of weight is number. The detailed schema of this table, also given in the figure, is

Person(name: string, weight: number)

The names of the columns could be more explicit, for example, firstName, instead of name and weightInPounds, instead of weight. Note the convention that is adopted here for names of columns that consist of more than one word. Spaces are not used between words in a name and the words after the first word in a name are capitalized so that the name is easier to read.

Instance
An instance of a table is the data in the table at a particular time. The word, instance, in this context means occurrence. The data contained in most databases changes with time as data is added, modified and removed from the database. In Figure 1.6 the instance of table, Person, has three rows of data at one time and four rows of data at another time. In the first instance, data about three persons is stored in the table; the data for each person is in one row. In the second instance, data about an additional person is also stored in the table. For each person, there are two data items, one in each column.



```
Database Contents:

Schema-based
       The database consists of one table.
       The table has two columns
       The brief schema of this table is:
            Person(name, weight)
       The detailed schema of this table, which includes
               domain for each column, is:
           Person(name: string, weight: number)
```

Instance: at one time:

Name	Weight
Tom	160
Dick	145
Harry	

Instance: at another time:

Name	Weight
Tom	160
Dick	145
Harry	
Jane	110

Figure 1.6 Database contents

Data in a row can be missing as illustrated by the third row which has the value, Harry, for the name but no value for the weight. The word, null, means no value or empty and so one can say that the weight of Harry is null. When data is inserted in a table and the data is missing (data has no value), the corresponding value is said to be null for the data instead of, for example, a zero-length string. The word, null, does not appear in the table.

1.4 Three-tier Architecture

There are three distinct components of a database web application and their functions are implemented in three distinct software parts or modules. Formally, these parts are called tiers and this compartmentalization of the software is called a three-tier architecture. The word tier in this context means layer or level.

In terms of function, the three tiers are the (a) user interface (presentation tier), (b) application rules and logic (application tier), (c) database system and its contents, storage and access (data storage tier). In terms of software to implement the function, the tiers are (a) web client, (b) web server and associated software to interact with database, (c) database server. The web client is called a browser and the most popular browsers at present are Firefox and Internet Explorer. The most popular web server is Apache. At the time of writing, the most popular commercial relational database server is Oracle and the most popular open-source (free) database server is MySQL and its equivalent MariaDB. New database systems have been introduced to address the newer web-based applications; these databases use models which are not relational and lumped under term NoSQL (not discussed in this text).

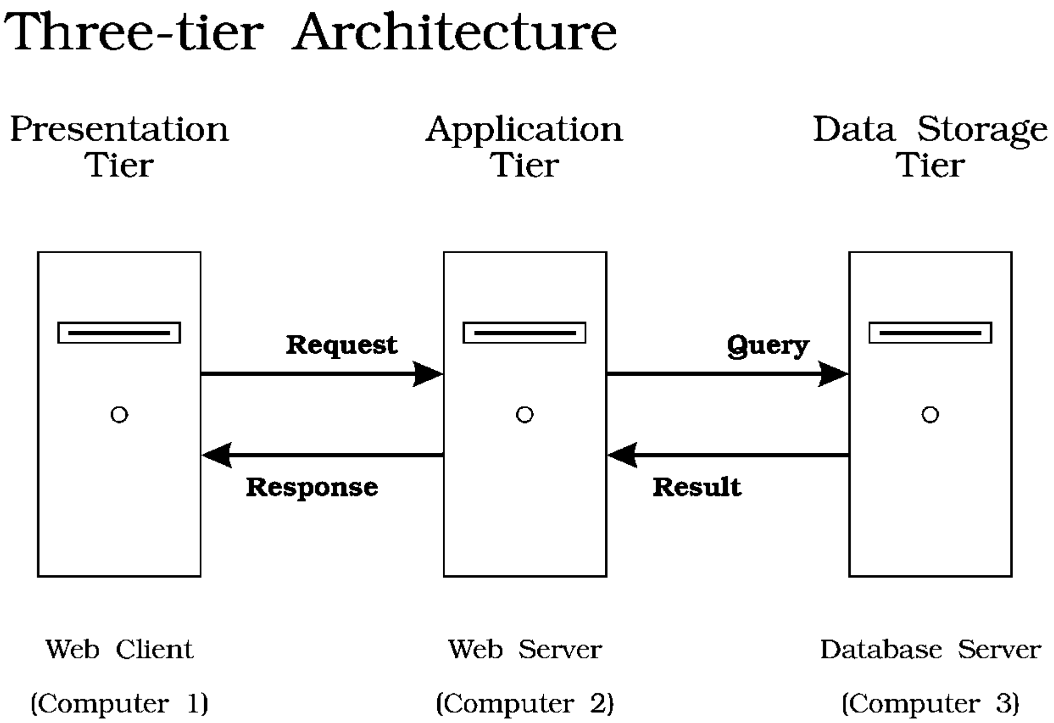


Figure 1.7 Three-tier architecture

The three tiers can reside on one, two, or three computer systems. These systems are different in terms of the software that is used on the computers. However, the hardware of the computers (processor, main memory, hard disk and so on) could be similar or even identical.

For the present consider that there are three physically different computers: (a) a web client computer, (b) a web server computer, and (c) a database server computer.

If the tiers reside on only two physical computers, one computer, web client computer, contains the web client software (browser) and another computer contains both the software of the web server and database server.

Also it is possible to have only one physical computer which contains all of the software; this is a convenient arrangement for software development.

Figure 1.7 shows the three-tier architecture with each tier represented by one computer. The presentation tier is a short name for the user interface tier. This tier is one or more web client computers which presents (displays) an interface to users. Users input data by means of the keyboard or mouse and receive data from the monitor. The application tier (short for application rules and logic tier) is one or more web server computers which store web pages. Web pages are requested by a web client computer and programs in the web pages are executed by the web server. The data storage tier (short for data storage and access tier) is one or more database server computers which are accessed (arrow labeled, query) by the web server computers. The data storage tier sends data (arrow labeled, result) to the application tier which in turn sends data (arrow labeled, response) to the presentation tier.

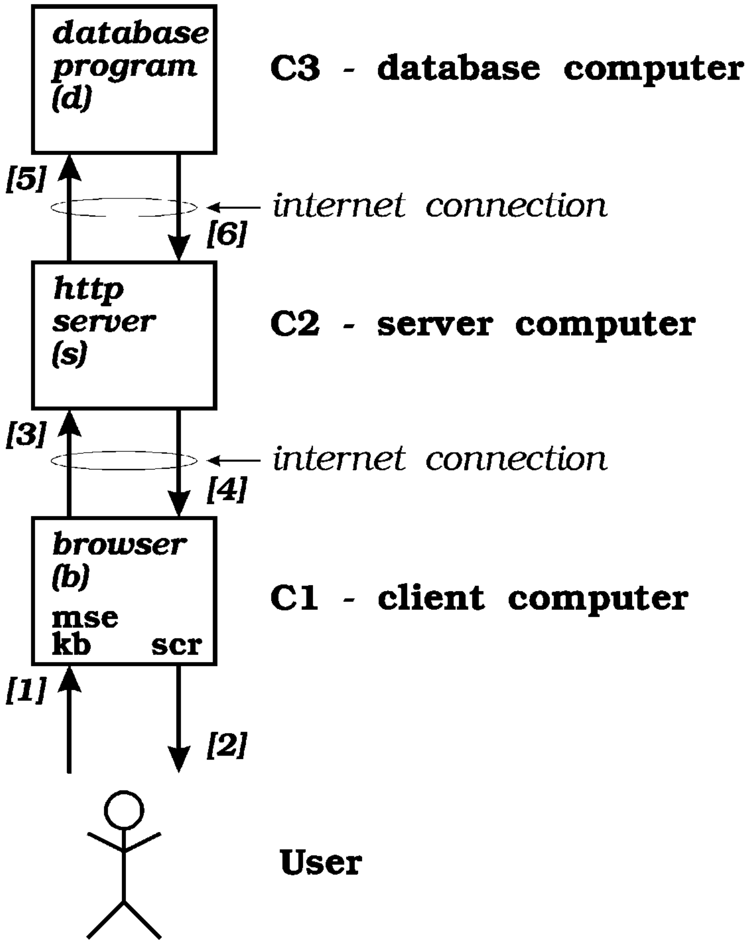


Figure 1.8 Sequence of Operations in a Three-tier architecture

1.5 **Operation**

The three tiers are shown in Figure 1.8 for the case where each tier is a physically different computer. Figure 1.8 is the same as Figure 1.7 but with detail which is used to describe the operation of the computers. The names of the computers are shortened to some extent. The web client computer is named client computer and labeled, C1, the web server computer is named server computer and labeled, C2, and the database server computer is named database computer and labeled, C3. The labels are used for convenience when referring to the computers. The computers, C1, C2 and C3, in Figure 1.8 are labeled 1, 2 and 3, respectively, in Figure 1.7. All three computers are part of the Internet (the Internet is discussed in a later section). The internet connections shown in Figure 1.8 allow communication between programs in C1 and C2, and in C2 and C3.

The software which is used directly by a database web application is shown in the figure: a browser in C1, an http server (also called web server) in C2 (http is the acronym for hypertext transfer protocol) and a database program in C3. An example of a browser is Internet Explorer when the operating system is Windows, an example of an http server is Apache and an example of a database program is MySQL. The paths numbered, [3], [4], [5], [6], are flows of information to and from the software in the computers.

The person using the database is called the user in Figure 1.8. The paths numbered, [1], [2], are flows of information between the user (person) and the browser (software). The user communicates with the browser of C1 by giving input to the keyboard (kb) and mouse (mse), and by receiving output from the screen (scr).

The operation of a database web application can be visualized using Figure 1.8. The level of understanding, described below, is probably sufficient for most users of web databases. The example used in a previous section, the telephone directory of Canada, is used again. The user gives input to the browser when the first web page, the home page, is displayed and then the result of the search is displayed on the second web page, the result page. So the description is divided into two parts, one for each web page.

(1) After User starts the browser in C1, User types the address, http://www.canada411.ca/, in the browser and then presses Enter - path [1].
Browser (b) sends a request to the http server (s) in C2 - path [3].
s sends a response to b which includes a web page - path [4].
b displays the first web page (home page) on the screen.
User views the web page - path [2].

(2) User completes one of the input text forms, then clicks the submit button, on the form - path [1].
b sends a request to s on C2 - path [3].
s sends a request (query) to the database program (d) on C3 - path [5].
d sends a response (result) to s - path [6].
s sends a response to b - path [4].
b displays a new web page on the screen which shows the result of the database query.
User views the result on the web page - path [2].

The flows of information in Figure 1.8 are either requests or responses. Paths [1], [3] and [5] are always requests and paths [2], [4] and [6] are always responses. Furthermore, the user always initiates the first request, path [1]. The terminology, request and response, is always used for paths [3] and [4], respectively. The term, query, is often used for path [5] instead of request and the term, result, is often used for path [6] instead of response, as labeled in Figure 1.7. However, path [5], is not always a query (question about data in the database). For example, path [5], could be an instruction to add data to the database, and in this case path [6] is not a result (data from the database).

1.6 **Internet**

There are millions of computers all over the world which are connected using hardware and software in such a way that the computers can exchange data. The various methods of connecting these computers are phone lines (twisted pairs of wires), TV cables (coaxial cables), fiber optics (glass or plastic strands) and wireless communication (electromagnetic waves traveling through space at the speed of light). Information is transferred in small chunks of data called packets. One communication between two computers will consist of many packets and will include control information such as the sender and the receiver. In general, the connection between two computers involves many other computers, which are sometimes called nodes that act as traffic cops to direct the flow of information.

Internet, with a capital I, is the name for this very large system of interconnected computers. The Internet resembles the global telephone system to some extent. Each telephone in the world has a unique telephone number and each internet computer has its own unique number, called an IP (internet protocol) address which is represented by the name of the computer.

There are two kinds of computers on the Internet in terms of data transfer: a computer that receives data, called a client, and a computer the supplies (or sends) data, called a server. Some computers can also transfer data in both directions and therefore are both a client and server.

A user (a person) operates the client by using the keyboard, mouse and screen. A program (called a browser) in the client requests data from a program (called a http server or web server) in the server and the server responds by sending the data to the client. Then the data is displayed on the screen of the client for the user to see.

1.7 Web

The name, web, is short for, World Wide Web, and is the collection of all the information stored on server computers, called web servers that are connected by the Internet. All information on the web is stored in files, commonly called web pages, which reside on the hard disks of the web servers. Note that a web server is not only a computer but is any device connected to the internet which is programmed to serve data, such as a web cam (digital video camera). The name, web, is well-chosen because most web pages have links (references) to other web pages and the links between all web pages form a very complex pattern which resembles a web.

The word, web, is sometimes used to refer to the server computers on which the web pages are stored (containers of web pages), instead of the web pages themselves (contents of server computers). This is not the strict definition of web. However, if the web is regarded as the computers which store the web pages, it is important to realize that the Internet and the web is not the same thing. The web is only part of the Internet, only the server computers of the Internet that store web pages. The rest of the Internet are other servers such as email and fileservers and client computers including mobile devices. However, because of the ease of use these servers are accessible using web browsers.

Figure 1.9 is a simple illustration of a small part of the Internet and the web. One client computer, client1, and two server computers, server1 and server2, are shown. All three computers are part of the Internet. But only the two server computers are part of the web since only they store web pages. The client computer is not part of the web. The client has a program called a browser which requests a web page from a server when a user initiates the request, as described below. The server has a program called a http server (http stands for hypertext transfer protocol) which receives a request for a web page from a browser and responds by sending a copy of the web page to the browser. The browser then displays the contents of the web page on the screen.

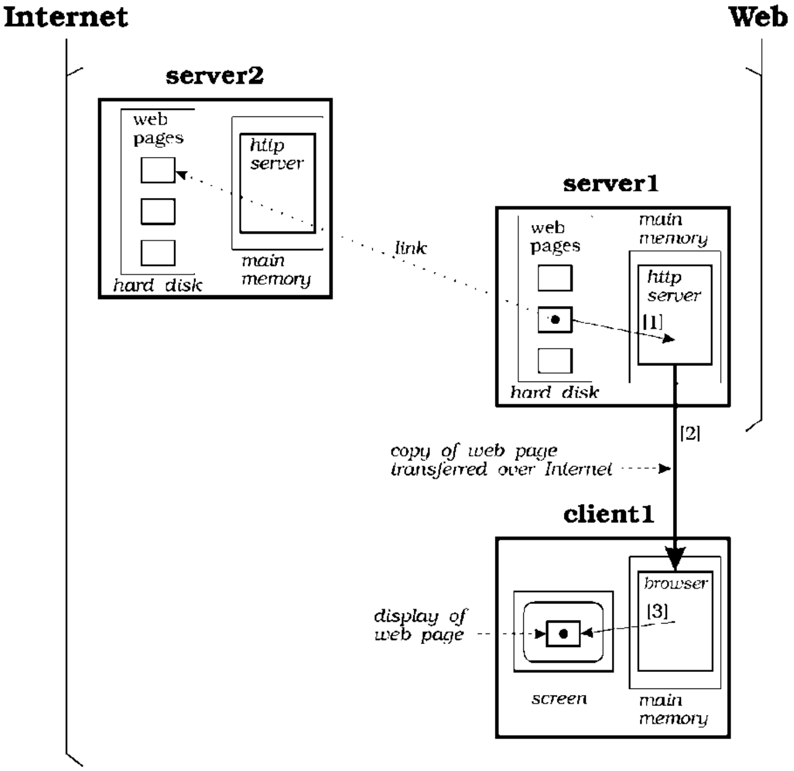


Figure 1.9 Internet and web

The user can initiate a request for a web page in two ways.

(1) The address of the web page can be typed into the address box at the top of the browser and then when Enter is pressed the web page is requested. This is shown in Figure 1.9. The computer, client1, makes a request for a web page from server1 which is transmitted over the Internet from the client to the server (this path is not shown in Figure 1.9). The web page is retrieved from the hard disk of the server by the http server (a program), label [1], and is sent to client1 over the Internet, label [2], where it is displayed by the browser of the client on the screen, label [3].

(2) If a web page is already being displayed by the browser and there is a link (address of another web page) in that web page, the link can be clicked to request the web page referenced by the link. A link in a web page is represented by a dot in Figure 1.9. In Figure 1.9 the web page displayed by client1 has a link to a web page on server2, as shown by the dotted line with the arrow between web pages on server1 and server2. When the link is clicked on the screen of client1, a copy of the web page from server2 is sent to client1 (the path is not shown in Figure 1.9) and the browser displays the new web page in place of the previous web page. In general, the web page displayed by client1 can have many links to different web pages which can be on server2, as well as on server1 and other servers not shown in the figure.

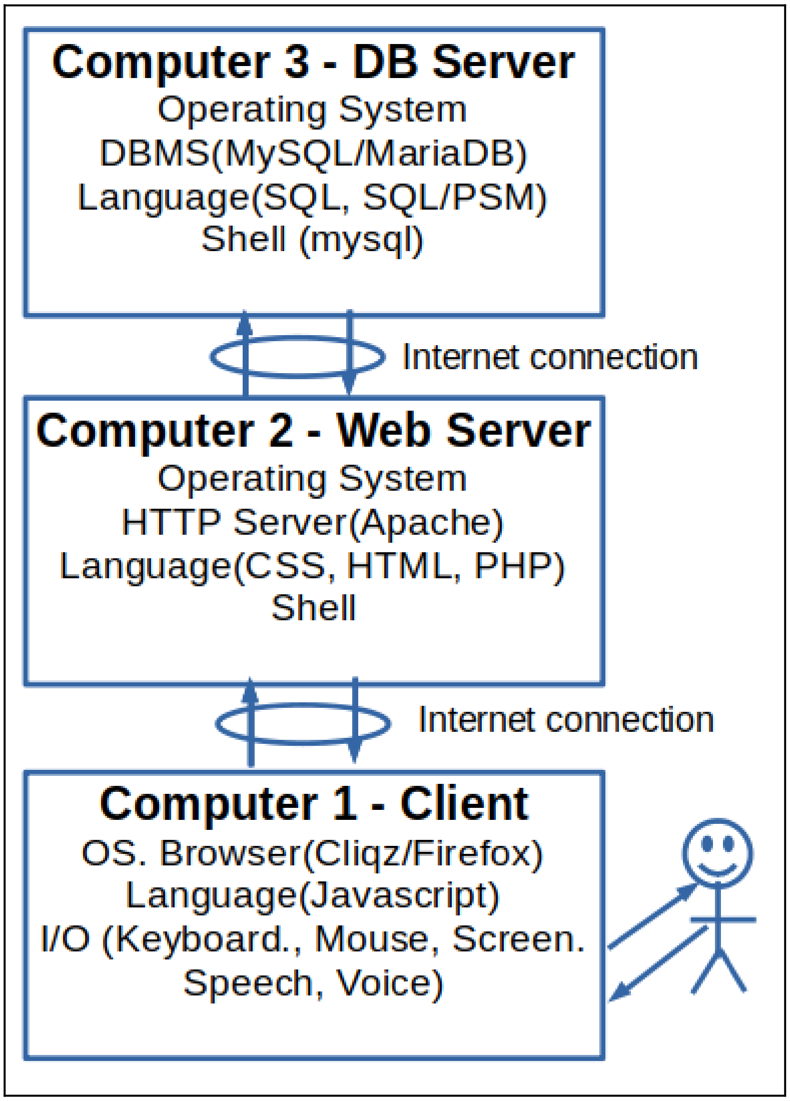


Figure 1.10 Software

The dotted line in Figure 1.9 is the link of one web page on one server computer to another web page on another server computer. The line connects two web pages and represents one strand of the World Wide Web. There are billions of such connections between pages of the World Wide Web.

1.8 Three-tier Software

The software that is needed for database web programming is listed in Figure 1.10. In general, the software of each tier is stored on a different computer. So there are three different computers which participate in a database web application, which are labeled Computer 1, 2 and 3 in Figure 1.10. The three computers are given the brief names, client, server and database. Note that Figure 1.10 is the same as Figure 1.8 but used for a different purpose. Refer to Figure 1.10 starting at the bottom.

1.8.1 User

The user is a person sitting in front of the client computer at home or at school or elsewhere. The keyboard, mouse and screen are the means by which the user interfaces to (communicates with) the browser (a program).

1.8.2 Client (Computer 1)

1.8.2.1 Operating System

An operating system like Windows or Linux can be used. One important difference between Windows and Linux is that Windows is neither open nor free but Linux is both open and free with many different distributions..Windows has gone through many versions and each version has some incompatibilities with its predecessor. A popular distribution of Linux is by Red Hat and the open source version is the Fedora Core now being managed by the Fedora organization: https://getfedora.org/.

The size of Linux is large and so the download of Linux is only practical with a high-speed internet connection. For example, the download speed of high-speed cable is currently from a few to 50 or more Mbps (megabits per second): 4 Mbps is approximately 0.5 MB/s (megabytes per second) = 512 kB/s (kilobytes per second). If the size of files to be downloaded is 2 GB = 2048 MB, then the time taken for the download with a constant download speed of 4Mbps is 2048 MB/(0.5 MB/s) = 4096 s = 68 minutes = 1.33 hours. Using a modem with speed, 56 kbps (kilobits per second) = 7 kB/s (kilobytes per second), the download time will be about 512/7 = 73 times longer than for high-speed cable, that is, it will take almost 32 days. Note that in the calculations above the prefixes mean the following: G, giga, means 2^{30} instead of 10^{9}, mega, M, means 2^{20} instead of $10^{6,}$ and kilo, k, means 2^{10} instead of 10^{3}.

1.8.2.2 Browser

With Windows the usual browser is Internet Explorer which is being replaced by Edge. Another popular browser is Firefox, derived from Mozilla, a free and open software and available for both Windows and Linux. It is important that the browser be a recent version, preferably the latest version, so that it complies with the latest HTML standards. Firefox is the most popular browser with Linux. Other popular browsers that are used with Linux are Opera, Konqueror, Seamonkey and Cliqz,

1.8.2.3 Languages

HTML (hypertext markup language) is the language of web pages and is supported by all browsers. CSS (cascading style sheets), which can be regarded as a distinct web page language, is considered to be part of HTML in Figure 1.10. JavaScript is the most popular client-side scripting language supported by both Internet Explorer and Firefox. JavaScript instructions are embedded in a web page, meaning that they are in the same file as the HTML instructions. The JavaScript instructions are examined by the browser and executed, if necessary, which may change the display of the web page on the screen. Typically, JavaScript instructions validate data that the user is putting in a form (by typing or using the mouse) and so JavaScript instructions in this case are executed as data is added to the form.

A form is created on a web page by HTML instructions and is used to receive information from the user. Another client-side scripting language is VBScript (Visual Basic Script) which is a Microsoft product that is only supported by Internet Explorer.

1.8.3 Server (Computer 2)

1.8.3.1 Operating System

A Windows or Linux operating system can be used. See the comments about Windows and Linux in 1.8.2 Client (Computer 1). Linux is used most often on a server because it is free and because Apache and PHP work well with Linux. Also, a Linux shell (e.g., Bash, Csh, Tcsh) is more useful than a Windows shell (called DOS Shell or Command Prompt depending on the version of Windows operating system).

1.8.3.2 HTTP (hypertext transfer protocol) Server

Apache is the most-used HTTP server. Apache is free software, which is part of the reason for its popularity. But it is also open source which means that the source code can be obtained by anyone and as a consequence thousands of programmers have contributed and are contributing to the development of Apache. As a result Apache is the most reliable and secure HTTP server.Apache can be downloaded from: http://www.apache.org.

1.8.3.3 Languages

PHP is a very popular server-side scripting language. PHP is also free software and open source. A PHP interpreter is attached to the Apache web server as a module, that is, the PHP interpreter is integrated as part of the Apache web server. PHP originally stood for, Personal Home Page tools. Now PHP stands for, PHP: Hypertext Pre-processor, an example of recursion in a name. PHP instructions are embedded in a web page, meaning that they are in the same file as the HTML instructions. When a web page is requested by a client, the server reads the web page from the hard disk to main memory of the server and then parses (analyzes) the web page. If PHP instructions are embedded in the web page, the instructions are sent to the PHP interpreter which executes the instructions. Then the output of the PHP instructions are put in the web page in place of the PHP instructions themselves. After this is done, the server sends the web page to the browser.

PHP has support for all major database systems, MySQL for example. For each database system, there are a set of functions which are used to access the database. The calls to these functions are called the application programming interface (API).
PHP can be downloaded from: http://www.php.net.

Other server-side scripting languages are ASP and JSP. ASP is Active Server Pages by Microsoft and can be written in different scripting languages. ASP comes with VBScript and JScript (the Microsoft implementation of JavaScript). Interpreters of other scripting languages can be installed and those languages can be used in ASP. JSP is JavaServer Pages by Sun and is written in Java.

1.8.4 Database (Computer 3)

1.8.4.1 **Operating System**

A Windows or Linux operating system can be used. See the comments about Windows and Linux in 1.8.2 Client (Computer 1).

1.8.4.2 **DBMS (database management system)**

MySQL is the most-used free DBMS. MariaDB is a recent version of the original MySQL code developed by the creator of MySQL There are a set of MySQL PHP functions which are executed to access the database. A popular DBMS among individuals and small organizations is Microsoft Access. One of the main features of Access is that almost all database operations can be done by using a graphical interface without the need to know SQL. Access is not free. The most popular commercial DBMS is Oracle, which is not free while MySQL/MariaDB is free.

1.8.4.3 Languages

SQL (structured query language) is a standard database language. A version of SQL is used by almost all relational DBMS's. SQL/PSM (SQL/persistent stored module) is a procedural language extension to SQL which is used to write procedures and triggers. SQL/PSM is a formal name used in the SQL standard maintained by ANSI (America National Standards Institute). In MySQL documentation the term, stored procedures, is used instead of the name, SQL/PSM. In Oracle, PL/SQL (procedural language/SQL) is an implementation of SQL/PSM.

1.8.4.4 Shell

The name of the MySQL/MariaDB shell is mysql. A shell is a program which is a command-line interpreter. When mysql is started, a command line appears where any SQL instructions can be typed and executed. Also using other commands, SQL scripts (files containing SQL instructions) can be produced and executed from the command line.
SQLPlus is the shell for Oracle.

1.9 Other Required Software

1.9.1 Editor

An editor is needed to type the web pages. Web pages are text files and so the editor must be a text editor. If the web pages are produced using Windows, the Windows text editors, Notepad or WordPad, can be used. If a word processor, for example Word, is used, the file must be saved as type, DOS text, because the control characters of the word processor must not appear in the file. If the web pages are produced using Linux, the popular text editor, emacs, can be used.

1.9.2 **Internet Applications**

There are two other programs, described as internet applications, which are needed, in general, for the development of a database web application: remote login and file transfer.

All internet applications involve communication between programs running on two different computers on the Internet. The terminology, client and server, is used to refer to the two computers.

1.9.2.1 Remote Login

The internet application, telnet, was the original program that was used to connect from one computer (client), which the user operates, to another computer (server) so that the client can use the resources (run programs for example) of the server, as if the user was sitting in front of a keyboard, mouse and screen connected directly to the server. Both computers must be running their versions of telnet. The user must have an account on the server and must have permissions granted to access the programs and files which are needed. Due to security problems with telnet, it is now replaced with ssh (secure shell). A version of ssh can be downloaded from: http://www.chiark.greenend.org.uk/~sgtatham/putty/

Examples:
(a) In Figure 1.10 suppose the user is also a web programmer and creates a web page. One way to do this is to login from the client (Computer 1) to the server (Computer 2). After the remote login, the keyboard, mouse and screen of the client interact with the server as if they were connected directly to the server. Then an editor on the server and the services of the operating system on the server can be used to type the web page and store it on the server hard disk.
(b) Suppose the user in Figure 1.10 is also a database programmer and wishes to use mysql (MySQL shell) on the Database computer (Computer 3) to run SQL commands to test a database. This is done in two steps: login to the server computer (Computer 2) from the client computer (Computer 1) and then login to the database computer (Computer 3) from the server computer (Computer 2). At the mysql prompt, SQL commands or SQL scripts (files containing one or more SQL commands) can be typed and executed.

1.9.2.2 File Transfer

The internet application, ftp (file transfer protocol) or scp (Secure copy), is used to exchange files between the user's computer (client) and another computer (server) over the Internet. Both computers must be running a daemon for the protocol: a daemon is a program that runs as a background process and responds to a request to the corresponding protocol. The user must have an account on the server and must have permissions granted to access the files and directories which are needed.

The terms, download and upload, identity the direction of file transfer. Download a file means to copy a file to the user's computer from some other computer. Upload a file means to copy a file from the user's computer to some other computer. A version of a file transfer program with a graphical user interface can be downloaded from,

http://winscp.sourceforge.net/eng/

Examples:
(a) In Figure 1.10 suppose the user is also a web programmer and creates a web page on the client using an editor and stores the file on the hard disk of the client. Then the web page must be stored on the server hard disk. The user can upload the web page from the client (Computer 1) to the server (Computer 2) with ftp or scp.
(b) Suppose the same user has many web pages on the server and wishes to make a backup of all the files on the client computer. The user can download the web pages from the server (Computer 2) to the client (Computer 1) with ftp or scp.

1.10 Postscript

In this chapter and text the reference to the term 'computer', for instance in the three-tier architecture with a 'computer' at each tier, needs some clarification. With the development of computing what a computer is has evolved. The 'computer' may or may not be a separate physical system but a software/hardware construct.

In today's computing environment on the internet, the two tiers for the servers could be on a computing service called a cloud hosted by an independent organization and can be used by a multitude of servers. The cloud computing is the virtualization of a service that was offered in the mid 1960’s called time-sharing. At that time organizations, instead of investing in their own 'main-frame' computing systems, shared computing resources on a system which allowed multi-programming and multi-tasking. This allowed many users to use a high cost main-frame computing system at reasonable expense. The time sharing service was offered by new entrepreneurs who invested in a computing structure. They, for a fee, allowed smaller organizations to connect to this computing system to do their computing. In this way, the latter need not make the capital investment for the hardware, nor incur the maintenance and human resources. The time-sharing was done initially in a batch environment and it evolved into 'real' time with the use of dedicated telecommunication lines. The use of time sharing thus lowered the cost of using computing capability. This promoted the interactive use of computers and the development of new applications.

The introduction of mini-computers and personal computers in the late 1960s and 1970s turned the computing paradigm to local systems. However, with increasing use of computers, new applications and on-line presence on the internet gave rise to a new type of time-sharing. This principal evolved into cloud computing which not only offers computing facilities but provides servers, both computing and storage, which appears to the end users as on-demand dedicated computing and storage systems.

The third tier, client, in many cases could be a mobile device, such as a cell phone or a tablet. These use one of a myriad of applications(apps) to access the higher level tiers

2 MySQL Shell

2.1 Introduction

MaraiDB and the older MySQL are the most popular open source relational database systems. Open source software can be used by anyone without cost. In contrast, Oracle is the most popular database management system which is not open source and therefore must be bought before it is used. However, Oracle is available free for personal use.

The MySQL database management system (DBMS) is a program which responds to requests to access the database, much like a web server responds to requests to access web pages. So the MySQL DBMS is a server program and is sometimes called the MySQL server or, simply, the database server.

There are two parts to this chapter: (1) the installation of MySQL and (2) the use of the MySQL shell.

(1) MySQL can be installed on a home computer so that web pages which access a database can be tested before transferring the web pages to a web site on a remote computer. However, it is not sufficient to download and install only MySQL. A web server and a server-side programming language (PHP is used) must also be installed with MySQL on the home computer. This is covered in the second section of this chapter

(2) The MySQL shell is one of two ways to access a MySQL database. The name of the MySQL shell is mysql. The MySQL shell, like an operating system shell, is a command-line interpreter program. It receives SQL commands typed on the keyboard and executes the commands. The use of the shell, mysql, is covered in the remainder of the chapter after the second section.

The other way to access a database is by a set of MySQL functions which communicate with a MySQL database. These functions are called in a PHP script. The MySQL functions are covered in a later chapter.

2.2 Installation of MySQL

MySQL together with other software can be installed on a home computer so that development of database-driven web applications can be done entirely on a home computer. The other software applications that are required are a web server and a server-side programming language. There are a number of software packages which install all of the applications at the same time.

EasyPHP is a free software package which implements WAMP on a Windows-based computer. WAMP is an acronym which stands for Windows, Apache, MySQL and PHP. If the operating system is Linux, then a LAMP (Linux, Apache, MySQL, PHP) installation must be done.

The installation of EasyPHP is described in the rest of this section.

2.2.1 EasyPHP

The web site for EasyPHP is at address: http://www.easyphp.org/

At the time of writing, the most recent version of windows-based EasyPHP currently called EasyPHP Devserver 17.0 has the following components: PHP 7.x / 5.5.x / 5.4.x / 5.3.x, Apache, Nginx, MySQL, PhpMyAdmin, Xdebug + modules + components. In order to understand the interaction of the components, phpMyAdmin, a graphical interface included in the package, is not used in this text.

2.2.2 Download

Before downloading EasyPHP, make a directory to store the downloaded file. For example, the directory could have the name, Downloads, and be in the root directory, C:. Downloads of other applications can also be stored in this directory. The download of EasyPHP is one executable file (a file with extension, .exe) with size of about 60 MB (http://www.easyphp.org).

2.2.3 Install

Before installing EasyPHP, make a directory with name, EasyPHP, in the root directory of the same partition as the Windows operating system, which is usually C. Install EasyPHP in this directory instead of the default installation directory (C:\Program Files (x86)\EasyPHP-Devserver-17, for the current version). It is better to have a shorter path for the installation directory, which requires less typing and is easier to remember. Start the installation by double clicking the executable file which was downloaded and stored in: C:\Downloads

The current version of EasyPHP requires about 392 MB of space on disk to install and run. Note that the third setup screen warns that WAMP, Windows-based EasyPHP, should be used only for development. LAMP (Linux, Apache, MySQL, PHP) should be used for production since Windows is not considered to be stable and secure enough. Click, install, in the last setup screen to start the installation.

2.2.4 Starting of servers

There are two servers: the web server, Apache, and the database server, MySQL. Both are programs which can be started either manually or automatically

.

2.2.4.1 Manual start

The Apache and MySQL servers are started by clicking EasyPHP in the EasyPHP start menu directory. This starts the executable file, C:\EasyPHP\EasyPHP.exe assuming that EasyPHP was installed in C:\EasyPHP. If EasyPHP is used regularly, a shortcut to EasyPHP can be put on the desktop for simpler starting. After the Apache and MySQL servers start, the EasyPHP status window shows a green traffic light for each server, Figure 2.1. Minimize the EasyPHP status window.

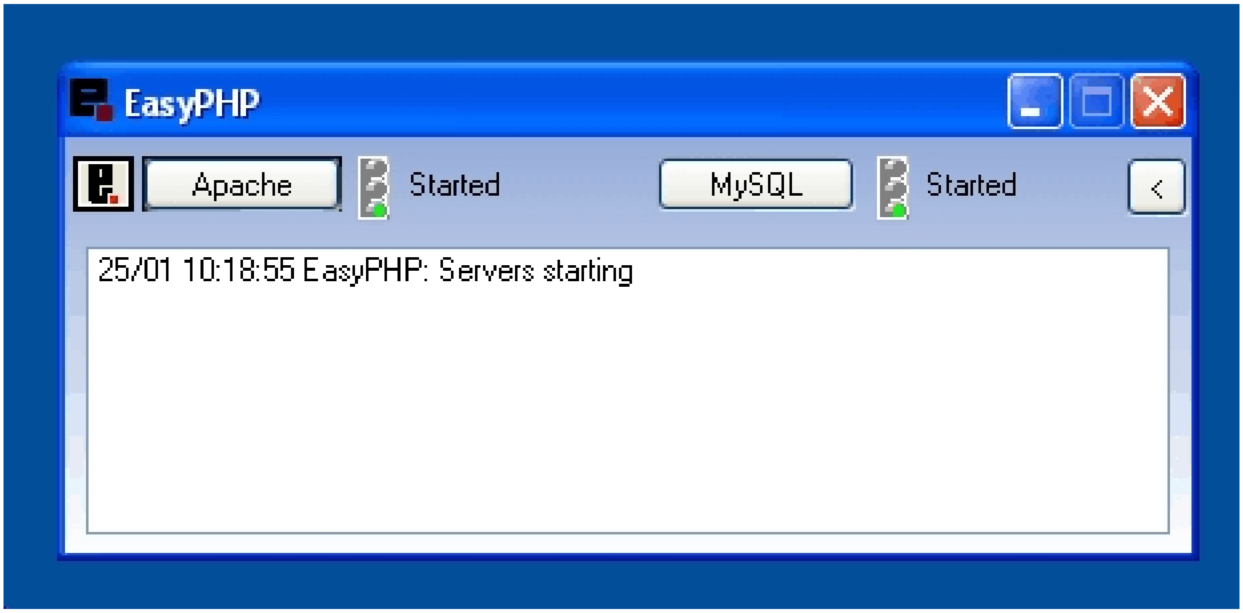
When EasyPHP is running, there is an EasyPHP icon (large e with a red dot) in the system tray (lower right side of the screen). Double click the icon in the system tray to see the status window again. The Apache or MySQL servers can be stopped and started individually by clicking the Apache or MySQL buttons and selecting Start or Stop. If both servers are stopped, EasyPHP does not stop and so the EasyPHP icon remains in the system tray. To stop EasyPHP, click X in the status window or right click the icon in the system tray and then click, Exit.

2.2.4.2 Automatic start

The Apache and MySQL servers can be started automatically each time Windows starts by selecting the option to do this as follows. Right click the icon in the system tray and then click,

Configuration/EasyPHP

In the configuration window click the box, Start on windows startup, so that a check appears.



```
25/01 10:18:55 EasyPHP: Servers starting
```

Figure 2.1 Status window for servers

2.2.5 Name of local computer

A browser, Internet Explorer for example, is installed on a computer called the client computer, or client for short. The client computer is also called the local computer. The web server, Apache, is installed on another computer called the server computer, or server for short. The browser communicates with the web server by using a name which identifies the computer on which the web server is installed.

When the web server and browser are installed on the same computer, as in the case of EasyPHP, the default name of the server computer is: localhost. This is the name which is set in the configuration file for Apache, httpd.conf, on the line: ServerName localhost

This name represents the local computer address, also called the loopback address, which has the IP address: 127.0.0.1The location of the configuration file, assuming that EasyPHP was installed in C:\EasyPHP is: C:\EasyPHP\apache\conf\httpd.conf

The configuration file, httpd.conf, can be opened with a text editor, Notepad for example, and configuration names such as ServerName can be found by searching with the text editor. Do not change the configuration file!

2.2.6 Document root directory

Web pages which are served by Apache, or any other web server, must be stored in the document root directory or in subdirectories of the document root directory.The document root directory is the root directory of the web site. It has the default name: www

The name of the document root directory, assuming that EasyPHP was installed in C:\EasyPHP, is set in the configuration file for Apache, httpd.conf, on the line:

```
DocumentRoot "C:/EasyPHP/www"
```

.
This can be confirmed by opening httpd.conf with a text editor, as explained in the last subsection. The value of DocumentRoot is the absolute path of the document root directory,

C:\EasyPHP\www

To confirm that www is in C:\EasyPHP, use Windows Explorer.

2.2.7 Browser

Any browser, for example, Internet Explorer or Firefox, can be used to request web pages from the Apache server.

Suppose that a file, *filename*, is stored in a directory, *Directoryname*, which is a subdirectory of the document root directory. So the absolute pathname of the file is

C:\EasyPHP\www\Directoryname\filename

The page is displayed in the browser by typing in the address bar of the browser,

http://localhost/*Directoryname/filename*

Alternatively, the IP address, 127.0.0.1, can be typed instead of localhost,

http://127.0.0.1/*Directoryname/filename*

2.2.8 Test

Test the installation of Apache and PHP as follows.

Use a text editor, Notepad for example, to make a file with the following contents. Name the file, hello.php.

```
<?php echo "<h1>hello, world</h1>"; ?>
```

Save the file in the document root directory. So the absolute pathname of the file is

C:\EasyPHP\www\hello.php

assuming that EasyPHP was installed in C:/EasyPHP.

Using the browser, request the page, hello.php, by typing in the address bar of the browser:

http://localhost/hello.php

The output of the browser is: **hello, world**

Do not forget to start the Apache server before requesting a web page.

2.3 Alternate Downloads

There are other sites for downloading software for WebDB development. An additional site for the current platform is: https://www.apachefriends.org This site has a package that requires minimum configuration.

A user of some of the Linux distributions could use the software install command such as *apt-get, dnf* or *yum* to download the various components individually. For example, for the Fedora Core distribution the series of steps are: install the software, update the configuration files, enable the software and start the services(daemons).

```
dnf -y install httpd;     dnf -y install mariadb;    dnf-y groupinstall php
```

The usual place for the configuration files for these software servers are in the directories:

```
/etc/httpd;     /etc/my.cnf.d;     /etc/php.d
```

The system wide configuration for apache, maraiadb and php are in the files:

```
/etc/httpd/conf/httpd.conf;     /etc/my.cnf;     /etc/php.ini
```

These files could be edited to update the location of the various directories. For example, the home directory for the web could be changed from the default to: /home/WWW and /homeWWW/secure

The commands to enable and start the daemons for Apache and MariaDB are:

```
systemctl enable mariadb
systemctl enable httpd
systemctl start mariadb
systemctl start httpd
```

2.4 mysql

The program, mysql, is a shell which allows the user to type SQL (Structured Query Language) and other commands on the command line of the program. The shell, mysql, in MySQL corresponds to the shell, SQLPlus, in Oracle. Both shells have the same roles as shells for operating systems, such as tcsh and bash in UNIX and Linux.

There are two kinds of interfaces to MySQL: text-based interface and graphical user interface (gui). The program, mysql, is the text-based interface, also called command-line interface, where the user communicates with MySQL by typing on the keyboard. The other kind of interface, gui, displays commands in a window and the user executes the commands by clicking the commands with the mouse, and thereby communicates with MySQL. There is a gui available for MySQL but it will not be used here.

The documentation for the mysql shell is in Section 4.5.1 of the MySQL 5.1 Reference Manual. The internet address of the manual is given in the last section, References.

2.4.1 Location of mysql

The shell program executable file, mysql.exe, is stored in the directory, bin, which has absolute path,

```
C:\EasyPHP\mysql\bin
```

assuming that EasyPHP was installed in C:\EasyPHP.
Using Windows Explorer, verify that mysql.exe is in this directory.

2.4.2 Start operating system shell

The shell program, mysql, is started from the operating system shell program. In Windows the shell is called the DOS shell in early versions of Windows and Command Prompt in later versions.
In Windows XP, for example, start the operating system shell by clicking

Start/Programs/Accessories/Command Prompt

The shell prompt will be the root directory or a subdirectory of the root directory followed by the greater than symbol, >. Assume the shell prompt is

```
C:\>
```

2.4.3 Search path

The directory which contains mysql must be added to the path which is searched by the operating system shell when a program name is typed at the prompt of the operating system shell. Type the command, help, at the shell prompt to display a list of all commands available in the operating system shell. Press the Enter key to execute the command,

```
C:\> help
```

Syntax and other information about a command is displayed by typing the name of the command after help,

C:\> help *commandname*

For example, for the command, set, type,

```
C:\> help set
```

Type the command, set, to display a list of all environment variables,

```
C:\> set
```

Note the value of the path variable. If it does not include the path to mysql, the path must be added.
Add the path to mysql to the existing path with the command, set, which follows. The variable, %path%, is the value of the existing path. The individual paths are separated by a semicolon (;). Do not type spaces before and after the equal sign.

```
C:\> set path=%path%;C:\EasyPHP\mysql\bin
```

Type the command, set, again to check that the path variable is correct.

```
C:\> set
```

In Windows XP the change to the path variable is not permanent, that is, when the operating system shell is stopped, the changes made with set disappear. Stop (type command, exit) and then start the shell. List the environment variables and look at the value of the path variable,

```
C:\> set
```

To permanently change the value of the path variable in Windows XP, click Start/Control Panel/ System. Then in the Systems Properties window, click the tab, Advanced, and then click, Environment Variables. In the Environment Variables window, click the System variable, Path, and then click, Edit. Add to the end of the path variable,

```
C:\EasyPHP\mysql\bin
```

and then click OK in each of the open windows.

Stop and start the operating system shell and list the environment variables to verify that the change to the path variable has been made,

```
C:\> set
```

2.4.4 Start mysql

To start mysql, type at the operating system shell prompt, assumed to be C:\>, the command,

```
C:\> mysql -h hostname -u username
```

The name of the shell program is mysql.exe but it is not necessary to type the extension, .exe.

If the server runs on the same computer on which the user logs on, it is not necessary to specify hostname to start mysql. Then mysql can be started with the command,

```
C:\> mysql -u username
```

The username, root, was created when MySQL was installed and can be used.

Start mysql using the username, root. Do not forget to start the MySQL server first.

```
C:\> mysql -u root
```

The prompt of mysql is,

```
mysql>
```

Type the command, help, to display a list of all mysql internal (built-in) commands,

```
mysql> help
```

2.4.5 Stop mysql

To stop mysql, type the command, exit, at the prompt of the mysql shell. The command, quit, also stops mysql.

```
mysql> exit
```

2.4.6 Stop operating system shell

To stop the operating system shell, type the command, exit, at the prompt.

```
C:\> exit
```

2.5 Sample database

The sample database consists of one table which is used to introduce the basic MySQL shell commands. This is the same table as used in Chapter 1 Introduction, but with four columns instead of two. The schema (structure) of the table and an instance (contents) of the table are given in Figure 2.2, together with the ER diagram. The ER diagram in this simple case is just a graphical representation of the schema of the table. ER diagrams are introduced in the next chapter.

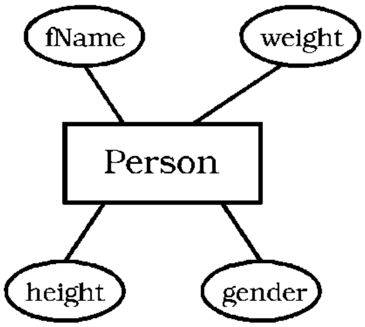
2.6 Demonstrations of mysql commands

The program, mysql, is a shell program. A mysql command is an order to mysql to perform a particular task. The command is given by typing the name of the command, and arguments (other information after the name), if required, on the command line at the prompt, mysql>. The command is executed by pressing Enter. When the cursor is flashing in front of the prompt, mysql is ready to receive a command.

There are (a) built-in commands of mysql and (b) SQL commands. The built-in commands of mysql are executed by functions in the mysql program. Some built-in commands access the database and some do not. The SQL commands are sent to the MySQL server which executes the commands and sends output, if any, back to mysql for display on the screen. All SQL commands interact with the database.

The shell program, mysql, is not case sensitive meaning that instructions can be typed in uppercase or lowercase or a combination of both. In some books, it is conventional to type all keywords of commands in uppercase so that the command is easy to read, but this is not done here. The understanding of the commands should not be based on the case which is used. Lowercase is used almost exclusively for ease of typing. However, some arguments of commands are case sensitive. For example, filenames are case sensitive; for example, the file with name, file1, is not the same file as the file with name, File1

ER Diagram



Instance

Person

fName	weight	height	gender
Tom	160	1.78	m
Dick	145	1.73	m
Harry		1.93	m
Jane	105	1.57	f

Schema

Person(fName, weight, height, gender)

Figure 2.2 Sample database

This section demonstrates common mysql commands using the simple database in Figure 2.2. Some of the common SQL commands are demonstrated, since the main purpose of the MySQL shell is to execute SQL commands. The SQL commands in this chapter are used without explanation. They are systematically covered in the next chapter. A recent version of the MySQL Reference Manual covers all SQL commands in Chapter 12, SQL Statement Syntax, of the manual.

The MySQL Reference Manual contains a tutorial which demonstrates the use of mysql. The tutorial is Chapter 3, Tutorial, of the manual. The internet address of the manual is given in the last section, References. Not all commands used in the tutorial of the MySQL Reference Manual are demonstrated here, so it is advisable to look at the tutorial after studying this section.

2.6.1 Start mysql

Confirm that MySQL server is running. Double click the EasyPHP icon in the system tray to display the status window. When a green traffic light is shown beside the MySQL button, the

MySQL server is running. Click the MySQL button to start the server, if necessary. Then minimize the status window.

Start the operating system shell. In recent versions of Windows click

Start/All Programs/Accessories/Command Prompt

Type at the prompt, assumed to be C:\>, the command to start mysql using the username, root. If a password is required, type it when requested.

```
C:\> mysql -u root
```

When the prompt, mysql>, appears, a connection has been made to the MySQL server.

2.6.2 General: version(), user(), now(), evaluate expression, cancel command

A command can be typed on one line or more than one line. A semicolon (;) must be typed at the end of the last line for most commands. After the semicolon, Enter is pressed to execute the command.

The up arrow can be used to display each line of the previous commands that were typed. If a command is typed on only one line, the command can be executed again by using up arrow to display the command instead of retyping the command and then pressing Enter.

Functions, a name ending with (), and operators are covered in Chapter 11, Functions and Operators, of the MySQL Reference Manual.

2.6.2.1 version()

As a first example of a command, display the version number of the MySQL server, using the function, version(). The command is, select version().

```
mysql> select version();
```

The results are displayed in a table with one column which has the name, version(), and one row of data which has the version number.

2.6.2.2 user()

The function, user(), displays the user (yourself) which is displayed by the command, select user().

```
mysql> select user();
```

The results are displayed in a table with one column which has the name, user().

Display a different name for the column by using an alias.

```
mysql> select user() as "Me";
```

2.6.2.3 now()

Another useful command displays the present date and time, provided by the function, now(). Note that the date and time is from the clock of the MySQL server. If the server is in a different time zone than the user, the date and time won't be the same as at the location of the user.

```
mysql> select now();
```

The results are displayed in a table with one column which has the name, now().

Display a different name for the column by using an alias.

```
mysql> select now() as "Date and Time";
```

Note that the international standard format of date and time is used: year-month-day hour:minute:second.

2.6.2.4 version(), user(), now()

The output of more than one function can be displayed at the same time as follows.

```
             mysql> select version(), user(), now();
```

The results are displayed in a table with three columns which have the names, version(), user() and now().

A command can be typed on more than one line, for example, using four lines.

```
     mysql> select
             user(),
              now();
```

Display different column names by using aliases.

```
     mysql> select version() as "MySQL Version",
             user() as "Me",
             now() as "Date and Time";
```

The output of the last two commands, on a Linux system using Mariadb is displayed in Figure 2.3.

2.6.2.5 evaluate expression

The area of a circle in terms of its diameter is an example of an expression: Area = $\pi d^2/4$, where d is the diameter of a circle. The area of a circle with d = 1.23 is evaluated. The expression for the area of a circle includes the function, pi(), which returns the value of π, and the function, pow(x, y), which returns the value of x raised to the power of y.

```
     mysql> select pi()*pow(1.23,2)/4
              as AArea of circle with diameter = 1.23@;
```

2.6.2.6 cancel command, \c

A command can be cancelled before the command is executed by typing, \c, and then Enter. For example, a command can be cancelled on the third line as follows.

```
     mysql> select
             user();
```

2.6.3 Database: show, create, drop, use, database()

2.6.3.1 show

Display all databases.

```
             mysql> show databases;
```

2.6.3.2 create

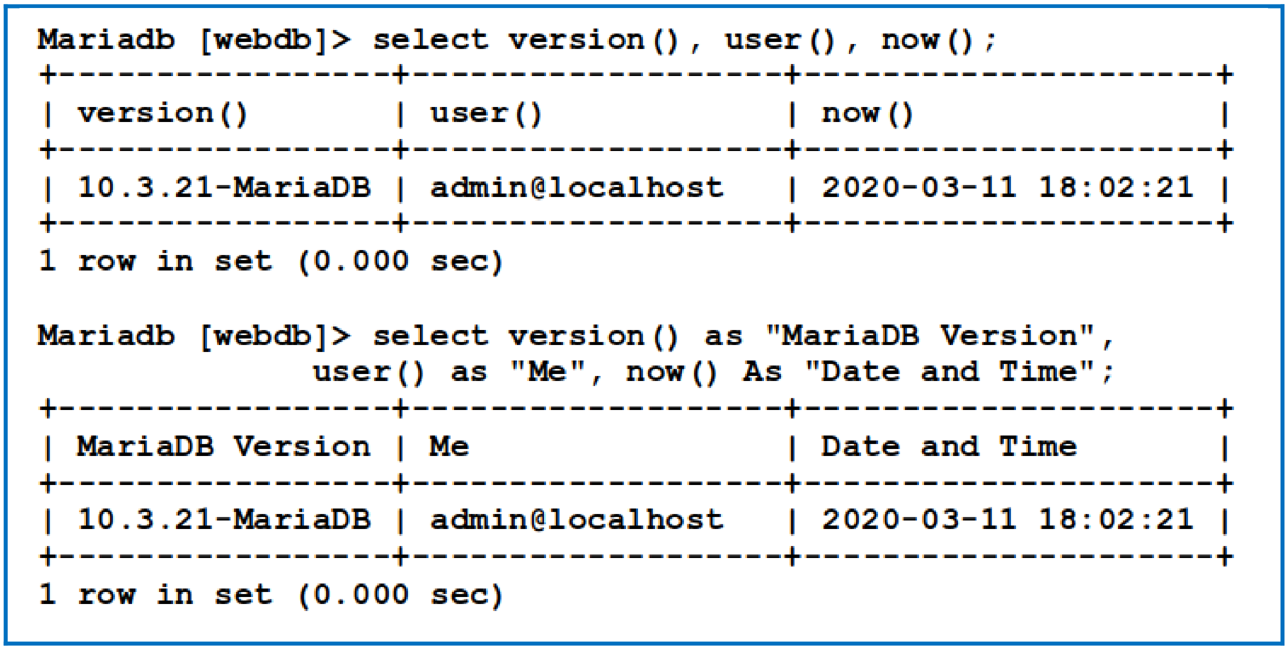
Create a database with name, webdb.

```
             mysql> create database webdb;
             mysql> show databases;
```

Create a database with name, db2.

```
             mysql> create database db2;
```

```
mysql> show databases;
```



```
Mariadb [webdb]> select version(), user(), now();
+-----------------+-------------------+---------------------+
| version()       | user()            | now()               |
+-----------------+-------------------+---------------------+
| 10.3.21-MariaDB | admin@localhost   | 2020-03-11 18:02:21 |
+-----------------+-------------------+---------------------+
1 row in set (0.000 sec)

Mariadb [webdb]> select version() as "MariaDB Version",
              user() as "Me", now() As "Date and Time";
+-----------------+-------------------+---------------------+
| MariaDB Version | Me                | Date and Time       |
+-----------------+-------------------+---------------------+
| 10.3.21-MariaDB | admin@localhost   | 2020-03-11 18:02:21 |
+-----------------+-------------------+---------------------+
1 row in set (0.000 sec)
```

Figure 2.3 Display of command: select version(), user(), now()

2.6.3.3 drop

Delete the database with name, db2.

```
mysql> show databases;
mysql> drop database db2;
mysql> show databases;
```

The drop database command deletes the database and all of its contents, that is, all tables in the database

2.6.3.4 use

Choose to use the database, webdb. The command, use, is one of the commands which are not terminated by a semicolon. However, a semicolon can be typed after the command, as below, with the same result.

```
mysql> use webdb;
```

2.6.3.5 database()

Display the database which is currently being used. This is provided by the function, database().

```
mysql> select database();
```

The output of the commands, use and select database(), are shown in Figure 2.4.

2.6.4 Table: show, create, drop, describe

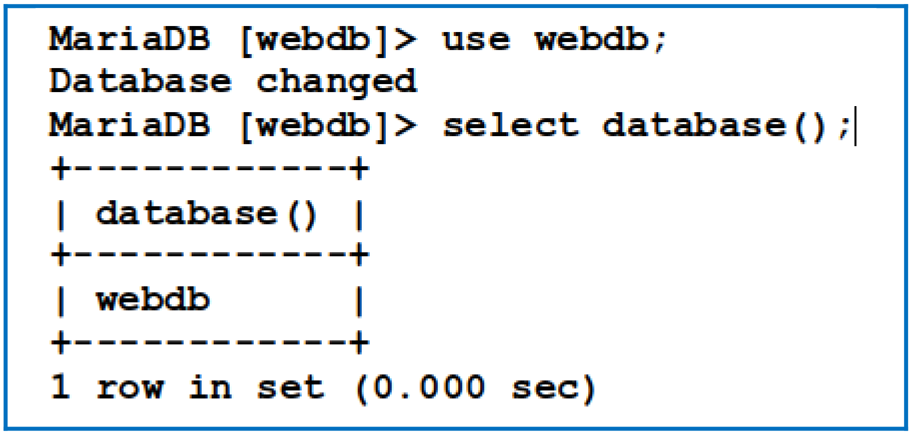
2.6.4.1 show

Choose webdb as the current database.

```
mysql> use webdb;
```

Display the names of all tables in database, db1. Assuming it is a new database, it does not contain any tables. The result is the message, Empty set.

```
mysql> show tables;
```



```
MariaDB [webdb]> use webdb;
Database changed
MariaDB [webdb]> select database();
+------------+
| database() |
+------------+
| webdb      |
+------------+
1 row in set (0.000 sec)
```

Figure 2.4 Display of commands: use and select database()

2.6.4.2 create

2.6.4.2.1 New table

Create the table, Person, in Figure 2.2. The data types of the columns are shown in the create table command below. The column, fName, is a string which can have a maximum length of 10 characters, weight is an integer with three digits, height is a real number with one digit before the decimal point and two digits after the decimal point and gender is a string with one character (f for female and m for male). The column, fName is declared to be a primary key and gender is declared to be not null. A primary key must have different values in each row of the table and is automatically, not null. In the instance of the database in Figure 2.2, one of the first names is Tom. If another person with the same name is inserted into the database, the name entered would have to be Tom2, for example.

From now on the prompt for the second and subsequent lines of a command, ->, will be omitted.

```
mysql> create table Person (
           fName varchar(10) primary key,
           weight integer(3),
           height float(3,2),
           gender char(1) not null);
```

Confirm that the table has been created by displaying the tables in the database. Now one table, Person, is listed in the database, webdb.

```
mysql> show tables;
```

2.6.4.2.2 Copy table

Create a copy of table, Person. This is the simplest way to make a backup of a table. The command will copy both the structure of the table and the contents of the table. However, the command will not declare the primary key in the copied table, as shown by the Describe command below. The asterisk, *, in the command means, all columns.

```
mysql> create table PersonBackup
```

```
                select * from Person;
mysql> show tables;
```

The output of the commands, create table and show table, for tables, Person and PersonBackup, are shown in Figure 2.5.

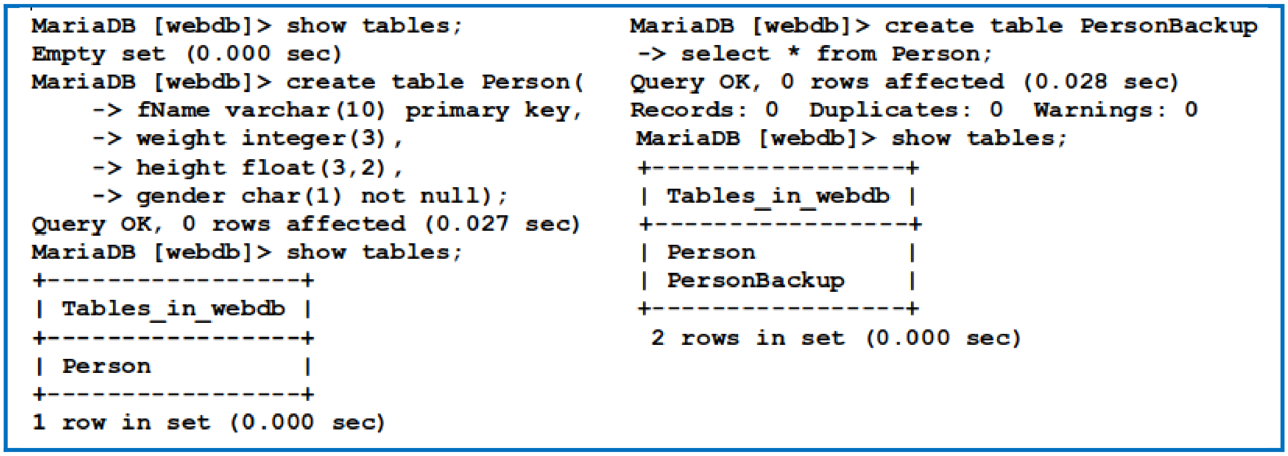
Create another copy of table, Person.

```
mysql> create table PersonBackup2
                select * from Person;
mysql> show tables;
```

2.6.4.3 drop

Delete the table with name, PersonBackup2.

```
mysql> show tables;
mysql> drop table PersonBackup2;
mysql> show tables;
```



```
MariaDB [webdb]> show tables;
Empty set (0.000 sec)
MariaDB [webdb]> create table Person(
    -> fName varchar(10) primary key,
    -> weight integer(3),
    -> height float(3,2),
    -> gender char(1) not null);
Query OK, 0 rows affected (0.027 sec)
MariaDB [webdb]> show tables;
+-----------------+
| Tables_in_webdb |
+-----------------+
| Person          |
+-----------------+
1 row in set (0.000 sec)
```

```
MariaDB [webdb]> create table PersonBackup
 -> select * from Person;
Query OK, 0 rows affected (0.028 sec)
Records: 0  Duplicates: 0  Warnings: 0
 MariaDB [webdb]> show tables;
 +-----------------+
 | Tables_in_webdb |
 +-----------------+
 | Person          |
 | PersonBackup    |
 +-----------------+
  2 rows in set (0.000 sec)
```

Figure 2.5 Display of commands: create table, show tables

2.6.4.4 describe

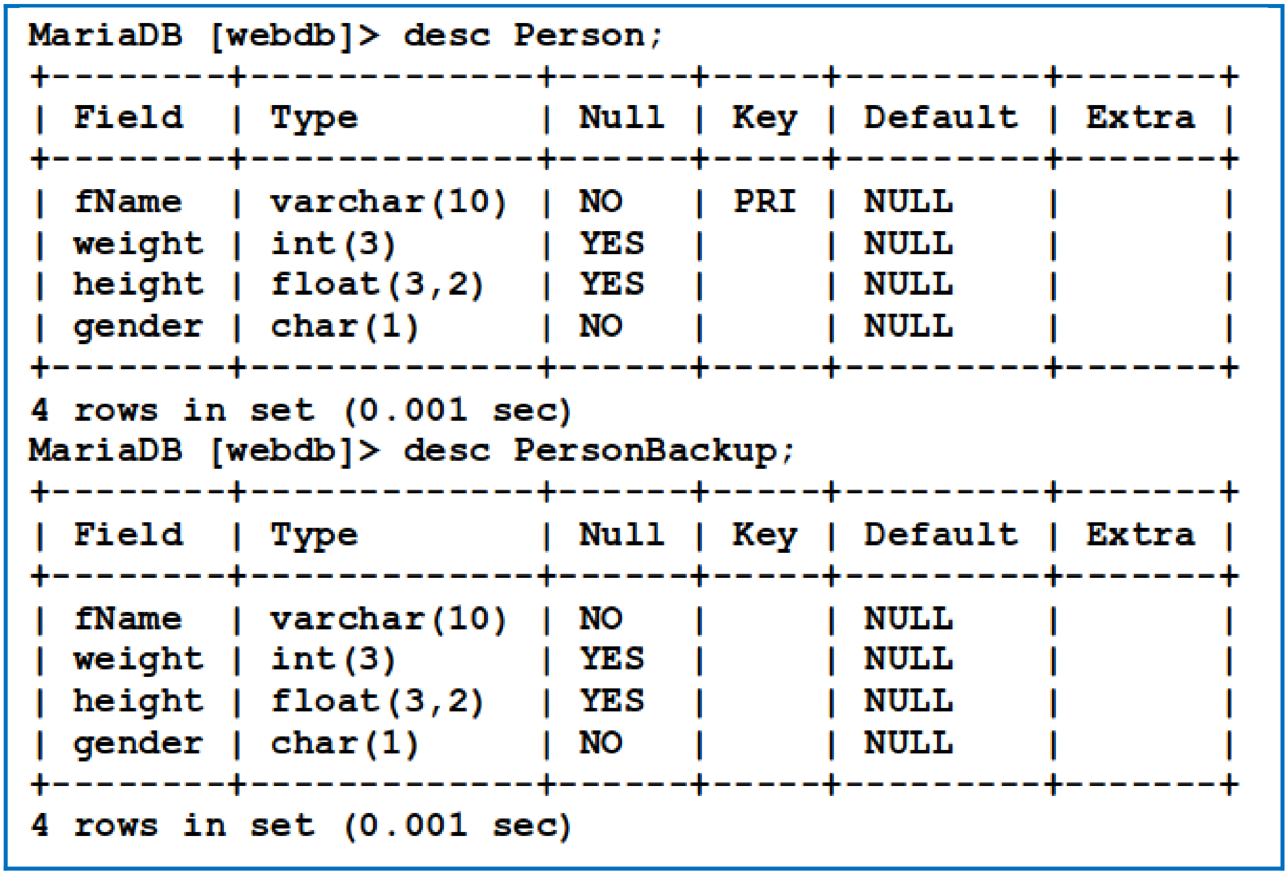
Display the properties of the columns of Person using the command, describe, with short form, desc, or the equivalent command, show columns from

```
mysql> desc Person;
mysql> show columns from Person;
```

Display the properties of the columns of PersonBackup, the table which was copied from Person. Note that the declaration of the primary key was not copied from Person to PersonBackup.

```
mysql> desc PersonBackup;
```

The output of the command, describe, for tables, Person and PersonBackup, are shown in Figure 2.6.



```
MariaDB [webdb]> desc Person;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| fName  | varchar(10) | NO   | PRI | NULL    |       |
| weight | int(3)      | YES  |     | NULL    |       |
| height | float(3,2)  | YES  |     | NULL    |       |
| gender | char(1)     | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.001 sec)
MariaDB [webdb]> desc PersonBackup;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| fName  | varchar(10) | NO   |     | NULL    |       |
| weight | int(3)      | YES  |     | NULL    |       |
| height | float(3,2)  | YES  |     | NULL    |       |
| gender | char(1)     | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0.001 sec)
```

Figure 2.6 Display of command: describe

2.6.5 Table: select, insert, update, delete

2.6.5.1 select

Display the data in all rows and columns of table, Person. Since the table was just created, it is empty, and the message, Empty set, is displayed. The asterisk, *, in the command means, all columns.

```
mysql> select * from Person;
```

2.6.5.2 insert

Populate the table, Person, with the data in the instance of the table in Figure 2.2.

```
mysql> insert into Person values
        ('Tom', 160, 1.78, 'm'),
        ('Dick', 145, 1.73, 'm'),
        ('Harry', null, 1.93, 'm'),
        ('Jane', 105, 1.57, 'f');
```

Display all data in the table, Person. There are four rows of data each with four columns, except the weight for Harry which is missing and is designated, NULL.

```
mysql> select * from Person;
```

The output of the command, insert, for table, Person, is shown in Figure 2.7.

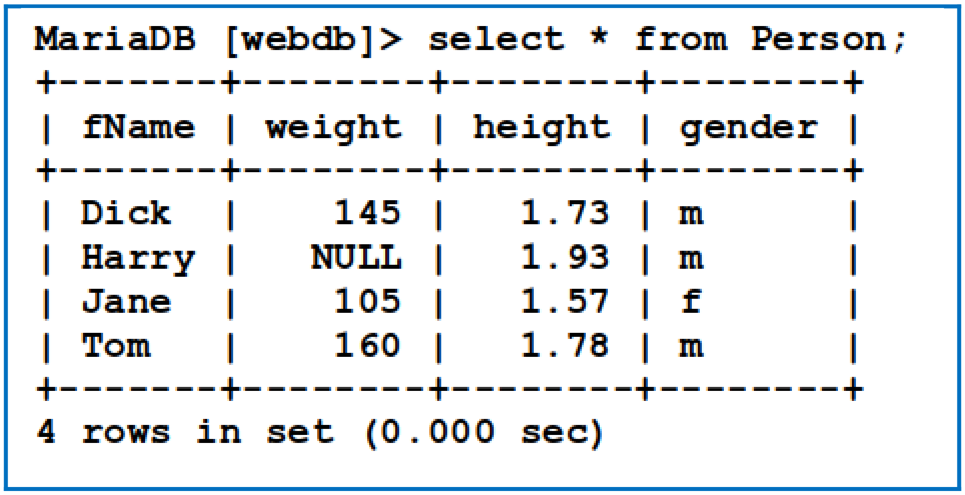
2.6.5.3 update

Modify the data for Harry by inserting the weight, 200.

```
mysql> update Person
           set weight = 200
           where fName = 'Harry';
```

Display all data in the table, Person, to verify that the weight of Harry was changed from NULL to 200.

```
mysql> select * from Person;
```



```
MariaDB [webdb]> select * from Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| Tom   |    160 |   1.78 | m      |
+-------+--------+--------+--------+
4 rows in set (0.000 sec)
```

Figure 2.7 Display of command: insert

2.6.5.4 delete

Remove the row for Harry.

```
mysql> delete from Person
           where fName = 'Harry';
```

Display all data in the table, Person, to verify that the row for Harry is missing.

```
mysql> select * from Person;
```

Add the row for Harry again, with the weight unknown.

```
mysql> insert into Person values
           ('Harry', null, 1.93, 'm');
```

Display all data in the table, Person, to verify that the row for Harry is present.

```
mysql> select * from Person;
```

Try to add a row for another person with the name, Tom. This is not allowed since fName is a primary key.

```
mysql> insert into Person values
           ('Tom', 175, 1.85, 'm');
```

(error, due to primary key for fName)

Try to add a row for another person without specifying the gender. This is not allowed since gender has the constraint, not null.

```
mysql> insert into Person values
```

```
                              ('Sue', 115, 1.62, null);
```

(error, due to not null for gender)

2.6.6 Table: null values

The term, null value, is commonly used but it is misleading since null is not a value. Null means absence of a value. Therefore, the six relational operators, **=, <>, >=, >, <=, <,** cannot be used with null. Instead the two operators,

```
     is null, is not null
```

are used.

Display the rows for which the weight is null.

```
     mysql> select * from Person where weight = null;     (error)
     mysql> select * from Person where weight is null;
```

Display the rows for which the weight is not null.

```
     mysql> select * from Person where weight <> null;    (error)
     mysql> select * from Person where weight is not null;
```

2.6.7 Rows: select

Display only the rows for which the weight is greater than 140 with all columns. Note that this does not include the row for Harry since the weight is unknown.

```
          mysql> select * from Person
                    where weight > 140;
```

Display only the rows for females.

```
          mysql> select * from Person
                    where gender = 'f';
```

2.6.8 Rows: sort

Display all rows and sort the rows by fName in alphabetical order.

```
          mysql> select * from Person order by fName;
```

Display all rows and sort the rows by fName in reverse alphabetical order. The keyword, desc, for descending is used.

```
          mysql> select * from Person order by fName desc;
```

2.6.9 Rows: count

Count the total number of rows in table, Person. The result includes the rows which have null in one or more columns. Since there is one person per row in this particular table, the count of the number of rows is the number of persons in the table.

```
          mysql> select count(*) from Person;
```

Count the total number of rows in table, Person, and name the column, Number of rows.

```
          mysql> select count(*) as “Number of rows” from Person;
```

Count the number of males and females in the table. This command uses, group by, which is covered in the next chapter.

```
          mysql> select gender, count(*) from Person group by
gender;
```

Count the number of rows in table, Person, which have values for weight. The result does not include the row which has null for the weight.

```
mysql> select count(weight) from Person;
```

2.6.10 Columns: select

Select only the columns, fName and gender, for all rows.

```
mysql> select fName, gender from Person;
```

Select only the column gender to make sure that there are only two genders, f and m. All rows are displayed and so the same genders are displayed more than once.

```
mysql> select gender from Person;
```

Select only the column gender and display only unique rows, that is, display each value of gender only once. The keyword, distinct, is used.

```
mysql> select distinct gender from Person;
```

2.6.11 Pattern matching

In SQL a single character is matched by an underscore, '_', and zero or more characters are matched by '%'. The two operators,

```
like, not like
```

are used instead of = (equal) and <> (not equal). The patterns are not case sensitive, by default.

Display all names which start with the letter, d. Dick is displayed.

```
mysql> select * from Person where fName like 'd%';
```

Display all names which do not start with the letter, d. Tom, Harry and Jane are displayed.

```
mysql> select * from Person where fName not like 'd%';
```

Display all names which contain the letter, a. Harry and Jane are displayed .

```
mysql> select * from Person where fName like '%a%';
```

Display all names which contain four letters. This pattern is specified by four underscores. Dick and Jane are displayed.

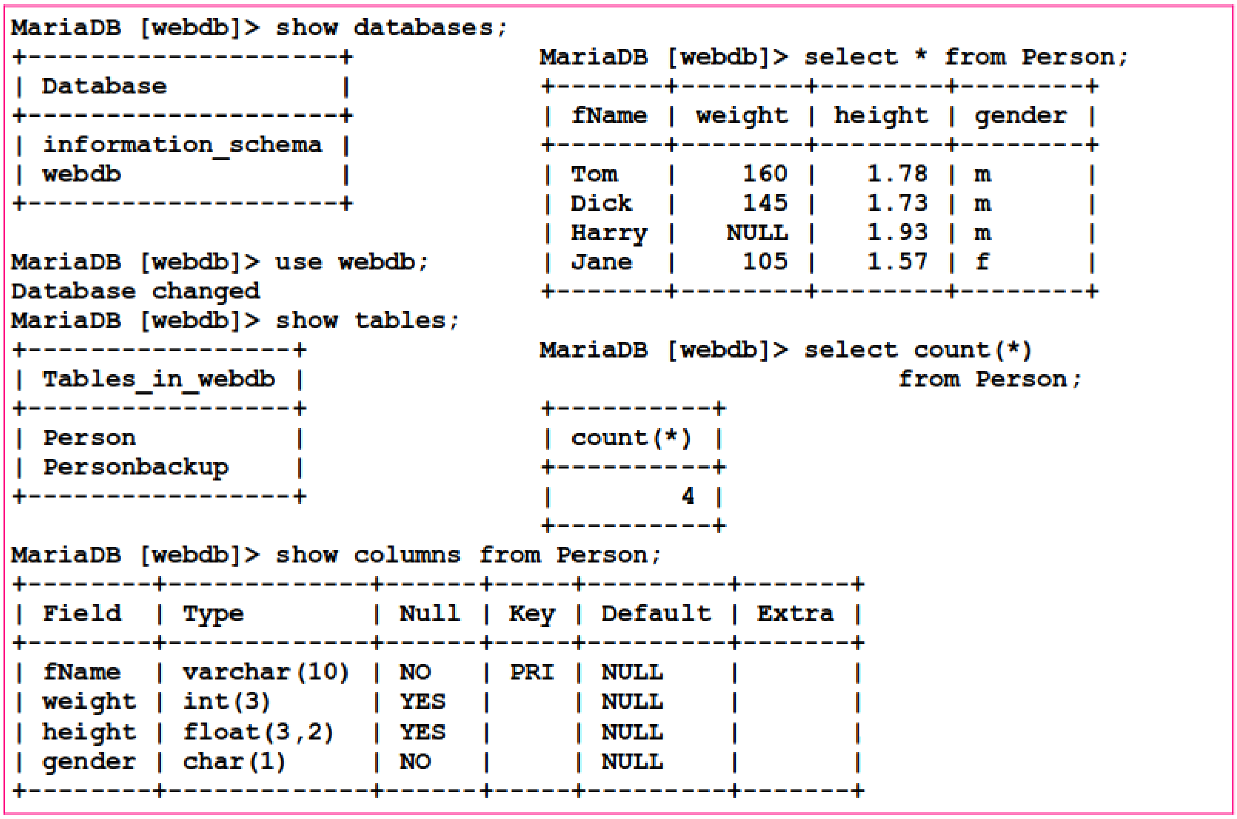
```
mysql> select * from Person where fName like '____';
```

2.6.12 Batch mode

Batch mode means to execute a file, instead of a single command. The file is called a batch file or script file or script, and it contains one or more mysql commands. A batch file is a text file and so must be created by a text editor. The simplest text editor in Windows is Notepad. Comments in a batch file start with # anywhere in a line and terminate at the end of the same line.

If the file which contains the mysql commands has absolute path and name, *path/filename*, the file is executed by the command \. (\ followed by .) or source.

```
mysql> \. path/filename
mysql> source path/filename
```



```
MariaDB [webdb]> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| webdb              |
+--------------------+

MariaDB [webdb]> use webdb;
Database changed
MariaDB [webdb]> show tables;
+-----------------+
| Tables_in_webdb |
+-----------------+
| Person          |
| Personbackup    |
+-----------------+

MariaDB [webdb]> show columns from Person;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| fName  | varchar(10) | NO   | PRI | NULL    |       |
| weight | int(3)      | YES  |     | NULL    |       |
| height | float(3,2)  | YES  |     | NULL    |       |
| gender | char(1)     | NO   |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

```
MariaDB [webdb]> select * from Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
+-------+--------+--------+--------+

MariaDB [webdb]> select count(*)
                        from Person;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
```

Figure 2.8 Display of command: source.

Type the following batch file using Notepad. Save the file with name, batch1.sql, in the root directory, C:\. The name of the file can have any extension, or none at all, but it is advisable to use the extension, .sql, as an indication that the file contains SQL command.

(Beginning of file)

```
#name of file: batch1.sql

#display databases
   show databases;
#choose a database
   use webdb;
#display tables
   show tables;
#display structure of table, Person
   show columns from Person;
#display contents of table, Person
   select * from Person;
#display total number of rows in table, Person
   select count(*) from Person;
```

(End of file)

Execute the batch file. Do not type a semicolon, ;, at the end of a command to execute a batch file; this is an error.

```
mysql> source C:\batch1.sql
```

The output of the command, source, is shown in Figure 2.8

3 SQL Basics

3.1 Introduction

Structured Query Language (SQL, pronounced as it is spelled) is the language used to access databases. SQL became the standard for relational database management systems in the 1980's. It was developed as a language for the relational database model but over the years enhancements have been made to it to extend it to Object and Object relational models and for the web and XML (Extended Markup Language). SQL has commands to perform tasks to create tables of a database and manipulate (add, remove, modify and retrieve) data in a database. Most commercial relational database management systems use SQL, and each adds extensions that are usually limited to that particular system.

SQL is not a language that is compiled, like C or C++ for example. Rather it corresponds to a scripting language like a shell program for Linux, where each command executes code that performs a task. SQL commands can be put into a file, called an SQL file or script, and then the file can be executed. The commands in the file are executed one after the other.

3.2 SQL Commands

A database has two aspects: container and contents. Accordingly, the SQL commands are divided into two basic parts, called the data definition language (DDL) and data manipulation language (DML). DDL defines the database container and DML manipulates the database contents. There is a third category of SQL commands, called the data control language (DCL), which is the rest of the commands in addition to the DDL and DML commands. DCL controls access to the database. All SQL commands must be terminated by a semicolon (;) when commands are typed at the prompt on the command line of the MySQL shell.

The three categories of SQL commands, a classification of the commands in each category and the first one or two keywords of the common commands are given in Figure 3.1.

The SQL commands in Figure 3.1 are listed in Table 3.1 with a brief description of each command followed by the keywords of the command. The keywords of the commands in Table 3.1 should be enough to remind an experienced user of the syntax of the commands. The detailed syntax and demonstrations of the commands are given in this chapter.

References are given in the last section of the chapter and includes web sites which introduce SQL: W3Schools and SQL.org. The former offers tutorials on many web subjects, including SQL, which are intended for beginners. SQL.org has information about SQL including introductory tutorials. Other references are the web site for the MySQL and MaraiDB knowledge base and reference manual..

SQL Commands

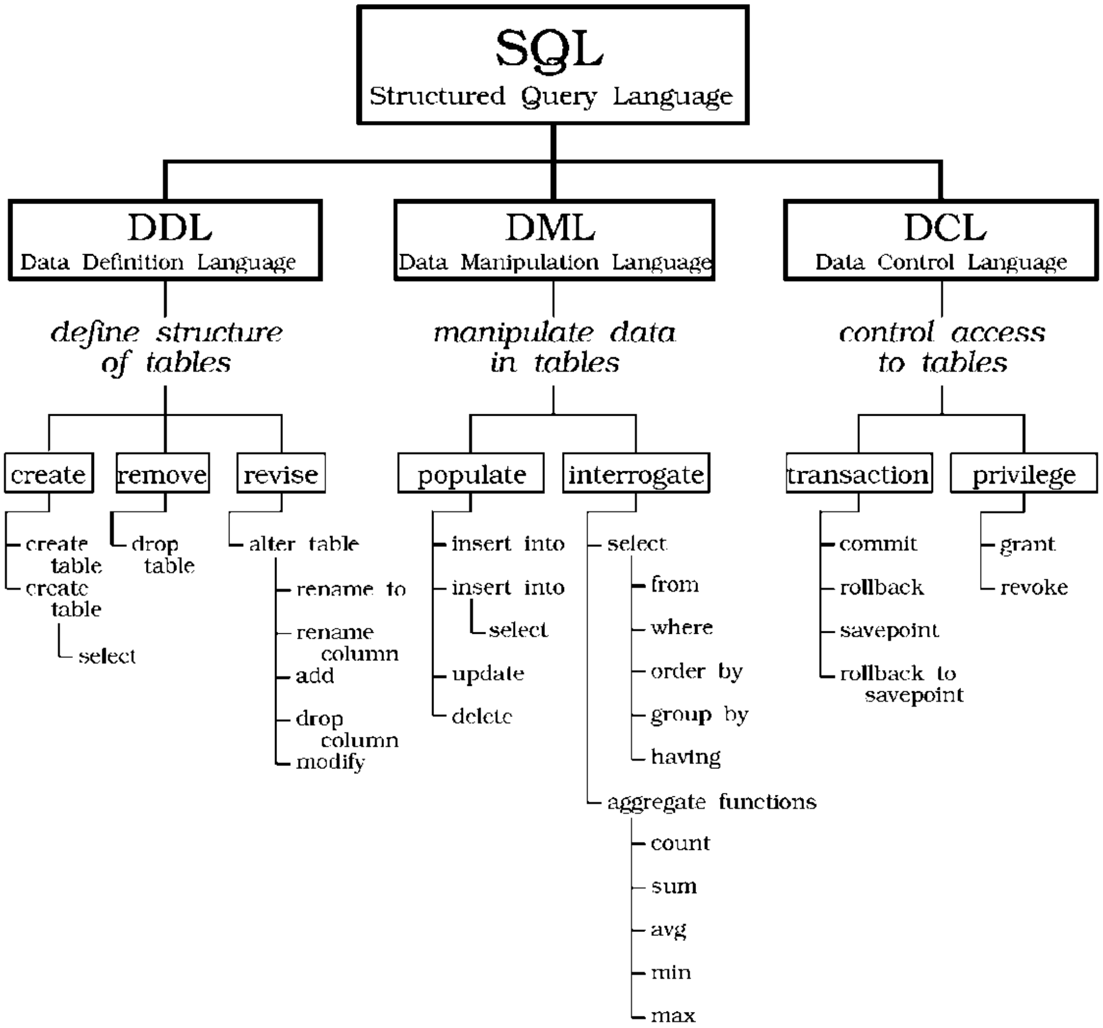


Figure 3.1 SQL commands

3.2.1 Data Definition Language (DDL)

DDL commands create and modify the container of the database. The container of a relational database is one or more tables. A table is specified by a structure which is called a schema. There is one schema for each table, and it gives the name of the table and the names of the columns (attributes) of the table.

The common DDL commands are CREATE TABLE, ALTER TABLE and DROP TABLE

Category	Description	Description **Command**
DDL	Define table, remove table	create a table: CREATE TABLE . . . delete a table: DROP TABLE . . .
	revise table	change the name of a table ALTER TABLE . . . RENAME TO . . . add one column to a table ALTER TABLE . . . ADD . . . remove one column from a table ALTER TABLE . . . DROP COLUMN . . . change the name of a column of a table ALTER TABLE . . . RENAME COLUMN . . change the data type of a column of a table
	reproduce table	copy a part or all of a table CREATE TABLE . . . AS SELECT . . . FROM . . .

Table 3.1 DDL

Category	Description	Description **Command**

DML	populate table	add one new row in a table
		INSERT INTO . . . VALUES . . .
		add one or more rows from one table into another table
		INSERT INTO . . . SELECT . . . FROM . . .
		modify one or more rows in a table
		UPDATE . . . SET . . . WHERE . . .
		remove one or more rows from a table
		DELETE FROM . . . WHERE . .

	interrogate table	retrieve data from a table general case, [ ] means optional: SELECT . . . FROM . . . [WHERE . . . GROUP BY . . . HAVING . . . ORDER BY . . .] retrieve <u>all</u> rows with some or all columns from a table SELECT . . . FROM . . . retrieve <u>some</u> rows with some or all columns from a table SELECT . . . FROM . . . WHERE . . . retrieve <u>some</u> rows with some or all columns from a table using wildcards SELECT . . . FROM . . . WHERE . . . LIKE . . . retrieve <u>some</u> rows with some or all columns from a table and sort according to one or more columns SELECT . . . FROM . . . WHERE . . . ORDER BY . . .

Table 3.1 (continued) DML 1 of 2

Category	Description	Description **Command**

DML	interrogate table	retrieve <u>some</u> rows with some or all columns from a table and sort according to one or more columns
		SELECT . . . FROM . . . WHERE . . . ORDER BY . . .
		retrieve some rows with some or all columns from a table and sort according to one or more columns
		SELECT . . . FROM . . . WHERE . . .
		ORDER BY . . .
		retrieve aggregate numbers of all rows for one or more groups
		SELECT . . . FROM . . . GROUP BY . . .
		retrieve aggregate numbers of some rows for one or more groups
		SELECT . . . FROM . . . GROUP BY . . .HAVING . . .

	Aggregate functions	number of rows in a table with a value for a particular column
		COUNT(*columnName*)
		sum of all values of a particular column
		SUM(*columnName*)
		average of all values of a particular column
		AVG(*columnName*)
		minimum value in a particular column
		MIN(*columnName*)
		maximum value in a particular column
		MAX(*columnName*)

Table 3.1 (continued) DML 2 of 2

Category	Description	Description **Command**
DCL	manage transactions	save changes to database on disk COMMIT undo changes to database since last COMMIT ROLLBACK mark a point SAVEPOINT *point* undo changes to database to *point* ROLLBACK TO SAVEPOINT *point*
	administer privileges	allow access to a table GRANT . . . remove access to a table allowed by GRANT REVOKE . . .

Table 3.1 (continued) DCL

3.2.1.1 Constraints

DDL also creates constraints, which are restrictions on the data that can be stored in the database. There is only one mandatory constraint and that is called the domain. A domain is a data type with a restricted range of values. Each column in a table must have a domain. Three common domains are used in this chapter which have keywords: CHAR, VARCHAR and DECIMA.L. The domains with keywords, CHAR and VARCHAR, are one or more characters (a string) and the domain with keyword, DECIMA.L, is a real number (either whole or fractional). An example of the restricted range of values of a string is the domain, VARCHAR(10), which means a string with a maximum of 10 characters. An example of the restricted range of values of a number is, DECIMAL(3), which means an integer with a maximum of three digits.

All other constraints are optional. An example of a common optional constraint is the null constraint. If the constraint of a column of a table is declared, NOT NULL, all rows in the table must have a value in the column. Only the constraint, domain, is introduced in this chapter. All other constraints are introduced in the next chapter.

3.2.2 Data Manipulation Language (DML)

DML commands populate (add, remove and modify contents of) and interrogate (retrieve contents of) the database. The content is data (information about something) which is stored in the container (one or more tables) of the database.

There are only four basic DML commands: INSERT, DELETE, UPDATE and SELECT. INSERT adds data; DELETE removes data; UPDATE modifies data; SELECT retrieves data.

3.2.3 Data Control Language (DCL)

DCL commands manage transactions and administer privileges.

COMMIT and ROLLBACK are the basic commands that manage transactions. A transaction is defined in the section, DCL Demonstrations. GRANT and REVOKE are the commands that administer privileges.

3.3 Domain Constraints

A table consists of one or more attributes (columns) and each attribute has a domain (data type with a restricted range of values). Four domain constraints are introduced in this section; two are strings and two are numbers. They have the following syntax and each is described below.

CHAR(*size*): maximum *size* is 255 (2^8-1) characters, minimum *size* is 0 character

VARCHAR(*size*): maximum *size* is 65535 (2^{16}-1) characters, minimum *size* is 0 character

DECIMAL(*precision*, *scale*): exact numeric data type
precision range is 1 to 65 digits, *scale* range is 0 to 30 digits

DOUBLE: approximate numeric data type
accuracy is approximately 15 digits

Note that in MySQL documentation, the term, data type, is used instead of domain. However, the data type in a programming language does not mean the same as a domain in SQL because the range of values of a data type in a programming language is not restricted by the declaration of the data type. So domain is the preferred term to use in SQL.

In the rest of this chapter the term, data type, is used instead of domain, because the MySQL documentation uses the term, data type. All of the data types in MySQL are given in the MySQL documentation with the internet address in section, References, at the end of this chapter

3.3.1 String types

3.3.1.1 Fixed-length string

The data type for a fixed-length string in MySQL is CHAR(*size*). The maximum length of the string is *size* characters where each character is a byte. The maximum value of *size* is 255 and the minimum value is 0. The length of the string, *size*, must be specified. If the value of an attribute has length greater than *size* characters and if strict SQL mode is enabled, an error is caused. If the value of an attribute with CHAR(*size*) data type has length less than *size* characters, spaces are added to the end of the value so that the length is always *size* characters; MySQL documentation calls this right-padding. A zero-length string is right-padded to one character instead of *size* characters. When the value of string of data type, CHAR(*size*), is retrieved, the trailing spaces are removed.

3.3.1.2 Variable-length string

The data type for a variable-length string in MySQL is VARCHAR(*size*). The maximum length of the string is *size* characters where each character is a byte. The maximum value of *size* is 65535 and the minimum value is 0. The length of the string, *size*, must be specified. If the value of an attribute has length greater than *size* characters and if strict SQL mode is enabled, an error is caused. If the value of an attribute with VARCHAR(*size*) data type has length less than *size* characters, only the characters of the value are stored; spaces are not added to the end of the value (as is done for CHAR(*size*)). So generally, VARCHAR(*size*) uses less memory than CHAR(*size*). Values of strings with data type, VARCHAR(*size*), are stored with a length prefix before the characters in the string. The length prefix stores the number of bytes in the string, that is, the value of *size*. The length prefix is one byte if *size* is 255 or less, and is two bytes if *size* is greater than 255.

Examples of the number of bytes of storage for CHAR(4) and VARCHAR(4) which is required for strings of length 0 (zero-length string) to 5 bytes is given below.

Value of data	Storage	
	CHAR(4)	VARCHAR(4)
' '	4 bytes	2 bytes
'a'	4 bytes	2 bytes
'ab'	4 bytes	3 bytes
'abc'	4 bytes	4 bytes
'abcd'	4 bytes	5 bytes
'abcde'	error	error

The bytes stored for string, 'ab', in the case of CHAR(4) are the characters, a and b, followed by 2 spaces, for a total of four bytes. The bytes stored for string, 'ab', in the case of

VARCHAR(4) are the length prefix byte which has value 2 followed by the characters, a and b, for a total of three bytes. The string of five characters, 'abcde', causes an error when inserted in a data type of size 4 when strict SQL mode is enabled.

3.3.2 Numeric types

3.3.2.1 Fixed-point number

A fixed-point number data type is an exact numeric data type.

The data type for a fixed-point number is DECIMAL(*precision*, *scale*) where *precision* is the total number of digits in the number, which is also called the number of significant digits, and *scale* is the number of digits after (to the right of) the decimal point. DEC, NUMERIC and FIXED are synonyms for DECIMAL. Note that leading zeros in a number are not part of the *precision*. For example, the *precision* of 0.00123 is 3. The *precision* has range 1 to 65 digits and the *scale* has range 0 to 30 digits. If a value of a number exceeds the maximum value (or is less than the minimum value) specified by *precision*, MySQL returns an error. If a value exceeds the *scale*, MySQL rounds the number. For example, if the value of a number is 1234.5678 and the data type is DECIMAL(*precision*, *scale*), the data stored for different values of *precision*, *p* and *scale*, *s* is given below.

Value of data	DECIMAL(*p*, *s*)	Data stored
1234.5678	DECIMAL(8, 4)	1234.5678
1234.5678	DECIMAL(8, 2)	1234.57
1234.5678	DECIMAL(8, 0)	1235
1234.5678	DECIMAL(8, -2)	1200
1234.5678	DECIMAL(7, 4)	error

Note that an integer is specified by *scale* = 0, third example above. The notation, DECIMAL(*p*), can be used instead of DECIMAL(*p*, 0) to specify an integer. A negative *scale* rounds a number to a position to the left of the decimal point, for example, *scale* = -2 rounds to the nearest multiple of one hundred, fourth example above. When the value exceeds the *scale*, the number is rounded as in the second, third and fourth examples above. When the value exceeds the *precision*, an error occurs as in the fifth example above.

3.3.2.2 Floating-point number

A floating-point number data type is an approximate numeric data type.

There are two data types for a floating-point number: FLOAT and DOUBLE. DOUBLE and REAL are synonyms for DOUBLE PRECISION. The synonym, DOUBLE, is used here, instead of DOUBLE PRECISION. The range of values for DOUBLE is
-1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and
2.2250738585072014E-308 to 1.7976931348623157E+308.

A value of data type, DOUBLE, is accurate to approximately 15 significant figures.

There is also the non-standard syntax, FLOAT(*precision*, *scale*) and DOUBLE(*precision*, *scale*), where *precision* and *scale* have the same meaning as for a fixed-point number. Values of data type, FLOAT, are stored in four bytes. Values of data type, DOUBLE, are stored in eight bytes.

3.4 Identifiers

Identifiers are names which identify objects in a database. User-defined names are, for example, table names and attribute (column) names. The word, name, instead of the word, identifier, will be used.

3.4.1 Valid Names

Names can consist of uppercase letters (A . . . Z), lowercase letters (a . . . z), digits (0. . . 9) and underscore (_). In particular, a name cannot contain a space. A name must start only with a letter. Examples of valid names are: FirstName, firstName, first_Name, firstName2. Examples of invalid names are First Name, _firstName, 2firstName. SQL is not case sensitive. For example, FirstName and firstName represent the same object. However, a character string is case sensitive. The string ‘tom’ is not the same as the string ‘Tom’. Also when comparing a string value to an attribute of type CHAR or VARCHAR, the case of the string value is significant. For example firstName='tom' and firstName='Tom' are not the same values of attribute, firstName. Note that MySQL requires that strings be enclosed in single or double quotes.

3.4.2 Name Conventions

In this text, the following conventions are used for the case of names, so that the names are easier to read. Since SQL is not case sensitive, the names can be typed in lowercase for convenience at the mysql prompt.

3.4.2.1 Keywords

All characters are uppercase. Keywords include names of commands and domain (data type) names.
Examples of keywords: CREATE TABLE, VARCHAR

3.4.2.2 Tables

The first character is uppercase. The name is a noun and the noun is singular.
Examples of tables: Person, UndergraduateStudent

3.4.2.3 Relationships

The first character is uppercase. The name is a verb or a phrase containing a verb, and the verb is third-person singular.
Examples of relationships: Gives, GivenBy

3.4.2.4 Attributes

The first character is lowercase. The name is a noun and the noun is singular.
Examples of attributes: name, firstName, fName

3.4.2.5 Domains

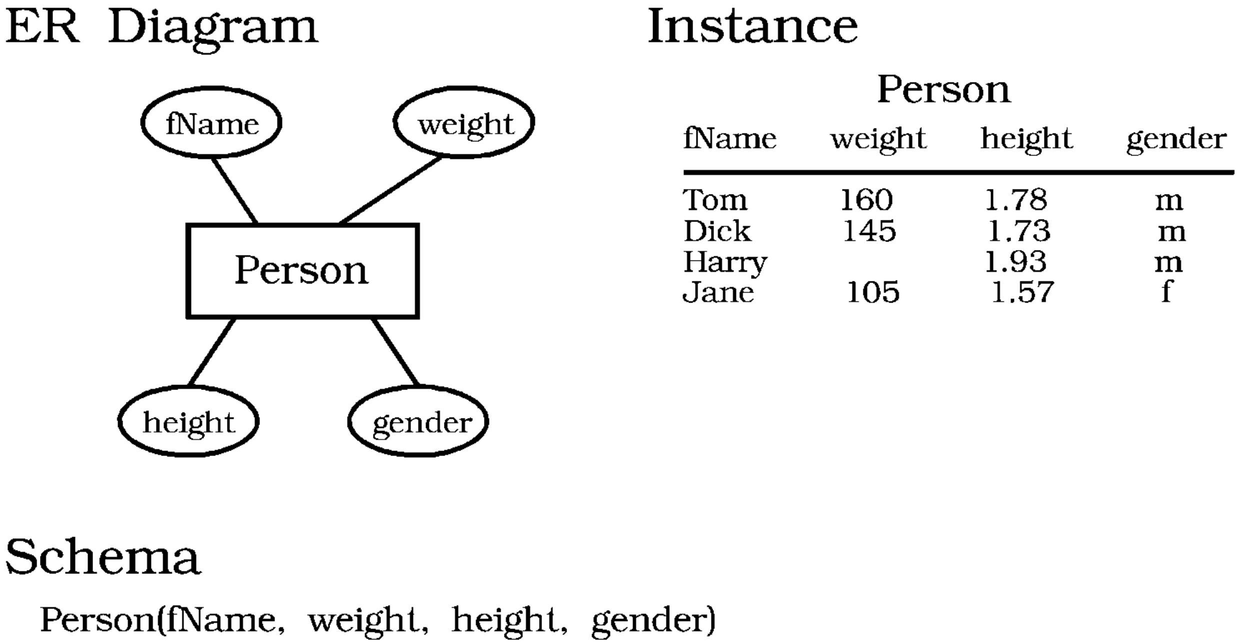
The first character is uppercase. The name is similar to the corresponding name of the attribute.
Example of domain: FirstNameType for attribute, firstName

3.4.2.6 Compound names

A compound name consists of two or more words. The first character of the second, third, . . . words of the name is uppercase. There are no spaces in a compound name.
Examples of attributes: firstName, theFirstName
Example of tables: UndergraduateStudent.

3.5 Sample Database

A sample database that consists of one table is used to demonstrate the basic SQL commands. The basic concepts of a database are introduced in Chapter 1.



Instance

Person

fName	weight	height	gender
Tom	160	1.78	m
Dick	145	1.73	m
Harry		1.93	m
Jane	105	1.57	f

Schema

Person(fName, weight, height, gender)

Figure 3.2 Sample database

3.5.1 ER diagram

The entity-relationship (ER) diagram for the table (entity) is shown in Figure 3.2. There is only one entity in this database, Person. The meaning of the entity is conveyed by the name chosen for the entity. The entity has four attributes whose meaning is conveyed by the names chosen for the attributes. There are no relationships in this database since there is only one entity.

3.5.2 Schema

The schema for the entity is

Person(fName, weight, height, gender)

As expressed by the schema, the table of the entity has four columns: fName, weight, height, and gender.

3.5.3 Instance

An instance of the data in Person is also shown in a table in Figure 3.2. The table consists of four columns with the headings for the attributes of the schema for Person. The data consists of four rows, one for each of four distinct persons with first names: Tom, Dick, Harry, Jane. The weight of Harry is unknown and so that column for Harry is blank.

3.5.4 Constraints

Each attribute has a domain, which restricts the value of the attribute. The domain constraint is mandatory for every attribute. Attributes can have other constraints which are optional. The only optional constraint used in this chapter is, NOT NULL. All optional constraints are covered in the next chapter.

3.5.4.1 Domain constraint

The domains of two of the attributes, fName and gender, are a string, which is a sequence of one or more characters. The first name of a person, fName, is a string of the letters of the alphabet (a-z,A-Z) and zero or one hyphen (-). The domain, VARCHAR(10), will be used initially for the attribute, fName. This domain is a string of 1 to 10 ASCII characters which includes all of the characters that can be typed on a keyboard (all the letters, all the digits (0-9), punctuation characters, and so on). So it will be necessary to further restrict the characters in fName by programming methods, which are covered in a later chapter.

The gender of a person is either male or female and is represented by a string with one letter: m, for male, and f, for female. The domain, CHAR(1), will be used for gender. This domain is a string of one ASCII character which can be any character typed on the keyboard. The domain of gender can be further restricted to m or f by the CHECK constraint, which is covered in the next chapter.

Of course, two people can have the same first name and so this table would have to permit names like Tom and Tom2, which however is not the best way to distinguish among people with the same name. The best way is to give a different id (identification) number to each person and to add the attribute (column), idNumber, to the table to include the id number of the person in the table. This is not done here.

The domains of the other two attributes, weight and height, are numbers. The domain of these quantities is obviously not all of the real numbers, -infinity to +infinity, no matter what units are used for weight and height. Domains of fixed-point numbers are expressed as numbers which have a maximum value which is a power of 10 minus 1 and an accuracy which is represented by the number of significant digits in the value of the number. The domain, DECIMAL(3), is used for weight in pounds. This is a number with three significant digits, with no digits after the decimal point. Therefore, weight is an integer (a whole number), with a maximum value of 999 (10^3 - 1) and a minimum value of -999 (-(10^3 - 1)) , that is, a domain of -999 to 999. The weight of a person cannot be negative and cannot be less than a few pounds at birth and so the value of weight must be further restricted, which can be done with the CHECK constraint covered in the next chapter.

The domain, DECIMAL(3,2), is used for height in meters. This is a number with three significant digits and two digits after the decimal point. So height is a real number (a number that can have a fractional part) with a maximum value of 9.99 and minimum value of -9.99, a domain of -9.99 to 9.99. This domain must be restricted to, for example, 0.25 to 2.99, which can also be done with the CHECK constraint.

3.5.4.2 Null constraint

From the meaning of the table and its attributes, it is not acceptable to have a row of data in the table without a value for first name, fName. A row of data with a value of weight, height and gender but no first name would not be useful since a person is identified by a name, and not by weight or height or gender. However, a row of data with a value of first name but with missing values for the other attributes could be useful since the missing data could be added later.

The gender of person is a basic characteristic of a person, in contrast to weight and height. So a table with a value of first name but not a value of gender may not be useful. So each row of data should have a value of both first name and gender.

The constraint, NOT NULL, is used for attributes, fName and gender. This constraint does not permit a row of data without a value for fName and gender.

3.6 DDL (Data Definition Language) Demonstrations

The Data Definition Language of SQL is the commands which are used to define the database structure (schema) and revise the structure as the database evolves. The container of data is one or more tables (relations of the relational model). A table is constructed by using the CREATE TABLE command of SQL. The structure of a table is revised (changed) by the ALTER TABLE command. A table is deleted by the DROP TABLE command.

All commands are executed by the MySQL shell, mysql, which has the prompt, mysql>. The prompt is not given in the text of the chapter but is shown in the screen captures of mysql in the figures.

At the mysql prompt, display the databases. The database and table commands which follow are introduced in the previous chapter.

```
SHOW DATABASES;
```

Chose database, webdb, assuming it exists. Otherwise create the database and then chose it.

```
USE webdb;
```

Display the tables in webdb.

```
SHOW TABLES;
```

It is assumed that the tables which are created below, People and Person, do not exist. If the tables exist, delete the tables.

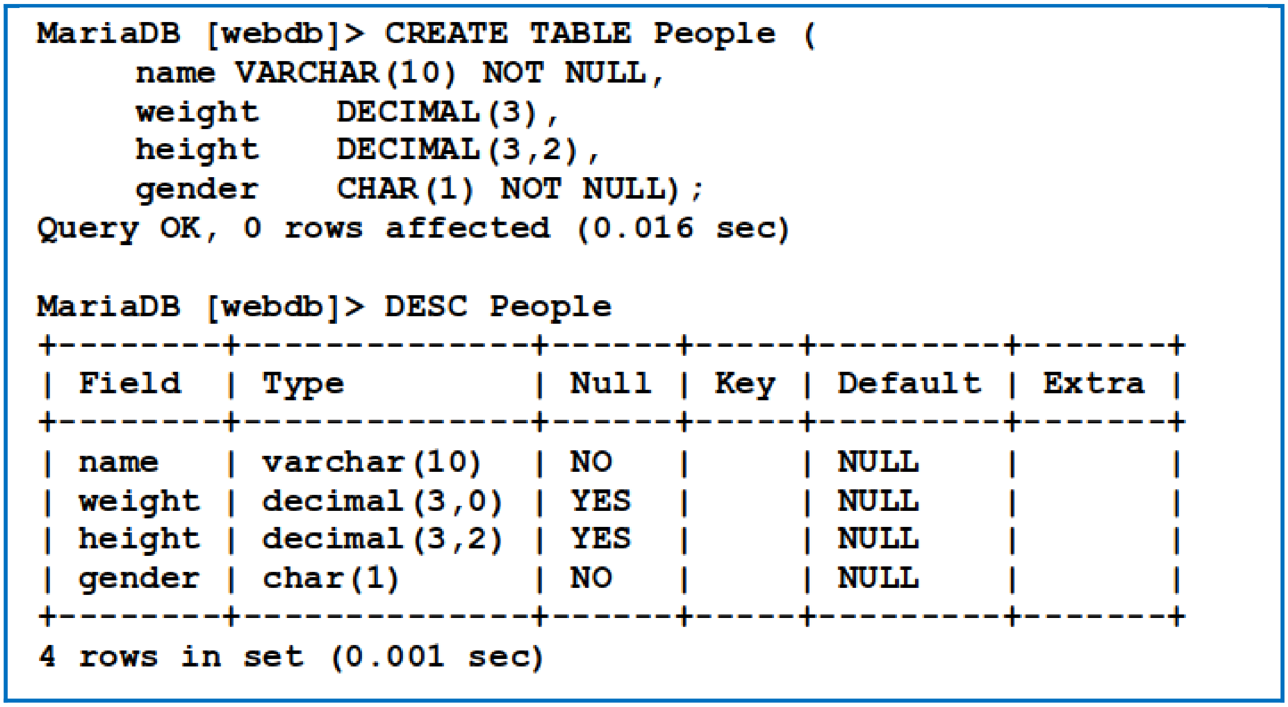
3.6.1 Create new table

Syntax:

CREATE TABLE *tableName* (
    columnName1 columnDefinition1,
    columnName2 columnDefinition2,
    . . .

columnNameN columnDefinitionN);

The clause, *columnDefinition*, contains the data type of the column and, optionally, the constraints of the column, for example, NOT NULL and PRIMARY KEY.



```
MariaDB [webdb]> CREATE TABLE People (
     name VARCHAR(10) NOT NULL,
     weight    DECIMAL(3),
     height    DECIMAL(3,2),
     gender    CHAR(1) NOT NULL);
Query OK, 0 rows affected (0.016 sec)

MariaDB [webdb]> DESC People
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| name   | varchar(10)  | NO   |     | NULL    |       |
| weight | decimal(3,0) | YES  |     | NULL    |       |
| height | decimal(3,2) | YES  |     | NULL    |       |
| gender | char(1)      | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.001 sec)
```

Figure 3.3 DDL command: CREATE TABLE

The CREATE TABLE command is used to create a new table with one or more columns. The name of the table which is created is *tableName*. The name of the table must be a unique name, that is, each table in the database must have a different name. The command defines *N* columns of the table and their column definitions (data types and constraints). The name of the first column is *columnName1* and its definition is *columnDefinition1*, and so on.

Create the table in the sample database, Figure 3.2, except use the name, People, instead of Person as the name of the table and use the name, name, instead of fName, as the name of the first column. These names will be changed later by ALTER TABLE commands. The domains of the four columns have been explained in the previous section, Sample Database. The columns, name and gender, are declared as NOT NULL which means that every row of data in the table must have a value for name and gender. The constraint, NOT NULL, is also discussed in the section, Sample Database.

```
CREATE TABLE People (
   name      VARCHAR(10) NOT NULL,
   weight    DECIMAL(3),
   height    DECIMAL(3,2),
   gender    CHAR(1) NOT NULL);
```

Display the description of the columns of table, People, using the mysql command, DESC. .

```
   DESC People;
```

Figure 3.3 shows the DDL command, create table, and the description of the columns of the table.

3.6.2 Rename an existing table

Syntax:

ALTER TABLE *tableName*
    RENAME *newTableName*;

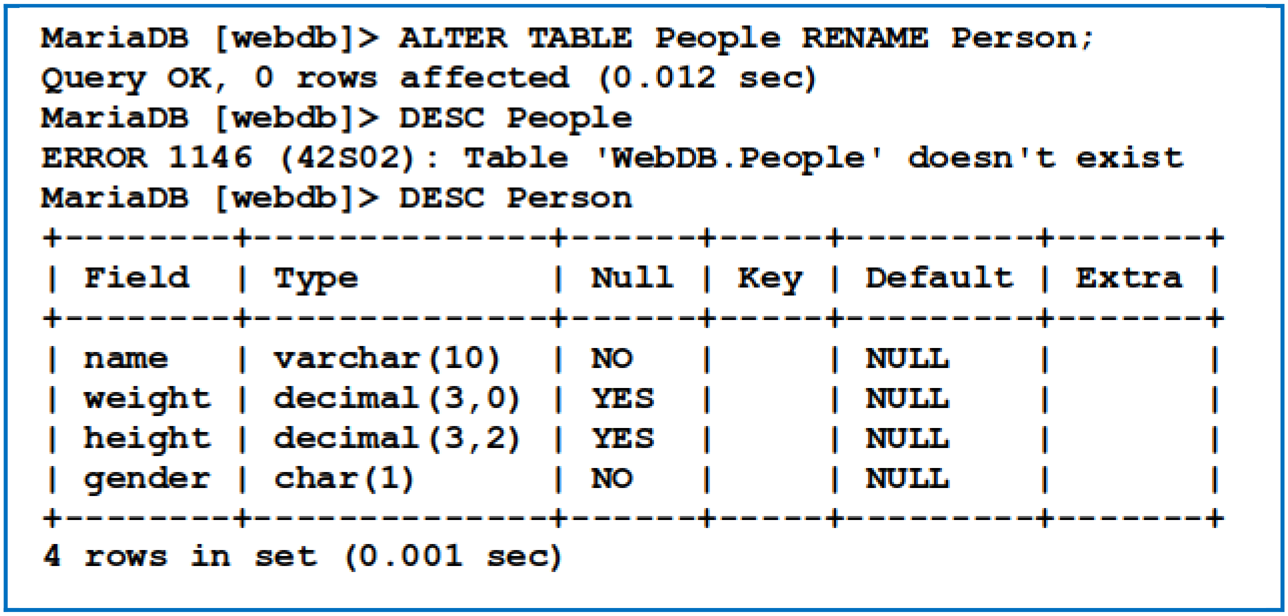
The ALTER TABLE . . . RENAME command is used to rename an existing table.

Change the name of table, People, to Person. After the table is renamed the original table, People, no longer exists. This will cause problems if the original table is already used by applications. Any application using the original table would need to be modified to refer to the new name. Until this is done, the application will fail.

```
   ALTER TABLE People
         RENAME Person;
```

Display the description of the columns of tables, People and Person.

```
   DESC People;
   DESC Person;
```



```
MariaDB [webdb]> ALTER TABLE People RENAME Person;
Query OK, 0 rows affected (0.012 sec)
MariaDB [webdb]> DESC People
ERROR 1146 (42S02): Table 'WebDB.People' doesn't exist
MariaDB [webdb]> DESC Person
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| name   | varchar(10)  | NO   |     | NULL    |       |
| weight | decimal(3,0) | YES  |     | NULL    |       |
| height | decimal(3,2) | YES  |     | NULL    |       |
| gender | char(1)      | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.001 sec)
```

Figure 3.4 DDL command: ALTER TABLE . . . RENAME

Figure 3.4 shows the DDL command, ALTER TABLE . . . RENAME, and the description of the columns of the tables, People and Person. The table, People, no longer exists.

3.6.3 Add a column to an existing table

Syntax:

ALTER TABLE *tableName*
ADD *columnName columnDefinition*;

The clause, *columnDefinition*, contains the data type of the column and, optionally, the constraints of the column, for example, NOT NULL and PRIMARY KEY.

The ALTER TABLE . . . ADD command is used to add an additional column to an existing table.

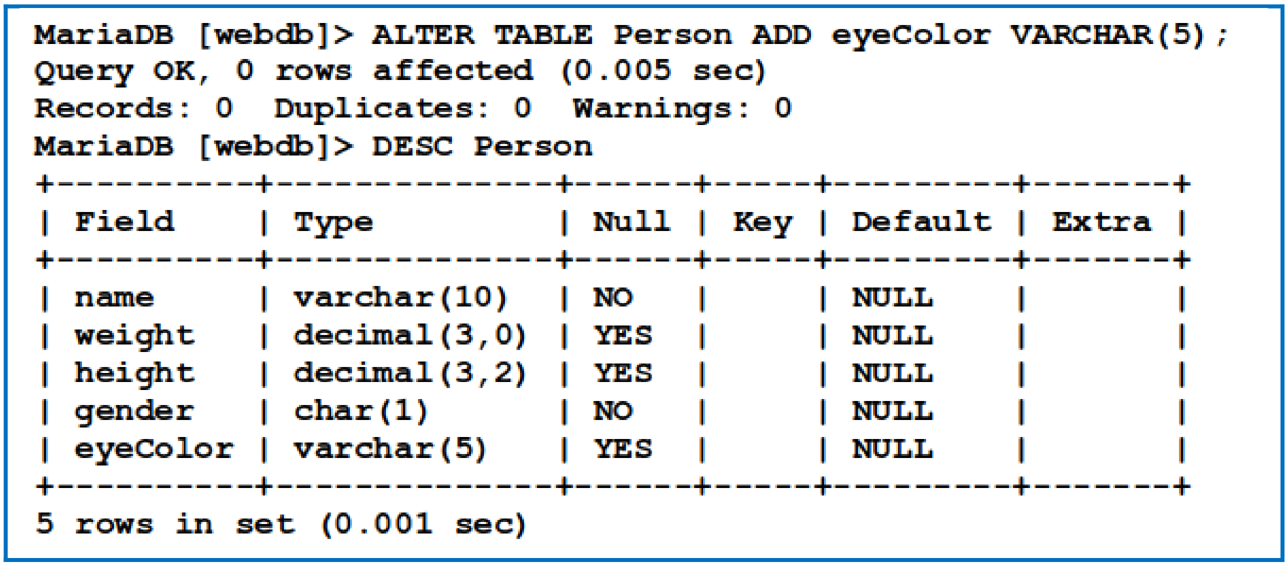
Add the column, eyeColor, with domain, VARCHAR(5), to table, Person. So the new column contains a string with a maximum of five characters.

```
ALTER TABLE Person
   ADD eyeColor VARCHAR(5);
```

Display the description of the columns of tables, People and Person.

```
DESC Person;
```

Figure 3.5 shows the DDL command, ALTER TABLE . . . ADD, and the description of the columns of the table, Person. Now the table has five columns.



```
MariaDB [webdb]> ALTER TABLE Person ADD eyeColor VARCHAR(5);
Query OK, 0 rows affected (0.005 sec)
Records: 0  Duplicates: 0  Warnings: 0
MariaDB [webdb]> DESC Person
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| name     | varchar(10)  | NO   |     | NULL    |       |
| weight   | decimal(3,0) | YES  |     | NULL    |       |
| height   | decimal(3,2) | YES  |     | NULL    |       |
| gender   | char(1)      | NO   |     | NULL    |       |
| eyeColor | varchar(5)   | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
5 rows in set (0.001 sec)
```

Figure 3.5 DDL command: ALTER TABLE . . . ADD

3.6.4 Remove a column from an existing table

Syntax:

ALTER TABLE *tableName*
DROP *columnName*;

The ALTER TABLE . . . DROP command is used to remove an existing column from an existing table.

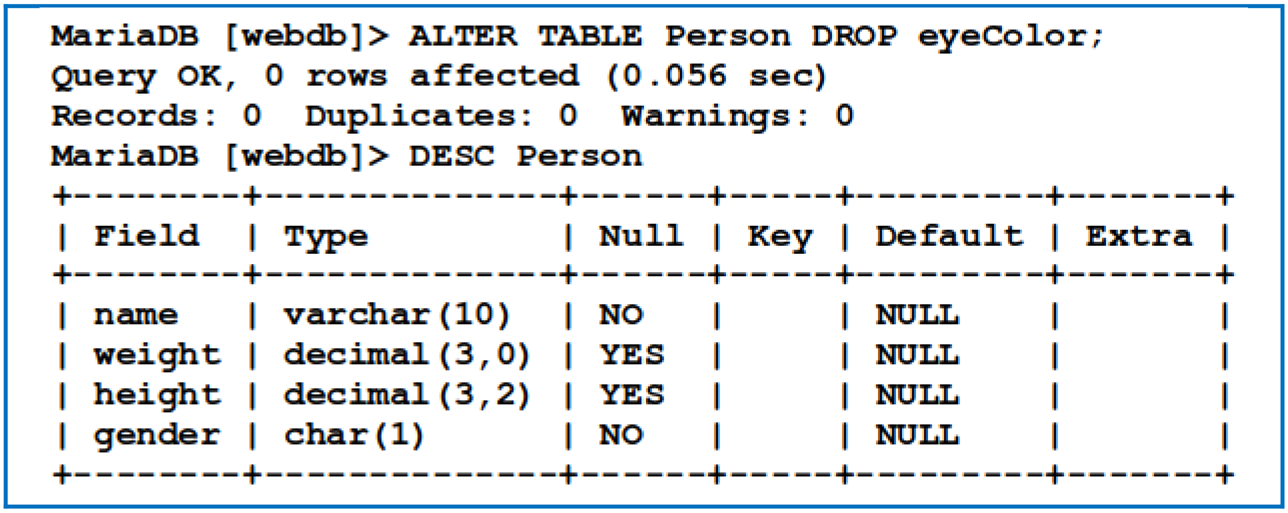
Suppose that the eye color of a person is not needed for the purpose of the database. Remove the column, eyeColor, from the table, Person.

```
ALTER TABLE Person
   DROP eyeColor;
```

Display the description of the columns of table, Person.

```
DESC Person;
```

Figure 3.6 shows the DDL command, ALTER TABLE . . . DROP, and the description of the columns of the table, Person. Now the table has four columns again.



```
MariaDB [webdb]> ALTER TABLE Person DROP eyeColor;
Query OK, 0 rows affected (0.056 sec)
Records: 0  Duplicates: 0  Warnings: 0
MariaDB [webdb]> DESC Person
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| name   | varchar(10)  | NO   |     | NULL    |       |
| weight | decimal(3,0) | YES  |     | NULL    |       |
| height | decimal(3,2) | YES  |     | NULL    |       |
| gender | char(1)      | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

Figure 3.6 DDL command: ALTER TABLE . . . DROP

3.6.5 Rename an existing column in an existing table

Syntax:

ALTER TABLE *tableName*
  CHANGE *oldColumnName newColumnName columnDefinition*;

The clause, *columnDefinition*, contains the data type of the column and, optionally, the constraints of the column, for example, NOT NULL and PRIMARY KEY.

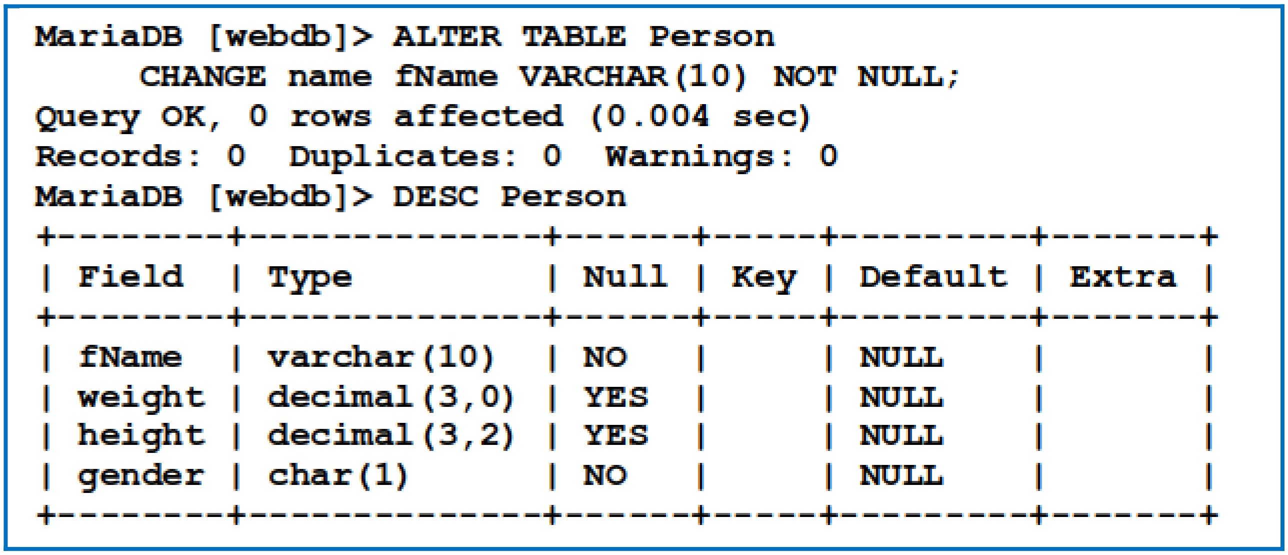
The ALTER TABLE . . . CHANGE command is used to change the name, data type and constraints of an existing column in an existing table. If only the name is changed, the existing data type and constraints must be given in *columnDefinition*. If only the data type or constraint are changed, the *newColumnName* is the same as the *oldColumnName*.

The name of the column, name, is not explicit enough. Change the name of column, name, to fName, an abbreviation for first name.

```
ALTER TABLE Person
   CHANGE name fName VARCHAR(10) NOT NULL;
```

Display the description of the columns of table, Person: `DESC Person;`

Figure 3.7 shows the DDL command, ALTER TABLE . . . CHANGE, and the description of the columns of the table, Person. Now the name of the first column is fName.



```
MariaDB [webdb]> ALTER TABLE Person
     CHANGE name fName VARCHAR(10) NOT NULL;
Query OK, 0 rows affected (0.004 sec)
Records: 0  Duplicates: 0  Warnings: 0
MariaDB [webdb]> DESC Person
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| fName  | varchar(10)  | NO   |     | NULL    |       |
| weight | decimal(3,0) | YES  |     | NULL    |       |
| height | decimal(3,2) | YES  |     | NULL    |       |
| gender | char(1)      | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
```

Figure 3.7 DDL command: ALTER TABLE . . . CHANGE

3.6.6 Change the data type of an existing column in an existing table

Syntax:

ALTER TABLE *tableName*
MODIFY *columnName columnDefinition*;

The clause, *columnDefinition*, contains the data type of the column and, optionally, the constraints of the column, for example, NOT NULL and PRIMARY KEY.

The ALTER TABLE . . . MODIFY command is used to change the data type of an existing column in an existing table without changing the name of the column, as in the previous command (ALTER TABLE . . . CHANGE).

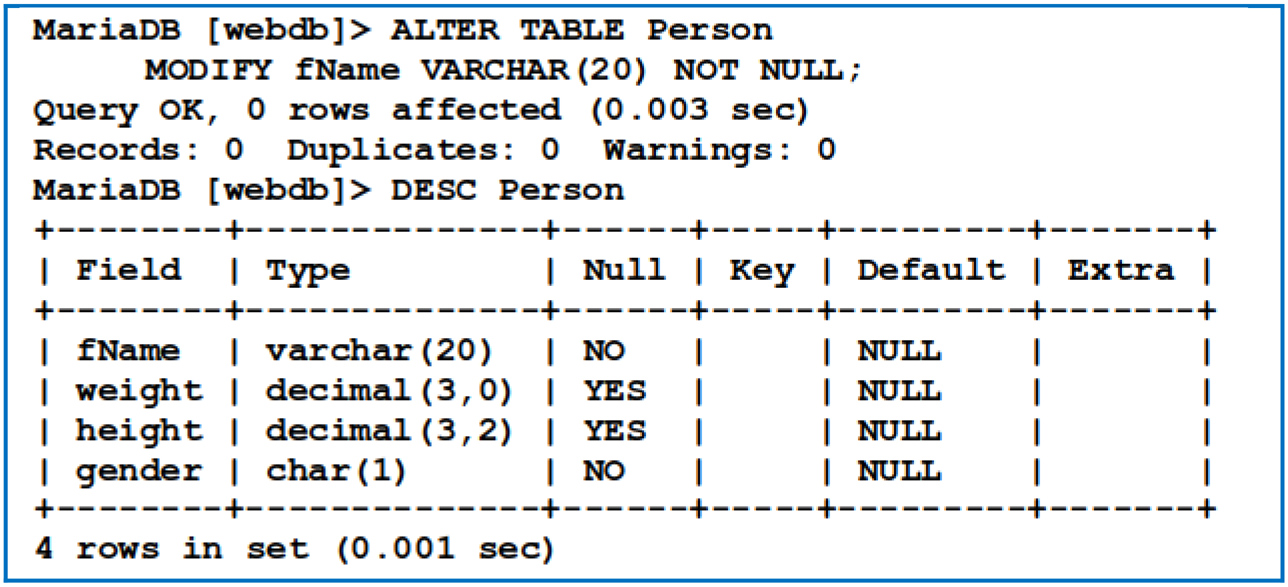
The first name of a person could be longer than 10 characters. Change the size of the domain for column, fName, from 10 to 20 characters. In general, care must be taken when this modification is made. The new domain should include all of the data of the old domain of the column. In this case, the new domain will always include all of the existing data, since the new size is larger than the old size.

```
ALTER TABLE Person
  MODIFY fName VARCHAR(20) NOT NULL;
```

Display the description of the columns of table, Person.

```
DESC Person;
```

Figure 3.8 shows the DDL command, ALTER TABLE . . . MODIFY, and the description of the columns of the table, Person. Now the size of the domain of column, fName, is 20.



```
MariaDB [webdb]> ALTER TABLE Person
     MODIFY fName VARCHAR(20) NOT NULL;
Query OK, 0 rows affected (0.003 sec)
Records: 0  Duplicates: 0  Warnings: 0
MariaDB [webdb]> DESC Person
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| fName  | varchar(20)  | NO   |     | NULL    |       |
| weight | decimal(3,0) | YES  |     | NULL    |       |
| height | decimal(3,2) | YES  |     | NULL    |       |
| gender | char(1)      | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.001 sec)
```

Figure 3.8 DDL command: ALTER TABLE . . . MODIFY

3.6.7 Create a new table from an existing table

Syntax:

CREATE TABLE *newTable* [*newColumnList*]
SELECT *existingColumnList*
FROM *existingTable*;

The CREATE TABLE . . . SELECT . . . FROM command is used to create a new table by using some or all of the columns of an existing table. All columns from *existingTable* can be included using * (meaning all columns) in place of *existingColumnList*. The names of the columns in *newTable* can be changed by listing the new names in *newColumnList*, but this is optional as indicated by the square brackets in the syntax. Both the schema (structure) and contents of the columns of the existing table are copied to the new table.

This command can be used to backup a table and so corresponds to the copy command, cp *existingFile newFile*, in a Linux shell when *existingColumnList* is all columns, *. In practice a database consists of many tables. A SQL file (script) would be written to backup all tables by first deleting all of the tables which were backed up last time, with the DROP TABLE command, and then copying all of the tables with the CREATE TABLE . . . SELECT . . . FROM command. If the tables in the database have names, Table1, Table2, and so on, the backed up tables could be given the names, BTable1, BTable2, and so on (B for backup). Remember that every table in a database must have a name different from all other tables.

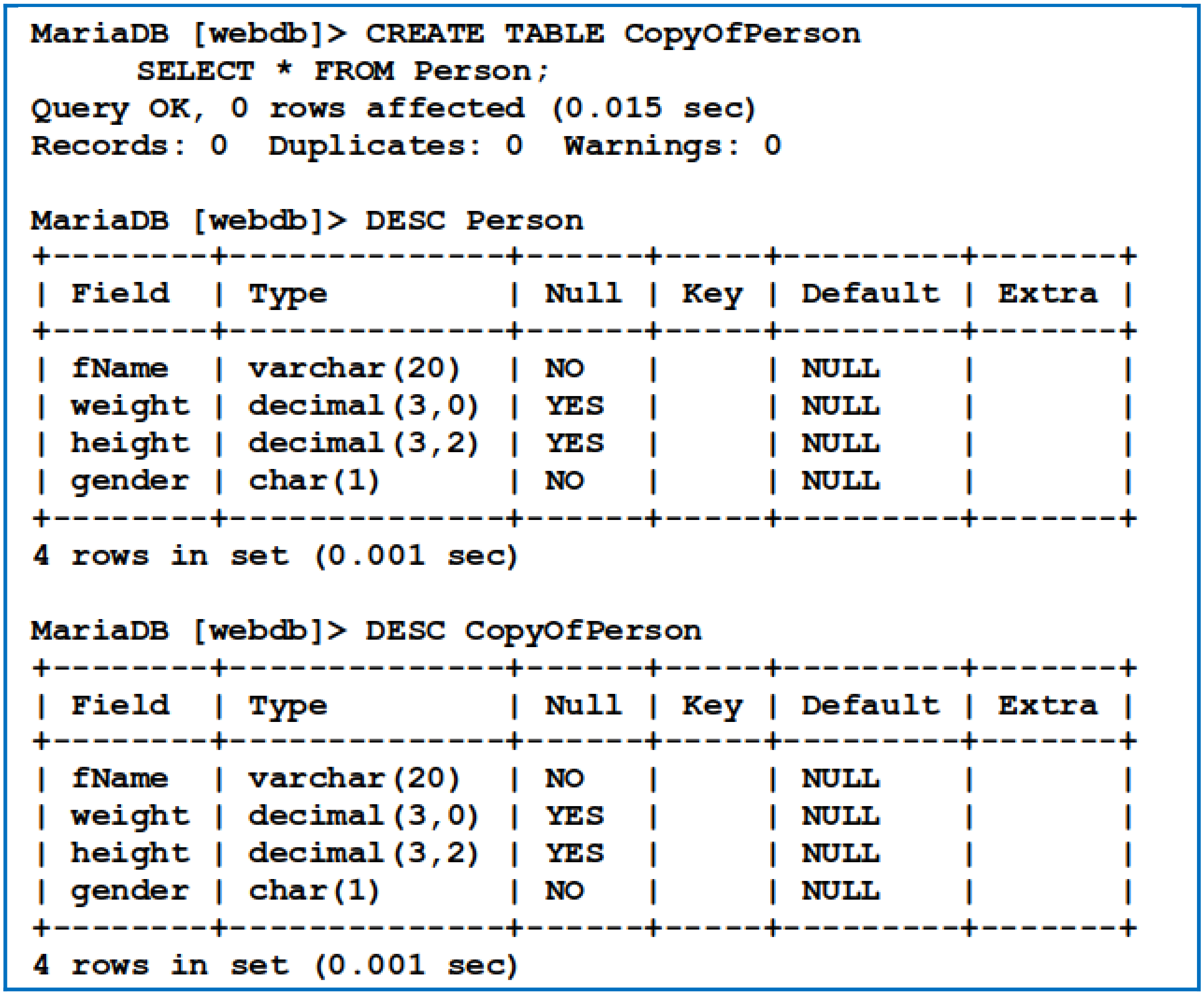
Use the name, CopyOfPerson, as the name of the new table. Determine if a table with this name exists by the command that displays the columns of the table.

```
DESC CopyOfPerson;
```

It is assumed that this table does not exist.

Copy the entire table, Person, to the new table, CopyOfPerson.

```
CREATE TABLE CopyOfPerson
   SELECT *
   FROM Person;
```



```
MariaDB [webdb]> CREATE TABLE CopyOfPerson
     SELECT * FROM Person;
Query OK, 0 rows affected (0.015 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [webdb]> DESC Person
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| fName  | varchar(20)  | NO   |     | NULL    |       |
| weight | decimal(3,0) | YES  |     | NULL    |       |
| height | decimal(3,2) | YES  |     | NULL    |       |
| gender | char(1)      | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.001 sec)

MariaDB [webdb]> DESC CopyOfPerson
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| fName  | varchar(20)  | NO   |     | NULL    |       |
| weight | decimal(3,0) | YES  |     | NULL    |       |
| height | decimal(3,2) | YES  |     | NULL    |       |
| gender | char(1)      | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.001 sec)
```

Figure 3.9 DDL command: CREATE TABLE . . . SELECT . . . FROM

Display the description of the columns of both the original table and the new table.

```
DESC Person;
DESC CopyOfPerson;
```

Figure 3.9 shows the DDL command, CREATE TABLE . . . SELECT . . . FROM, and the description of the columns of the original and new tables. The columns of the new table are the same as those of the original table.

3.6.8 Delete an existing table

Syntax:

DROP TABLE *tableName*;

The DROP TABLE command is used to delete an existing table. When a table is dropped, both the structure and the contents of the table are deleted. This command should be used with care. Any existing application which requires the dropped table would fail unless the application is modified. Also, if a table is deleted by mistake, it may or may not be possible to undo the change made by the command. The DCL commands, COMMIT and ROLLBACK, are concerned with making changes permanent and reversing changes.

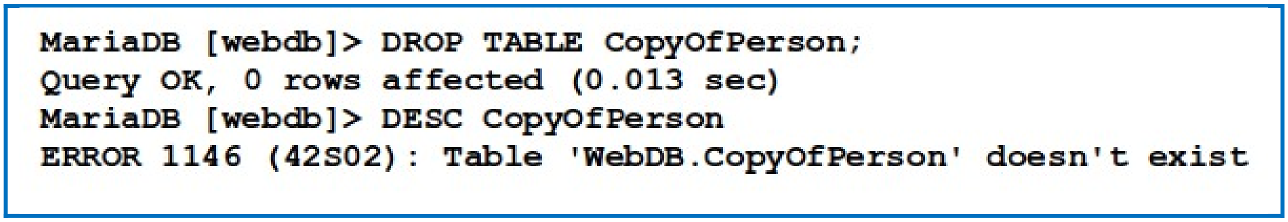
Delete the table, CopyOfPerson.

```
   DROP TABLE CopyOfPerson;
```

Display the description of the columns of the dropped table, CopyOfPerson.

```
   DESC CopyOfPerson;
```

Figure 3.10 shows the DDL command, DROP TABLE, and the command to show the description of the columns of the table, CopyOfPerson, after it is dropped. The table no longer exists.



```
MariaDB [webdb]> DROP TABLE CopyOfPerson;
Query OK, 0 rows affected (0.013 sec)
MariaDB [webdb]> DESC CopyOfPerson
ERROR 1146 (42S02): Table 'WebDB.CopyOfPerson' doesn't exist
```

Figure 3.10 DDL command: DROP TABLE

3.7 DML (Data Manipulation Language) Demonstrations

The Data Manipulation Language of SQL allows a user to populate tables and interrogate tables of a database. When a table is created by DDL, only its schema is defined, that is, only its structure exists. The table is empty, meaning that the number of rows in the table is zero. DML is used put data in rows of the table (populate the table), as well as to examine the data in the table (interrogate the table).

There are two types of DML commands: populate tables and interrogate (query) tables.
(a) Populate commands change the contents of the tables. The commands are INSERT, UPDATE and DELETE. The INSERT command adds new data to the database. The UPDATE command modifies existing data in the database. The DELETE command removes existing data from the database.
(b) There is one interrogate (query) command, SELECT, which has several variations. The SELECT command does not change the contents of the tables. The SELECT command retrieves data from the database.

It is not convenient to demonstrate the commands in the order of their classification above. Rather the INSERT command is covered first, followed by the SELECT command and then the UPDATE and DELETE commands.

3.7.1 Add data to a table

Syntax:

INSERT INTO *tableName* [*columnList*]
  VALUES (*valueList*1),
      (*valueList*2),
      . . .
      (*valueList*N);

The INSERT INTO . . . VALUES command is used to place new data in one or more rows of an existing table, *tableName*. The *columnList*, a list of names of columns from the table, can be given, but this is optional as indicated by the square brackets in the syntax. If *columnList* is given, *valueList*, a list of values of the columns in one row of the table, must be in the same order as *columnList*. If *columnList* is not given, *valueList* must be in the same order as the columns in the CREATE TABLE command that was used to create the table, as displayed by the DESC command. Obviously, the table must be created before data can be inserted into it, that is, you need the container before you can put something in it.

The table, Person, is initially empty. Confirm this by using the SELECT . . . FROM command, which is covered in the next subsection. The output of the command is, Empty set, meaning that the table exists and that it is empty.

```
SELECT * FROM Person;
```

Insert four rows into the table, Person, with the data in Figure 3.2. One INSERT INTO . . . VALUES command could be used to insert all four rows of data. But three commands instead of one are used in order to demonstrate the use of the *columnList*.
The *columnList* is given for the first command, where the *valueList* is in the same order as the *columnList*.

```
INSERT INTO Person (fName, weight, height, gender)
  VALUES ('Tom', 160, 1.78, 'm');
```

Since values of all of the columns are inserted and they are inserted in the same order as the columns were created by the CREATE TABLE command, it is unnecessary to give the *columnList*. The second command gives only the *valueList* for two rows with all of the columns in the same order as the columns were created by the CREATE TABLE command. When data is missing in a row, the keyword, null, must be typed in the *valueList*, which applies to the row of data with fName, Harry.

```
INSERT INTO Person VALUES
  ('Dick', 145, 1.73, 'm'),
  ('Harry', null, 1.93, 'm');
```

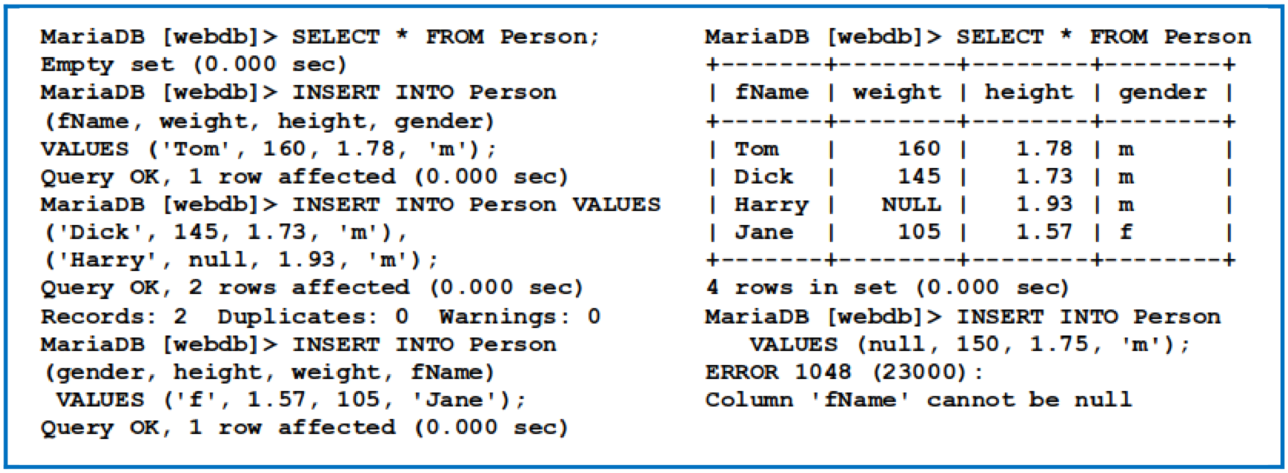
Values of all columns are inserted in the third command but the order of the *valueList* is changed and so the *columnList* must be given to correspond to the order of the *valueList*.

```
INSERT INTO Person (gender, height, weight, fName)
  VALUES ('f', 1.57, 105, 'Jane');
```

Display all data in table, Person, using the SELECT . . . FROM command, which is covered in the next subsection: `SELECT * FROM Person;`

Attempt to insert a row of data in table, Person, without a value for column, fName. This column was declared NOT NULL when the table was created and so the command is rejected.

```
   INSERT INTO Person
      VALUES (null, 150, 1.75, 'm');      (error)
```



```
MariaDB [webdb]> SELECT * FROM Person;
Empty set (0.000 sec)
MariaDB [webdb]> INSERT INTO Person
(fName, weight, height, gender)
VALUES ('Tom', 160, 1.78, 'm');
Query OK, 1 row affected (0.000 sec)
MariaDB [webdb]> INSERT INTO Person VALUES
('Dick', 145, 1.73, 'm'),
('Harry', null, 1.93, 'm');
Query OK, 2 rows affected (0.000 sec)
Records: 2  Duplicates: 0  Warnings: 0
MariaDB [webdb]> INSERT INTO Person
(gender, height, weight, fName)
 VALUES ('f', 1.57, 105, 'Jane');
Query OK, 1 row affected (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM Person
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
+-------+--------+--------+--------+
4 rows in set (0.000 sec)
MariaDB [webdb]> INSERT INTO Person
   VALUES (null, 150, 1.75, 'm');
ERROR 1048 (23000):
Column 'fName' cannot be null
```

Figure 3.11 DML command: INSERT INTO . . . VALUES

Figure 3.11 first shows the SELECT . . . FROM command which verifies that the table exists and is empty. Next the DML command, INSERT INTO . . . VALUES, is executed three times for valid data and the SELECT . . . FROM command is executed to show all the data in the table. Lastly, the command to insert invalid data, data for which fName is null, is shown with the error message.

3.7.2 Retrieve data from a table

Brief syntax:

SELECT [DISTINCT] *select_list*
FROM *table_list*
[WHERE *where_conditions*]
[GROUP BY *group_by_list*]
[HAVING *grouping_conditions*]
[ORDER BY *order_list* [ASC | DESC]]

The square brackets indicate parts of the command which are optional. The parts on each line of the syntax of the SELECT command are often called clauses.

- The *select_list* in the SELECT clause includes attributes (columns) and the aggregate functions: AVG, COUNT, MAX, MIN, SUM. DISTINCT returns only one copy of any duplicate rows. The SELECT clause effectively acts as the projection operation in relational algebra.
- The *table_list* in the FROM clause is used to specify the source tables for the SELECT operation.

- The *where_conditions* in the WHERE clause are used to give the constraints that are to be satisfied for the rows of the tables. The WHERE clause effectively acts as the selection operation in relational algebra.
- The GROUP BY clause specifies the subgroups of rows of source data, *group_by_list*.
- The HAVING clause gives the constraints, *grouping_conditions*, on the GROUP BY operation.
- The ORDER BY clause gives the required ordering of the result based on the values of the columns in the *order_list* and whether the ordering should be ascending (ASC) or descending (DESC). Ascending means in alphabetical order or numerical order and is the default. Descending means in reverse alphabetical order or reverse numerical order.

The entire or partial contents of one or more tables can be retriePRIMARY KEY (dCode));ved using the SELECT command.

The SELECT command is demonstrated here using only one table, Person, which has been created and populated earlier. The required clauses, SELECT and FROM, are present in every command. The optional clauses are demonstrated individually with the required clauses except for HAVING. The HAVING clause cannot be used without GROUP BY.

3.7.2.1 SELECT with FROM

Syntax:

SELECT *columnName1*, *columnName2*, . . .
FROM *tableName*;

All rows are selected by SELECT . . . FROM and some of the columns given by the column list after SELECT. All columns can be selected by using the shorthand, *, in place of the column list after SELECT. The symbol, *, designates all the columns of the table specified in the FROM clause.

The SELECT . . . FROM command is the simplest form of the command because it contains only the required parts, that is, there are no optional parts (given in square brackets in the syntax above). This operation is called a projection in relational algebra. This operation filters the columns, that is, removes some of the columns from a table.

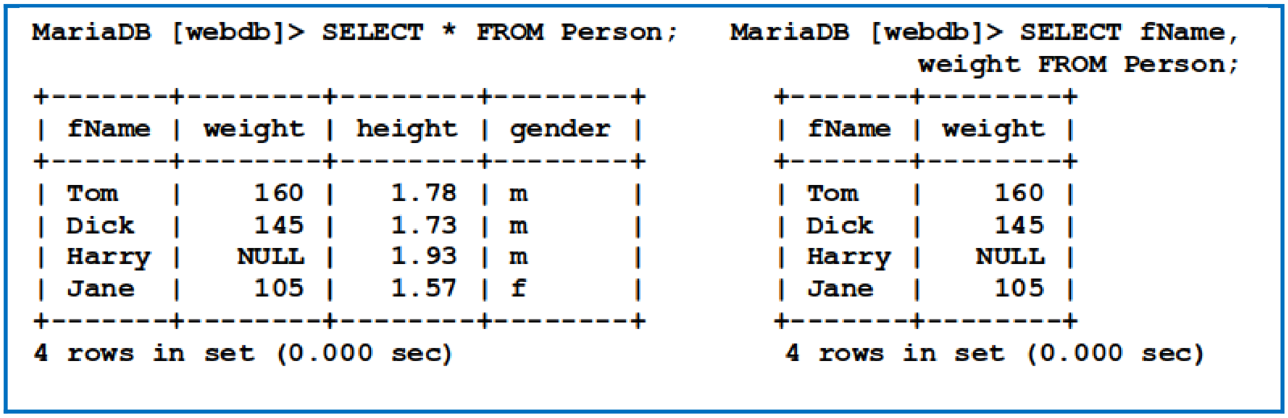
Display all columns of table, Person.

```
SELECT *
     FROM Person;
```

Display two of the four columns of table, Person.

```
SELECT fName, weight
     FROM Person;
```

Figure 3.12 shows the two SELECT . . . FROM commands.



```
MariaDB [webdb]> SELECT * FROM Person;   MariaDB [webdb]> SELECT fName,
                                                     weight FROM Person;
+-------+--------+--------+--------+         +-------+--------+
| fName | weight | height | gender |         | fName | weight |
+-------+--------+--------+--------+         +-------+--------+
| Tom   |    160 |   1.78 | m      |         | Tom   |    160 |
| Dick  |    145 |   1.73 | m      |         | Dick  |    145 |
| Harry |   NULL |   1.93 | m      |         | Harry |   NULL |
| Jane  |    105 |   1.57 | f      |         | Jane  |    105 |
+-------+--------+--------+--------+         +-------+--------+
4 rows in set (0.000 sec)                     4 rows in set (0.000 sec)
```

Figure 3.12 DML command: SELECT . . . FROM

3.7.2.2 SELECT with WHERE

Syntax:

SELECT *columnName1*, *columnName2*, . . .
FROM *tableName*
WHERE *where_conditions*;

The keyword, WHERE, specifies the conditions to be satisfied by rows of a table which are retrieved. To retrieve all rows, omit WHERE. The command with WHERE corresponds to the operation, selection, in relational algebra. This operation filters the rows, that is, removes some of the rows from a table.

Note that, unfortunately, the terminology used for SQL does not correspond to the terminology for the corresponding relational algebra operations. The rough equivalents of the algebraic operations with the SQL keywords are: algebraic projection is equivalent to SELECT, and algebraic selection corresponds to WHERE.

There are various operators used to express *where_conditions* in the WHERE clause. Two of the most important types of operators are the relational operators and logical operators. The relational operators are classified as comparison operators in MySQL documentation.

All of the operators are given in the MySQL documentation with the internet address in section, References, at the end of this chapter. The location in the documentation is Chapter 11 Functions and Operators.

The operators used most often are the following.

3.7.2.2.1 Relational operators

Syntax: WHERE *expression1 relational_operator expression2*

The relational operators are comparison operators which compare one expression with another expression and return either the value, TRUE or FALSE.

There are six relational operators. They are: =, <, >, !=, >=, <=. The operator, <>, as well as, !=, is permitted for not equals. Note that in the C language (and C++ and Java), two equal signs, ==, are used for the equals operator instead of one equal sign, =, in SQL. In C, the single equal sign is used for the assignment operator. The use of two equal signs for the relational operator, equals, in SQL is an error.

3.7.2.2.2 Logical operators

Syntax: WHERE *condition1 logical_operator condition2*

There are three logical operators. They are: AND, OR, NOT.

Logical operators combine the results of two conditions and return either the value, TRUE or FALSE.

3.7.2.2.3 LIKE, NOT LIKE operators

Syntax: WHERE *columnNameValue* LIKE | NOT LIKE *pattern*;

The comparison operators, LIKE and NOT LIKE, are used to match a string to a pattern. In a Linux shell, LIKE corresponds to the operator, =~ (match), and NOT LIKE corresponds to ! ~ (not match). The pattern is also a string which can contain wildcards. There are two wildcards in SQL used with LIKE: % and _. The wildcard, %, in SQL matches 0 or more characters in a string. The wildcard, _, in SQL matches one character in a string. The corresponding wildcards in a Linux shell and DOS shell are * for % and ? for _.

3.7.2.2.4 IS NULL, IS NOT NULL operators

Syntax: WHERE *columnNameValue* IS NULL | IS NOT NULL;

The comparison operators, IS NULL and IS NOT NULL, are the only operators which test if the value of an column is missing (null) or not missing (not null).

Display all rows.

```
SELECT *
     FROM Person;
```

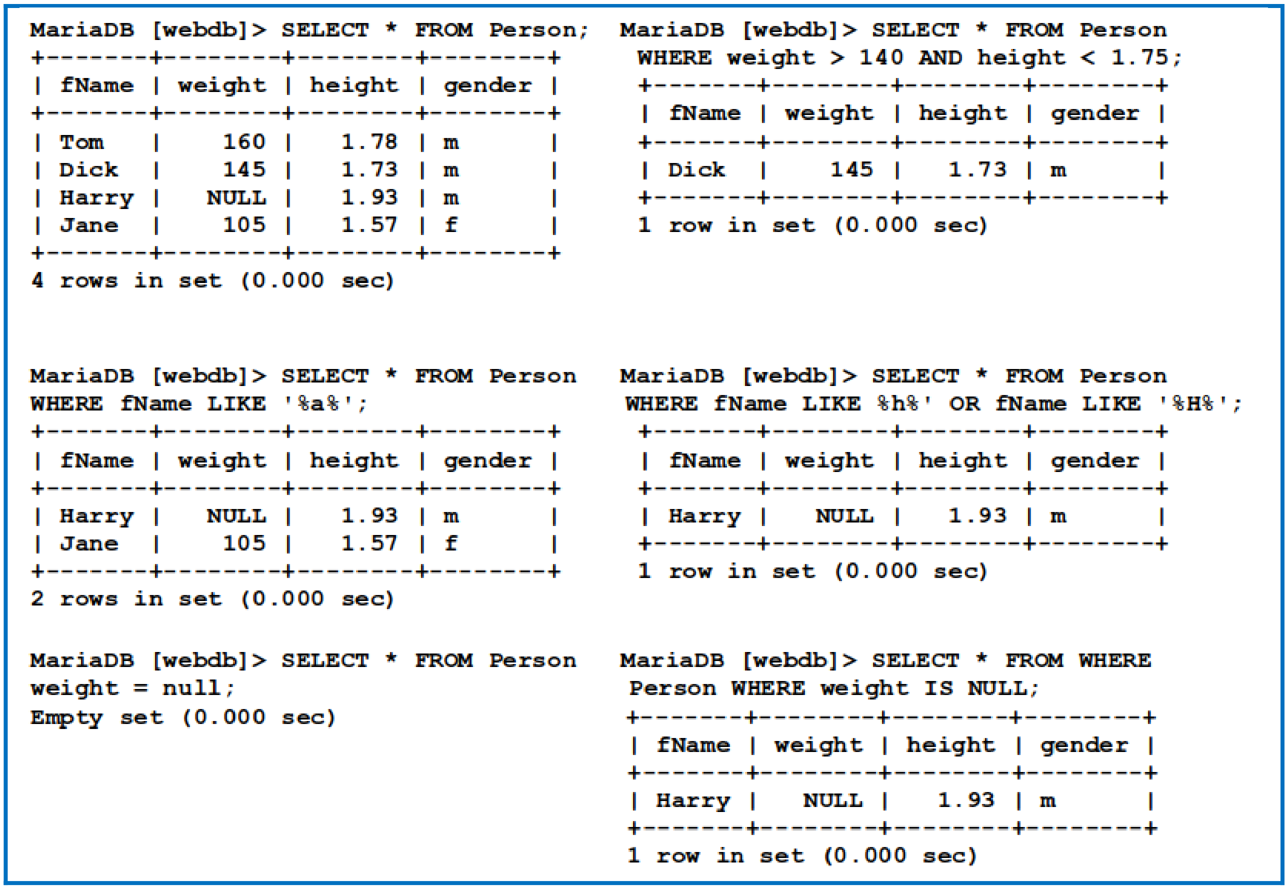
Display all rows for which weight is greater than 140 and height is less than 1.75.

```
SELECT *
     FROM Person
     WHERE weight > 140 AND height < 1.75;
```

Display all rows for which fName contains the character, a, in any position.

```
SELECT *
     FROM Person
     WHERE fName LIKE '%a%';
```

The characters in the pattern are case sensitive.Display all rows for which fName contains the character, h, regardless of case in any position. The only character, h, is the first letter of the name, Harry. Two patterns are needed to search for both h and H: '%h%' or '%H%'.



```
MariaDB [webdb]> SELECT * FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
+-------+--------+--------+--------+
4 rows in set (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM Person
 WHERE weight > 140 AND height < 1.75;
 +-------+--------+--------+--------+
 | fName | weight | height | gender |
 +-------+--------+--------+--------+
 | Dick  |    145 |   1.73 | m      |
 +-------+--------+--------+--------+
 1 row in set (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM Person
WHERE fName LIKE '%a%';
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
+-------+--------+--------+--------+
2 rows in set (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM Person
WHERE fName LIKE %h%' OR fName LIKE '%H%';
 +-------+--------+--------+--------+
 | fName | weight | height | gender |
 +-------+--------+--------+--------+
 | Harry |   NULL |   1.93 | m      |
 +-------+--------+--------+--------+
 1 row in set (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM Person
weight = null;
Empty set (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM WHERE
 Person WHERE weight IS NULL;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Harry |   NULL |   1.93 | m      |
+-------+--------+--------+--------+
1 row in set (0.000 sec)
```

Figure 3.13 DML command: SELECT . . . FROM . . . WHERE

```
SELECT *
     FROM Person
     WHERE fName LIKE '%h%' OR fName LIKE '%H%';
```

Display all rows for which the weight is null. The first SELECT command is incorrect because null is not a value; it is the absence of a value.

```
SELECT *
     FROM Person
     WHERE weight = null;     (error)
SELECT *
     FROM Person
     WHERE weight IS NULL;
```

Figure 3.13 shows the SELECT . . . FROM . . . WHERE commands and the output of the commands.

3.7.2.3 SELECT with ORDER BY

Syntax:

SELECT *select_list*
FROM *table_list*

ORDER BY *order_list* [ASC | DESC]

The clause, ORDER BY, in the SELECT command sorts rows according to *order_list*, which contains one or more columns. The default ordering is in the ascending order (ASC), meaning in alphabetical or numerical order (smallest number first). If the required order is descending (DESC), then DESC is placed after the column.

Most applications sort characters according to the value of the ASCII code of the characters. The ASCII code of uppercase letters is smaller than for lowercase letters, for example A is 65 , B is 66, a is 97, b is 98, and so on, where the numbers are in decimal. So if strings are displayed in order of ASCII code, strings starting with uppercase letters appear before strings starting with lowercase letters.

Add the name, frank, and gender, m, to the table, Person, without values for the other columns.

```
INSERT INTO Person
     VALUES ('frank', null, null, 'm');
```

Display all rows in alphabetical order (ascending order) of the first name, fName. Note that frank comes before Harry and after Dick. So MySQL sorts strings without regard to the case of the letters and so not in order of ASCII code.

```
SELECT *
     FROM Person
     ORDER BY fName;
```

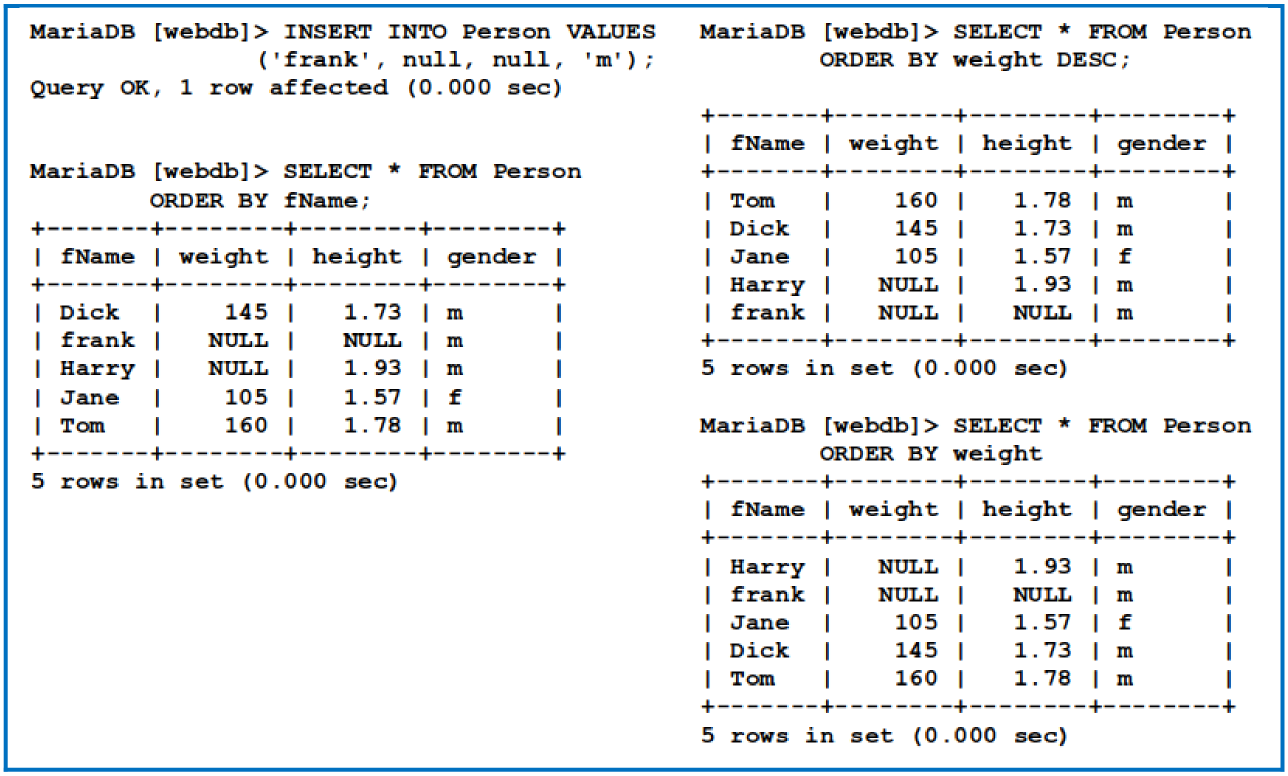
Display all rows in reverse numerical order of the weight (largest weight first). The rows for which weight is null (missing value) are displayed last.

```
SELECT *
     FROM Person
     ORDER BY weight DESC;
```

Display all rows in numerical order of the weight, weight (smallest weight first). The rows for which weight is null (missing value) are displayed first. So null is displayed as if the smallest number is associated with null. However, there is no numerical value for null.

```
SELECT *
     FROM Person
     ORDER BY weight;
```

Figure 3.14 shows the INSERT INTO . . . VALUES command and the three SELECT . . . FROM . . . ORDER BY commands and their output.



```
MariaDB [webdb]> INSERT INTO Person VALUES
                 ('frank', null, null, 'm');
Query OK, 1 row affected (0.000 sec)


MariaDB [webdb]> SELECT * FROM Person
        ORDER BY fName;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Dick  |    145 |   1.73 | m      |
| frank |   NULL |   NULL | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| Tom   |    160 |   1.78 | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM Person
        ORDER BY weight DESC;

+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Jane  |    105 |   1.57 | f      |
| Harry |   NULL |   1.93 | m      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)

MariaDB [webdb]> SELECT * FROM Person
        ORDER BY weight
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Harry |   NULL |   1.93 | m      |
| frank |   NULL |   NULL | m      |
| Jane  |    105 |   1.57 | f      |
| Dick  |    145 |   1.73 | m      |
| Tom   |    160 |   1.78 | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

Figure 3.14 DML command: SELECT . . . FROM . . . ORDER BY

3.7.2.4 SELECT with WHERE and ORDER BY

Syntax:

SELECT *select_list*
FROM *table_list*
WHERE *where_conditions*
ORDER BY *order_list* [ASC | DESC]

When both the WHERE and ORDER BY clauses are used in the same SELECT command, the rows are selected according to conditions specified by the WHERE clause and then arranged in the order specified by the ORDER BY clause.

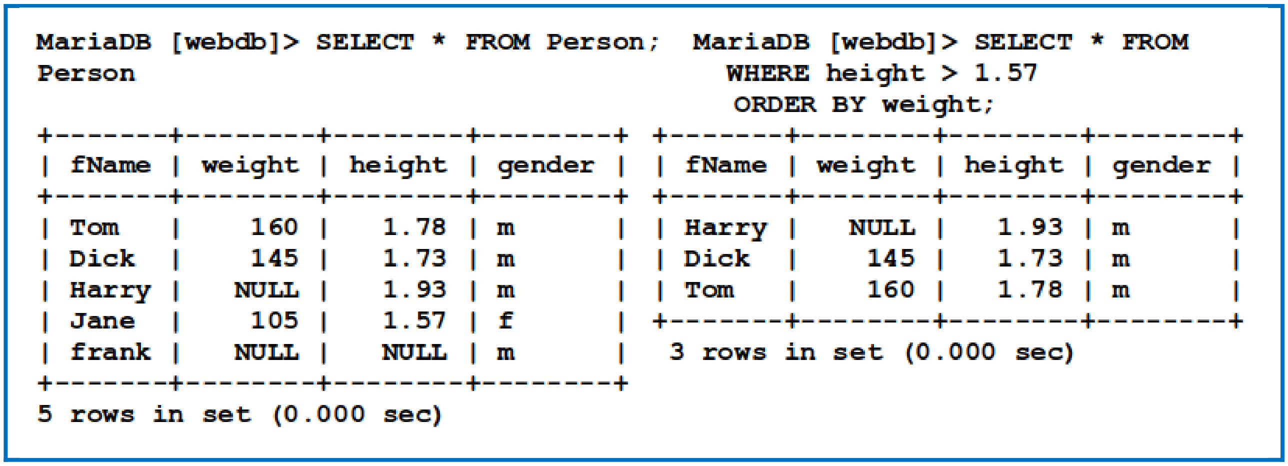
Display all rows of table, Person.

```
SELECT *
        FROM Person;
```

Display rows with height greater than 1.57 in order of increasing weight.

```
SELECT *
        FROM Person
        WHERE height > 1.57
        ORDER BY weight;
```

Figure 3.15 shows the last two SELECT commands and their output.



```
MariaDB [webdb]> SELECT * FROM Person;  MariaDB [webdb]> SELECT * FROM
Person                                      WHERE height > 1.57
                                            ORDER BY weight;
+-------+--------+--------+--------+ +-------+--------+--------+--------+
| fName | weight | height | gender | | fName | weight | height | gender |
+-------+--------+--------+--------+ +-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      | | Harry |   NULL |   1.93 | m      |
| Dick  |    145 |   1.73 | m      | | Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      | | Tom   |    160 |   1.78 | m      |
| Jane  |    105 |   1.57 | f      | +-------+--------+--------+--------+
| frank |   NULL |   NULL | m      |  3 rows in set (0.000 sec)
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

Figure 3.15 DML command: SELECT . . . FROM . . . WHERE . . . ORDER BY

3.7.2.5 Aggregate Functions

There are five aggregate functions that can be used in the SELECT command: COUNT, SUM, AVG, MIN, MAX. The aggregate functions are covered before the GROUP BY clause because the GROUP BY clause cannot be used without an aggregate function in the SELECT command.

There are many other built-in SQL functions in addition to the aggregate functions. An example is the function, ROUND, which is demonstrated together with AVG later in this subsection. There are also user-defined functions which are written in SQL/PSM, covered in a later chapter, and which can be used anywhere that the built-in functions can be used. The built-in SQL functions are listed and described in the MySQL documentation with the internet address in section, References, at the end of this chapter.

3.7.2.5.1 Aggregate Function: COUNT

Syntax:

COUNT(*columnName*)
COUNT(*)

COUNT counts the number of rows in a table. The * is a shorthand for all columns. When * is used with COUNT, all rows are counted, including rows with null (missing values) and including duplicate rows. When *columnName* is used, only the rows are counted with values of *columnName* which are not null.

Display all rows in the table, Person.

```
SELECT *
        FROM Person;
```

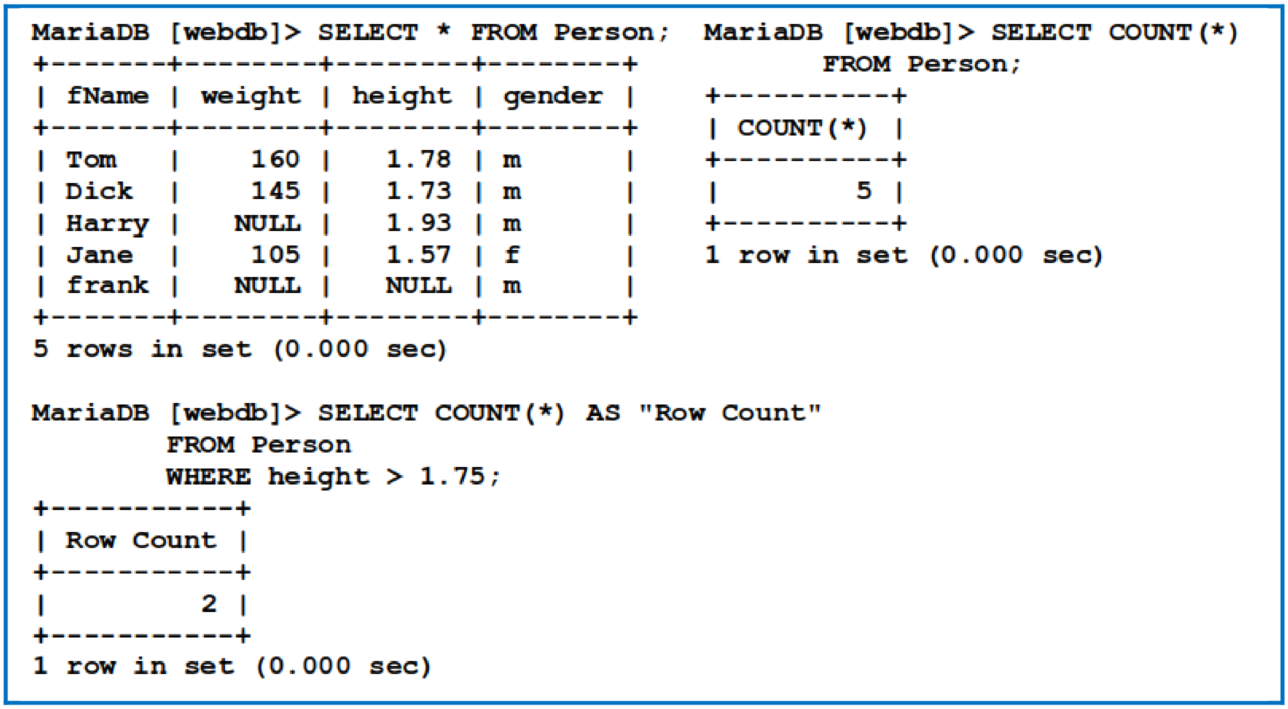
Display the number of all rows in the table, Person.

```
SELECT COUNT(*)
        FROM Person;
```

Display the number of rows in the table, Person, with height greater than 1.75. Also display the name of the column as Row Count.

```
      SELECT COUNT(*) AS "Row Count"
              FROM Person
              WHERE height > 1.75;
```

Figure 3.16 shows the last three commands, which demonstrate the use of COUNT.



```
MariaDB [webdb]> SELECT * FROM Person;  MariaDB [webdb]> SELECT COUNT(*)
+-------+--------+--------+--------+               FROM Person;
| fName | weight | height | gender |    +----------+
+-------+--------+--------+--------+    | COUNT(*) |
| Tom   |    160 |   1.78 | m      |    +----------+
| Dick  |    145 |   1.73 | m      |    |        5 |
| Harry |   NULL |   1.93 | m      |    +----------+
| Jane  |    105 |   1.57 | f      |    1 row in set (0.000 sec)
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)

MariaDB [webdb]> SELECT COUNT(*) AS "Row Count"
        FROM Person
        WHERE height > 1.75;
+-----------+
| Row Count |
+-----------+
|         2 |
+-----------+
1 row in set (0.000 sec)
```

Figure 3.16 Aggregate function: COUNT

3.7.2.5.2 Aggregate Function: SUM

Syntax:

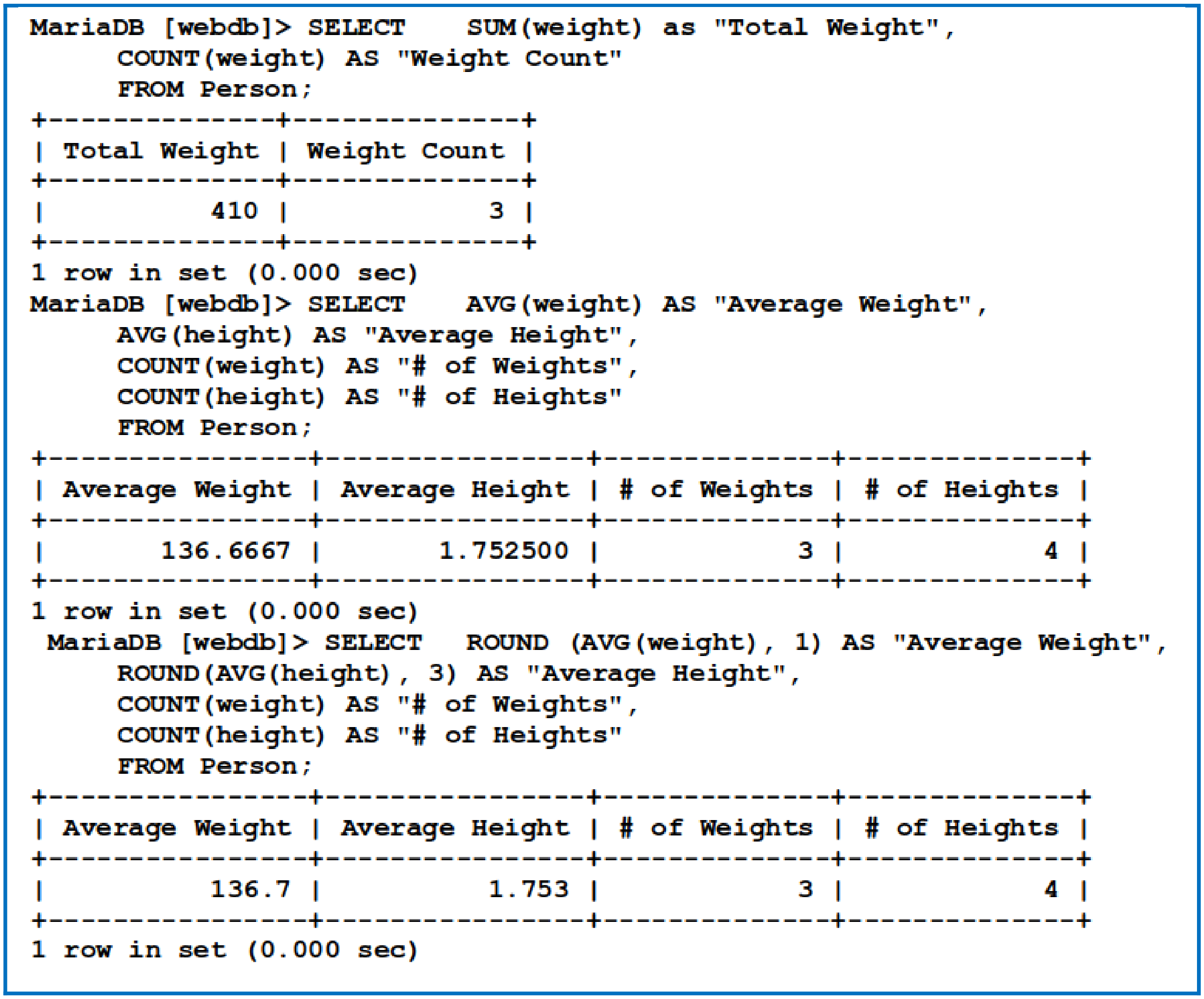
SUM(*columnName*)

SUM adds the values of all columns in the column with name, *columnName*. The column with name, *columnName*, has a numeric data type.

Display the total of all values of column, weight. Also display the number of rows which contribute to the sum. Display the names of the two columns as Total Weight and Weight Count.

```
      SELECT SUM(weight) as "Total Weight",
              COUNT(weight) AS "Weight Count"
              FROM Person;
```

Figure 3.17 shows the last command, which demonstrates the use of SUM. Verify the sum from the display of all rows of the table, Person, in Figure 3.16.



```
MariaDB [webdb]> SELECT   SUM(weight) as "Total Weight",
     COUNT(weight) AS "Weight Count"
     FROM Person;
+--------------+--------------+
| Total Weight | Weight Count |
+--------------+--------------+
|          410 |            3 |
+--------------+--------------+
1 row in set (0.000 sec)
MariaDB [webdb]> SELECT   AVG(weight) AS "Average Weight",
     AVG(height) AS "Average Height",
     COUNT(weight) AS "# of Weights",
     COUNT(height) AS "# of Heights"
     FROM Person;
+----------------+----------------+--------------+--------------+
| Average Weight | Average Height | # of Weights | # of Heights |
+----------------+----------------+--------------+--------------+
|       136.6667 |       1.752500 |            3 |            4 |
+----------------+----------------+--------------+--------------+
1 row in set (0.000 sec)
 MariaDB [webdb]> SELECT   ROUND (AVG(weight), 1) AS "Average Weight",
     ROUND(AVG(height), 3) AS "Average Height",
     COUNT(weight) AS "# of Weights",
     COUNT(height) AS "# of Heights"
     FROM Person;
+----------------+----------------+--------------+--------------+
| Average Weight | Average Height | # of Weights | # of Heights |
+----------------+----------------+--------------+--------------+
|          136.7 |          1.753 |            3 |            4 |
+----------------+----------------+--------------+--------------+
1 row in set (0.000 sec)
```

Figure 3.17 Aggregate functions: SUM and AVG

3.7.2.5.3 Aggregate Function: AVG

Syntax:

AVG(*columnName*)

AVG adds the values of all columns in the column with name, *columnName*, and then divides by the number of values which are not null. The column with name, *columnName*, has a numeric domain.

Display the average values of weight and height in the table. Also display the number of rows which contribute to each sum.

```
SELECT AVG(weight) AS "Average Weight",
       AVG(height) AS "Average Height",
       COUNT(weight) AS "# of Weights",
       COUNT(height) AS "# of Heights"
       FROM Person;
```

The average weight for the data in table, Person, is an irrational number (an unending number of digits after the decimal point). By default, seven digits are displayed, in this case, three before and six after the decimal point. Each weight is an integer (zero digits after the decimal point) and so the average should be expressed as a number with, at most, one digit after the decimal point.

Also by default, the average height is displayed by seven digits. The average of height should be expressed as a number with one more digit after the decimal point than for an individual value of height. The ROUND function is used to specify the number of digits after the decimal point. The declaration of ROUND is

ROUND(*n*, *m*)

where *n* is a number to be rounded, *m* is an integer and ROUND returns *n* rounded to *m* digits after the decimal point. The function, ROUND, is demonstrated in the SELECT command which follows.

Modify the last SELECT command to display the average value of weight with one digit after the decimal point and average height with three digits after the decimal point.

```
SELECT ROUND (AVG(weight), 1) AS "Average Weight",
        ROUND(AVG(height), 3) AS "Average Height",
        COUNT(weight) AS "# of Weights",
        COUNT(height) AS "# of Heights"
        FROM Person;
```

Figure 3.17 shows the commands which demonstrate the use of SUM and AVG. Verify the averages from the display of all rows of the table, Person, in Figure 3.16.

3.7.2.5.4 Aggregate Function: MIN

Syntax:

MIN(*columnName*)

MIN finds the minimum value of the column with name, *columnName*. The column with name, *columnName*, can have a numeric or character data type. For the case of a column with a character data type, MIN examines the characters starting at the beginning of the string of characters and finds the value that is nearest to the beginning of the alphabet, without regard to upper or lower case.

Display all the values in the table.

```
SELECT *
        FROM Person;
```

Display the minimum values of the columns, fName, weight and height.

```
SELECT MIN(fName) AS "Min First Name",
        MIN(weight) AS "Min Weight",
        MIN(height) AS "Min Height"
        FROM Person;
```

3.7.2.5.5 Aggregate Function: MAX

Syntax:

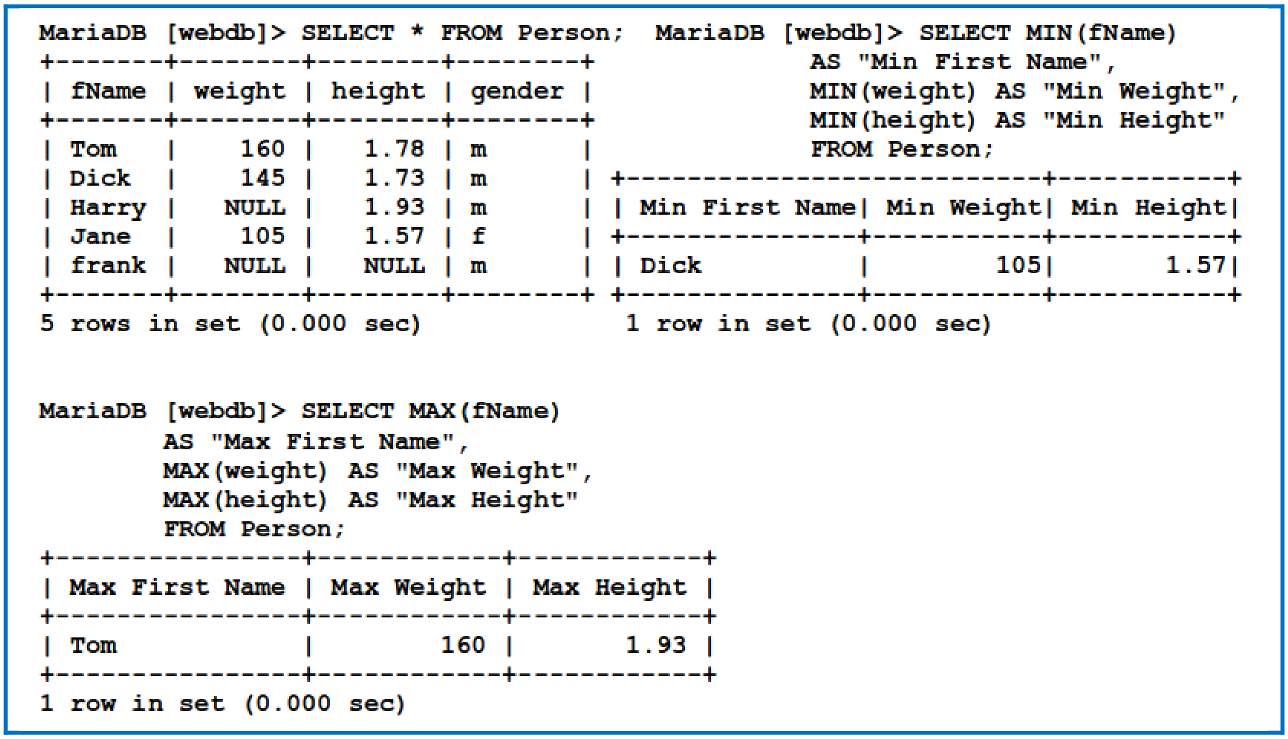
MAX(*columnName*)

MAX finds the maximum value of the column with name, *columnName*. The column with name, *columnName*, can have a numeric or character data type. For the case of a column with a character data type, MAX examines the characters starting at the beginning of the string of characters and finds the value that is nearest to the end of the alphabet, without regard to upper or lower case.

Display the maximum values of the columns, fName, weight and height.

```
SELECT  MAX(fName) AS "Max First Name",
        MAX(weight) AS "Max Weight",
        MAX(height) AS "Max Height"
        FROM Person;
```

Figure 3.18 shows first the command to display all values in the table, and then the last two commands, which demonstrate the use of MIN and MAX.



```
MariaDB [webdb]> SELECT * FROM Person;  MariaDB [webdb]> SELECT MIN(fName)
+-------+--------+--------+--------+                 AS "Min First Name",
| fName | weight | height | gender |                 MIN(weight) AS "Min Weight",
+-------+--------+--------+--------+                 MIN(height) AS "Min Height"
| Tom   |    160 |   1.78 | m      |                 FROM Person;
| Dick  |    145 |   1.73 | m      | +---------------+-----------+-----------+
| Harry |   NULL |   1.93 | m      | | Min First Name| Min Weight| Min Height|
| Jane  |    105 |   1.57 | f      | +---------------+-----------+-----------+
| frank |   NULL |   NULL | m      | | Dick          |        105|       1.57|
+-------+--------+--------+--------+ +---------------+-----------+-----------+
5 rows in set (0.000 sec)             1 row in set (0.000 sec)


MariaDB [webdb]> SELECT MAX(fName)
        AS "Max First Name",
        MAX(weight) AS "Max Weight",
        MAX(height) AS "Max Height"
        FROM Person;
+----------------+------------+------------+
| Max First Name | Max Weight | Max Height |
+----------------+------------+------------+
| Tom            |        160 |       1.93 |
+----------------+------------+------------+
1 row in set (0.000 sec)
```

Figure 3.18 Aggregate functions: MIN and MAX 40

3.7.2.6 SELECT with GROUP BY

Syntax:

SELECT *select_list*
FROM *table_list*
GROUP BY *group_by_list*

The GROUP BY clause of the SELECT command uses one or more columns, *group_by_list*, to subdivide the rows of a table into two or more groups. In table, Person, the rows of the table can be divided into two groups according to column, gender, one group being all of the males and the other, all of the females. If gender is unknown, in table, Person, and gender is not declared, NOT NULL, the value is blank (representing null) and this is treated as another group. So with one or more missing values of gender, there will be three gender groups: male, female, null. It would be better to use a character for unknown gender, ? for example. Then it would be easier to see persons in the table with unknown gender.

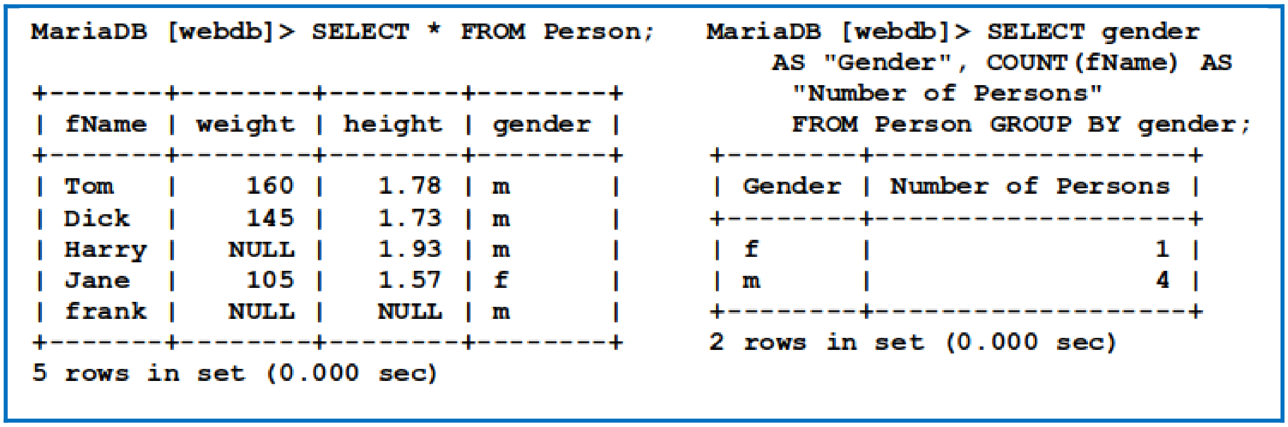
Display all rows in the table, Person.

```
      SELECT *
              FROM Person;
```

Display the number of males and females in the table. The output of this command gives two groups of gender: male and female.

```
      SELECT gender AS "Gender",
              COUNT(fName) AS "Number of Persons"
              FROM Person
              GROUP BY gender;
```

Figure 3.19 shows the command to display all rows of the table and the last command which demonstrates the use of GROUP BY.



```
MariaDB [webdb]> SELECT * FROM Person;

+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

```
MariaDB [webdb]> SELECT gender
    AS "Gender", COUNT(fName) AS
     "Number of Persons"
     FROM Person GROUP BY gender;
+--------+-------------------+
| Gender | Number of Persons |
+--------+-------------------+
| f      |                 1 |
| m      |                 4 |
+--------+-------------------+
2 rows in set (0.000 sec)
```

Figure 3.19 DML command: SELECT . . . FROM . . . GROUP BY

3.7.2.7 SELECT with GROUP BY and HAVING

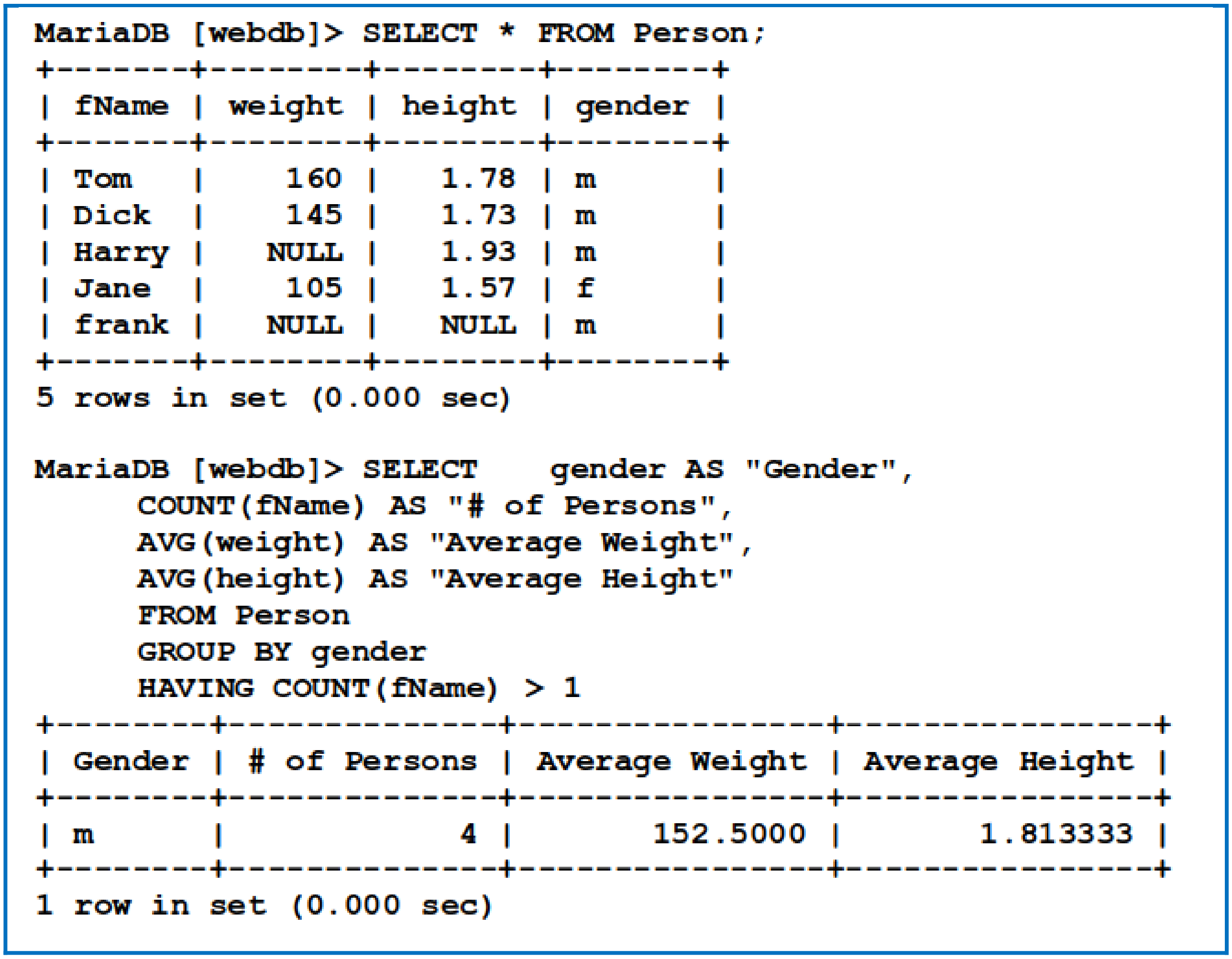
Syntax:

SELECT *select_list*
FROM *table_list*
GROUP BY *group_by_list*
HAVING *grouping_conditions*

The HAVING clause applies conditions, *grouping_conditions*, on GROUP BY (for results which are produced by grouping), just as WHERE acts as a condition for SELECT.

Display all rows in the table, Person.

```
      SELECT * FROM Person;
```



```
MariaDB [webdb]> SELECT * FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)

MariaDB [webdb]> SELECT   gender AS "Gender",
     COUNT(fName) AS "# of Persons",
     AVG(weight) AS "Average Weight",
     AVG(height) AS "Average Height"
     FROM Person
     GROUP BY gender
     HAVING COUNT(fName) > 1
+--------+--------------+----------------+----------------+
| Gender | # of Persons | Average Weight | Average Height |
+--------+--------------+----------------+----------------+
| m      |            4 |       152.5000 |       1.813333 |
+--------+--------------+----------------+----------------+
1 row in set (0.000 sec)
```

Figure 3.20 Command: SELECT . . FROM . . . GROUP BY . . . HAVING

Display the average weight and average height of the two genders in the table if the number of persons in the gender is greater than 1. This is an example of the use of GROUP BY with a condition on the grouping column using HAVING. The grouping column is fName and the condition is that COUNT(fName) > 1.

```
SELECT gender AS "Gender",
       COUNT(fName) AS "# of Persons",
       AVG(weight) AS "Average Weight",
       AVG(height) AS "Average Height"
       FROM Person
       GROUP BY gender
       HAVING COUNT(fName) > 1;
```

Figure 3.20 shows the command to display all rows of the table and the command which demonstrates the use of GROUP BY and HAVING. Note that there are four males but the average weight is the average of two numbers, the weights of Tom and Dick, since the weights of Harry and frank are missing. Also the average height is the average of three numbers since the height of frank is missing.

3.7.3 Copy data from one table to another table

Syntax:

INSERT INTO *existingDestinationTable* [*destinationColumnList*]
  SELECT *sourceColumnList*
  FROM *existingSourceTable*;

The list, *destinationColumnList*, is optional as indicated by the square brackets. The columns in *destinationColumnList* must correspond to the columns in *sourceColumnList*. If *destinationColumnList* is not specified, values for all columns in the destination table must be copied from the source table.

The INSERT INTO . . . SELECT . . . FROM command copies data from one existing table, the source table, to another existing table, the destination table. All rows in the source table with columns in *sourceColumnList* are copied. To copy fewer than all rows from the source table, use WHERE to select the rows; this is not shown in the syntax above. All columns of the source table are copied if * is used in place of *sourceColumnList*.

First copy the entire existing table, Person, to a new table, CopyOfPerson, using the command, CREATE TABLE . . . SELECT . . . FROM, demonstrated earlier. Both the structure and contents of the existing table are copied to the new table.

```
CREATE TABLE CopyOfPerson
   SELECT *
   FROM Person;
```

Display all data in table, CopyOfPerson.

```
SELECT * FROM CopyOfPerson;
```

Copy all data from existing table, Person, to existing table, CopyOfPerson.

```
INSERT INTO CopyOfPerson
   SELECT *
   FROM Person;
```

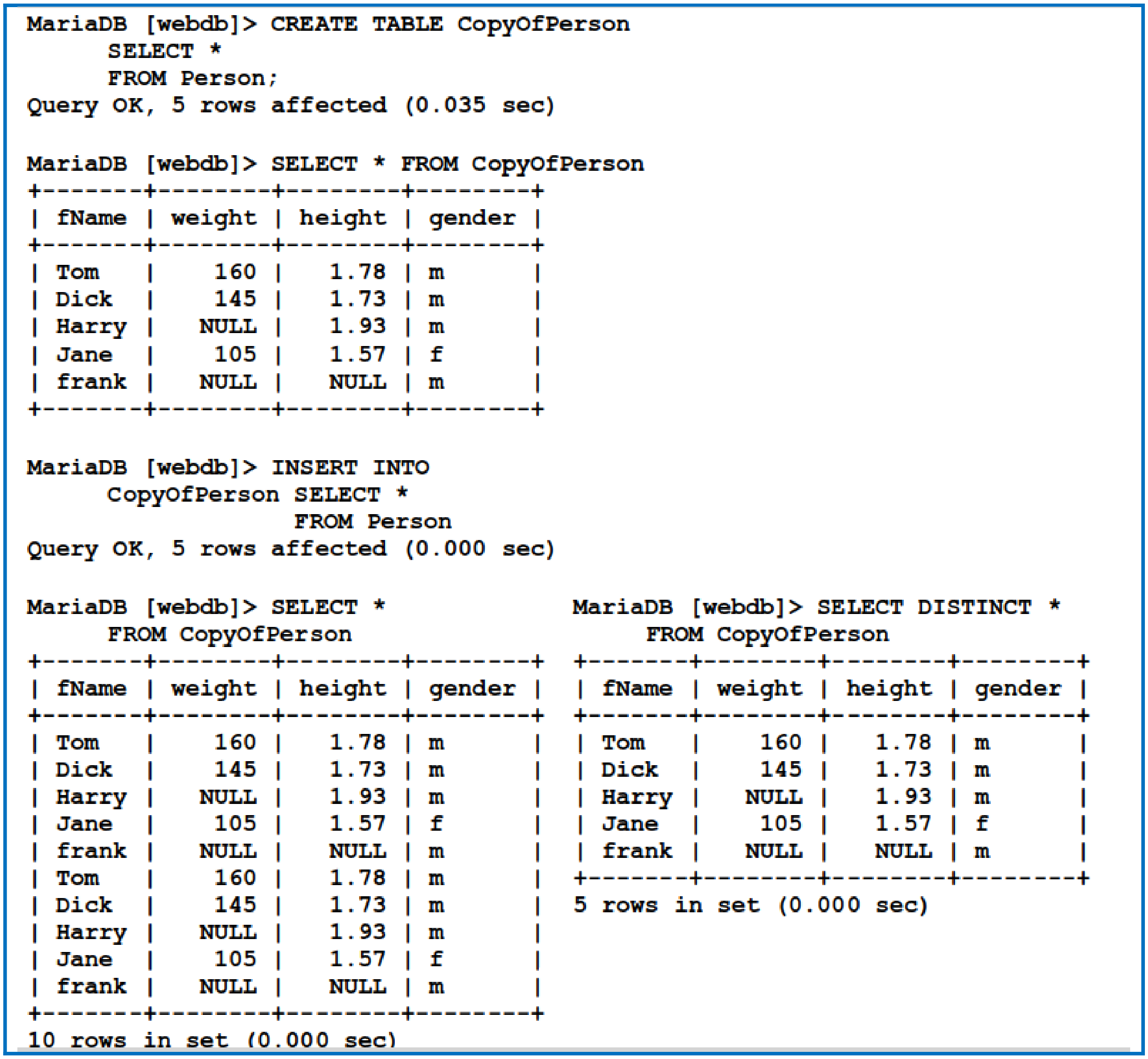
Once again, display all data in table, CopyOfPerson.

```
SELECT * FROM CopyOfPerson;
```

Display all data in table, CopyOfPerson, using the keyword, DISTINCT, to display the duplicate rows only once.

```
SELECT DISTINCT * FROM CopyOfPerson;
```

A new table, CopyOfPerson, was created which is a copy of table, Person. CopyOfPerson has five rows of data. Then the contents of all of table, Person, was copied to table, CopyOfPerson. Now, CopyOfPerson has ten rows of data. All of the rows are duplicates. This is only permitted because the table does not have a primary key, a constraint, which causes duplicate rows to be rejected. Primary keys will be covered in the next chapter. In practice, there is rarely a reason to insert duplicate rows into a table. It is done here only as a demonstration of the command to copy data from one table to another. It also allows the demonstration of the keyword, DISTINCT, in a SELECT command.



```
MariaDB [webdb]> CREATE TABLE CopyOfPerson
      SELECT *
      FROM Person;
Query OK, 5 rows affected (0.035 sec)

MariaDB [webdb]> SELECT * FROM CopyOfPerson
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+

MariaDB [webdb]> INSERT INTO
      CopyOfPerson SELECT *
                  FROM Person
Query OK, 5 rows affected (0.000 sec)

MariaDB [webdb]> SELECT *             MariaDB [webdb]> SELECT DISTINCT *
      FROM CopyOfPerson                    FROM CopyOfPerson
+-------+--------+--------+--------+  +-------+--------+--------+--------+
| fName | weight | height | gender |  | fName | weight | height | gender |
+-------+--------+--------+--------+  +-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |  | Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |  | Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |  | Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |  | Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |  | frank |   NULL |   NULL | m      |
| Tom   |    160 |   1.78 | m      |  +-------+--------+--------+--------+
| Dick  |    145 |   1.73 | m      |  5 rows in set (0.000 sec)
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
10 rows in set (0.000 sec)
```

Figure 3.21 DML command: INSERT INTO. . . SELECT . . . FROM

Figure 3.21 shows the commands above which demonstrate the use of:
INSERT INTO . . . SELECT . . . FROM.

3.7.4 Modify data in a table

Syntax:

UPDATE *tableName*
SET *columnAssignments*

WHERE *condition*;

The UPDATE command modifies the value of a column in a table. SET specifies the column or columns to be modified and the new values of the columns. WHERE is used to choose the rows which are modified. If WHERE is omitted, all rows are modified.

Display all data in the table, Person.

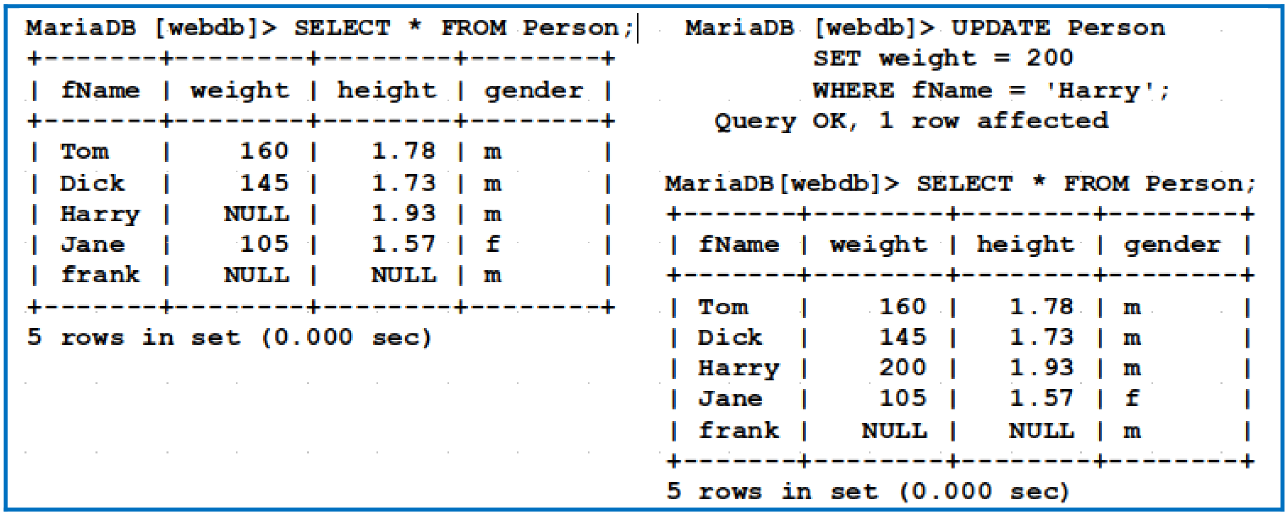
```
   SELECT * FROM Person;
```

Finally, Harry's weight is known. Update the weight of Harry from null (no weight) to 200. It is essential to include WHERE in the command because it specifies which row is to be updated. If WHERE is omitted, everybody in table, Person, would be assigned a weight of 200.

```
   UPDATE Person
      SET weight = 200
      WHERE fName = 'Harry';
```

Display all data in table, Person.

```
   SELECT * FROM Person;
```



```
MariaDB [webdb]> SELECT * FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |   NULL |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

```
MariaDB [webdb]> UPDATE Person
                 SET weight = 200
                 WHERE fName = 'Harry';
   Query OK, 1 row affected

MariaDB[webdb]> SELECT * FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

Figure 3.22 DML command: UPDATE . . . SET . . . WHERE

Figure 3.22 show the last commands which demonstrate the use of UPDATE . . . SET . . . WHERE.

3.7.5 Delete one or more rows of a table

Syntax:

DELETE FROM *tableName* WHERE *condition*;

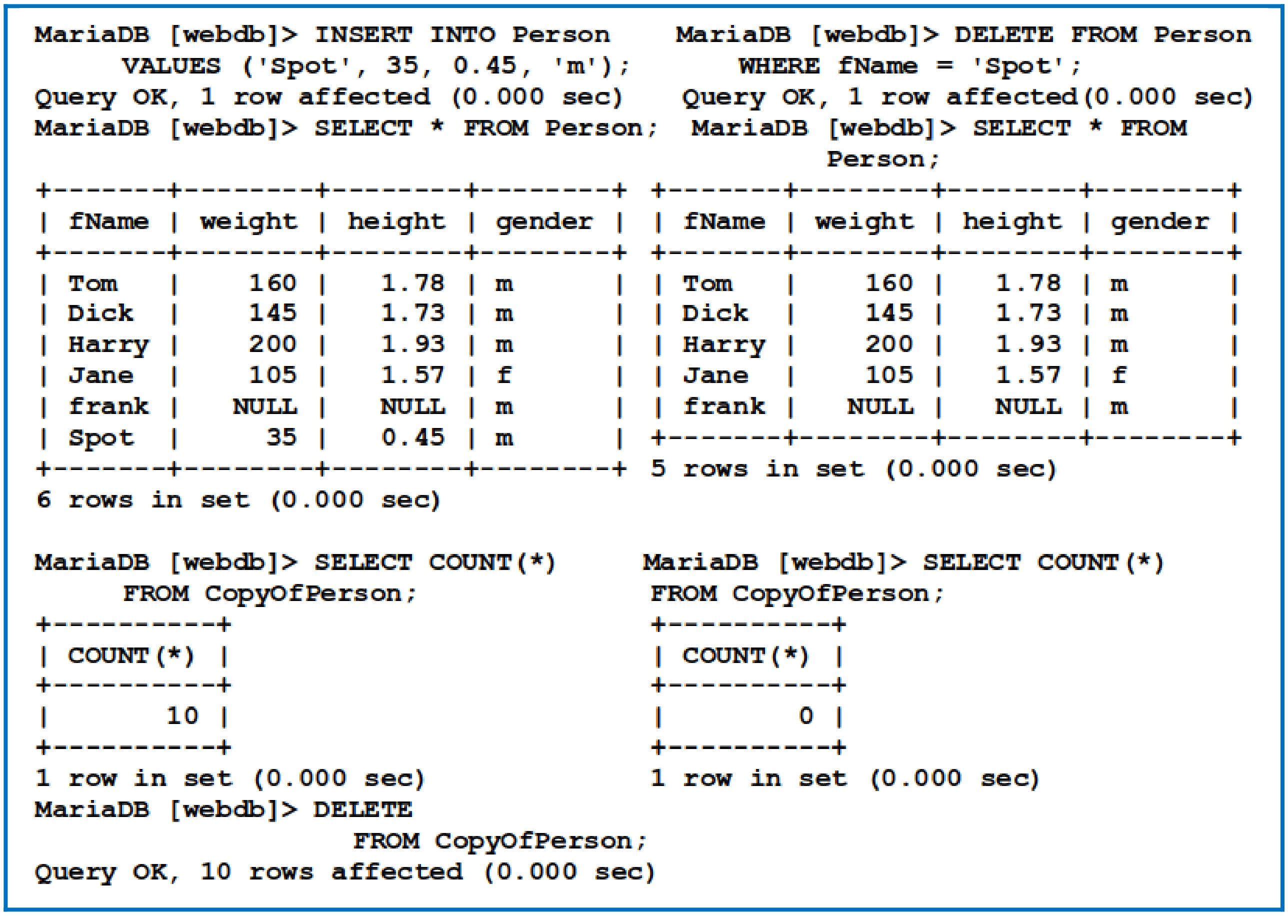
The DELETE FROM command deletes (removes) one or more rows from a table. The rows to be deleted are determined by WHERE. If WHERE is omitted, *condition* is assumed to be "TRUE" for all rows of the table and all rows are deleted. Before using the DELETE FROM command it is advisable to make a backup copy of the table using the CREATE TABLE . . . SELECT . . . FROM command.

First insert another row in the table, Person, by the command demonstrated earlier.

```
INSERT INTO Person
     VALUES ('Spot', 35, 0.45, 'm');
```

Display all data in table, Person.

```
SELECT * FROM Person;
```



```
MariaDB [webdb]> INSERT INTO Person    MariaDB [webdb]> DELETE FROM Person
     VALUES ('Spot', 35, 0.45, 'm');       WHERE fName = 'Spot';
Query OK, 1 row affected (0.000 sec)    Query OK, 1 row affected(0.000 sec)
MariaDB [webdb]> SELECT * FROM Person;  MariaDB [webdb]> SELECT * FROM
                                                Person;
+-------+--------+--------+--------+ +-------+--------+--------+--------+
| fName | weight | height | gender | | fName | weight | height | gender |
+-------+--------+--------+--------+ +-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      | | Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      | | Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      | | Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      | | Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      | | frank |   NULL |   NULL | m      |
| Spot  |     35 |   0.45 | m      | +-------+--------+--------+--------+
+-------+--------+--------+--------+ 5 rows in set (0.000 sec)
6 rows in set (0.000 sec)

MariaDB [webdb]> SELECT COUNT(*)     MariaDB [webdb]> SELECT COUNT(*)
     FROM CopyOfPerson;               FROM CopyOfPerson;
+----------+                          +----------+
| COUNT(*) |                          | COUNT(*) |
+----------+                          +----------+
|       10 |                          |        0 |
+----------+                          +----------+
1 row in set (0.000 sec)              1 row in set (0.000 sec)
MariaDB [webdb]> DELETE
                   FROM CopyOfPerson;
Query OK, 10 rows affected (0.000 sec)
```

Figure 3.23 DML command: DELETE FROM . . . WHERE

By mistake, the dog, Spot, which belongs to Dick and Jane, has been inserted in the table, Person. Delete the row with fName, Spot.

```
DELETE FROM Person
  WHERE fName = 'Spot';
```

Display all data in table, Person.

```
SELECT * FROM Person;
```

Delete all rows in table, CopyOfPerson. First display the number of rows in the table, then delete all rows and finally display the number of rows again.

```
SELECT COUNT(*) FROM CopyOfPerson;
DELETE FROM CopyOfPerson;
SELECT COUNT(*) FROM CopyOfPerson;
```

Figure 3.23 shows the commands which demonstrate the use of DELETE FROM.

3.8 DCL (Data Control Language) Demonstrations

The Data Control Language of SQL allows the management of transactions and security. Security is also called privileges.

3.8.1 Transactions

A transaction is a set of one or more DML commands (SELECT, UPDATE, INSERT, DELETE). The transaction command, COMMIT, saves a transaction on disk. All DML commands executed since the last execution of COMMIT constitute a transaction

The transaction command, ROLLBACK, deletes a transaction, that is, undoes changes to the database since the last COMMIT command.

3.8.1.1 COMMIT and ROLLBACK

Syntax:

```
COMMIT;
ROLLBACK;
```

A session starts when a user logs onto the database. When data is inserted, updated or deleted it is important to be able to make the changes permanent; this is called committing. In case there is an error it is also important to be able to reverse or undo changes which were made; this is called rollback. The process of committing (making changes persistent) and rollback (undoing changes) is done by the commands, COMMIT and ROLLBACK. The commands do not require an argument.

There are several ways that a transaction is terminated. A transaction ends when either a COMMIT or ROLLBACK command is executed. The execution of a DDL command such as CREATE, DROP, RENAME, or ALTER also terminates a transaction. If a user disconnects from MySQL, the current transaction is committed and so all the changes are made permanent. An abnormal termination of a session causes the current transaction to be rolled back.

The commit operation can be made immediate (commit after every command) without using the explicit command, COMMIT. This is done by setting the variable, autocommit, to ON. First show the value of the variable using the mysql command:

SHOW VARIABLES LIKE '*pattern*';

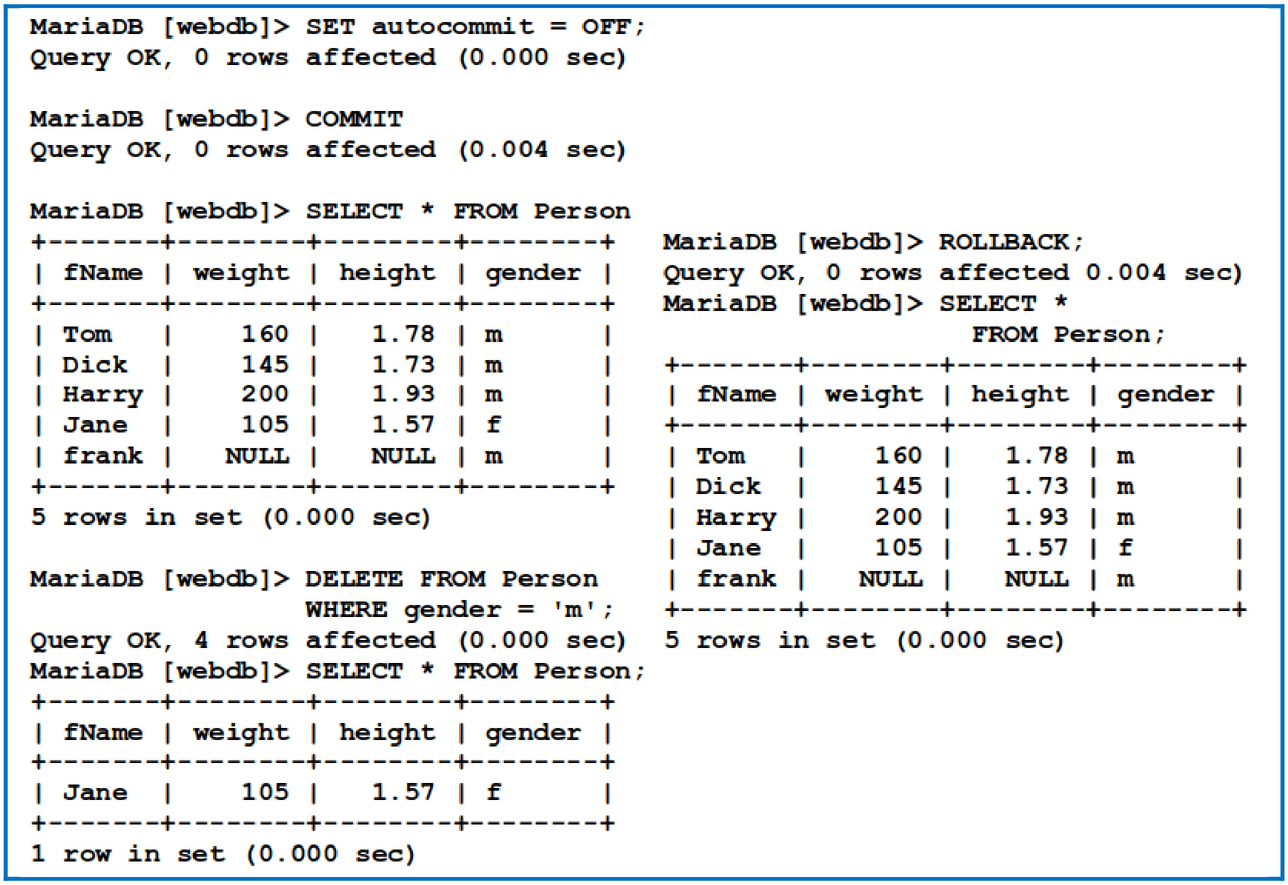
The *pattern* can be, 'autocommit', or to save typing use the *pattern*, 'a%', which will display all variables which start with the letter, a. The symbol, %, is the wildcard for zero or more characters.
Set the value of the variable, autocommit, to OFF or ON using the command:

SET *variableName* = *expression*;

The *variableName* is autocommit and the *expression* is OFF or ON.

Show the value of autocommit. Assuming the value is ON, set the value to OFF. Then show the value again to confirm that the value is set to OFF. Finally, set the value to ON. The four commands are given below. The commands are not shown in a figure.

```
SHOW VARIABLES LIKE 'a%';
SET autocommit = OFF;
SHOW VARIABLES LIKE 'a%';
SET autocommit = ON;
```



```
MariaDB [webdb]> SET autocommit = OFF;
Query OK, 0 rows affected (0.000 sec)

MariaDB [webdb]> COMMIT
Query OK, 0 rows affected (0.004 sec)

MariaDB [webdb]> SELECT * FROM Person
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)

MariaDB [webdb]> DELETE FROM Person
                 WHERE gender = 'm';
Query OK, 4 rows affected (0.000 sec)
MariaDB [webdb]> SELECT * FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Jane  |    105 |   1.57 | f      |
+-------+--------+--------+--------+
1 row in set (0.000 sec)
```

```
MariaDB [webdb]> ROLLBACK;
Query OK, 0 rows affected 0.004 sec)
MariaDB [webdb]> SELECT *
                     FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

Figure 3.24 DCL commands: COMMIT and ROLLBACK

Once a transaction is committed by the use of the COMMIT command, it cannot be undone by the ROLLBACK command. Also, if the variable, autocommit, is set to ON, the transactions (now each transaction is one command) are committed immediately and cannot be undone by a ROLLBACK. The changes before the last commit operation can only be undone by another transaction; such a transaction is called a compensating transaction. For example, the deletion of one row is compensated by the insertion of the same row.

Demonstrate COMMIT and ROLLBACK by first executing COMMIT, then deleting a row, then executing ROLLBACK to undo the deletion of the row. First set autocommit to OFF in case autocommit is set to ON.

```
SET autocommit = OFF;
COMMIT;
```

```
SELECT * FROM Person;
DELETE FROM Person WHERE gender = 'm';
SELECT * FROM Person;
ROLLBACK;
SELECT * FROM Person;
```

Figure 3.24 shows the commands which demonstrate the use of COMMIT and ROLLBACK.

3.8.1.2 SAVEPOINT and ROLLBACK TO SAVEPOINT

Syntax:

SAVEPOINT *nameOfSavepoint*
ROLLBACK TO SAVEPOINT *nameOfSavepoint*

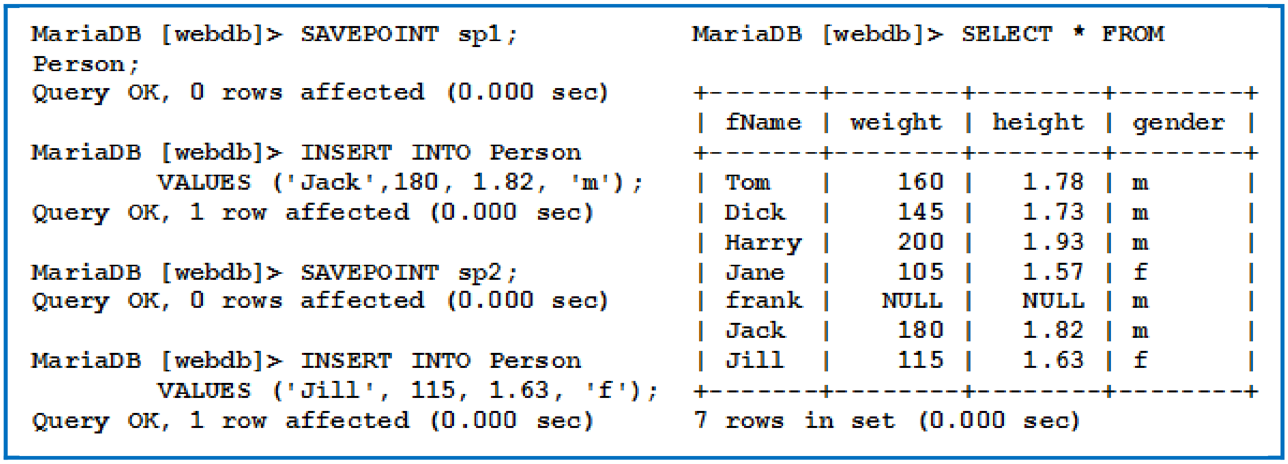
Another method of controlling changes to a database is by the use of a milestone or snapshot created by the command, SAVEPOINT. A snapshot is the status of the database at a given instant of time. The snapshot is given a name, *nameOfSavepoint*, which is the argument of the SAVEPOINT command. One or more SAVEPOINT commands can be executed after the last COMMIT command. When a COMMIT command is executed, all *nameOfSavepoint* are deleted.

To revert to the state of the database saved by SAVEPOINT, the ROLLBACK TO SAVEPOINT command is used with one of the arguments used by a SAVEPOINT command. When a rollback to savepoint is executed for *nameOfSavepoint*, all subsequent (later) *nameOfSavepoint* are deleted. The command, ROLLBACK TO SAVEPOINT, rolls back a transaction without ending the current transaction. All savepoints are deleted if COMMIT or ROLLBACK is executed.

Create two savepoints, with names sp1 and sp2, and after each insert a row into the table. Then display all data in the table.

```
SAVEPOINT sp1;
INSERT INTO Person
     VALUES ('Jack',180, 1.82, 'm');
SAVEPOINT sp2;
INSERT INTO Person
     VALUES ('Jill', 115, 1.63, 'f');
SELECT * FROM Person;
```

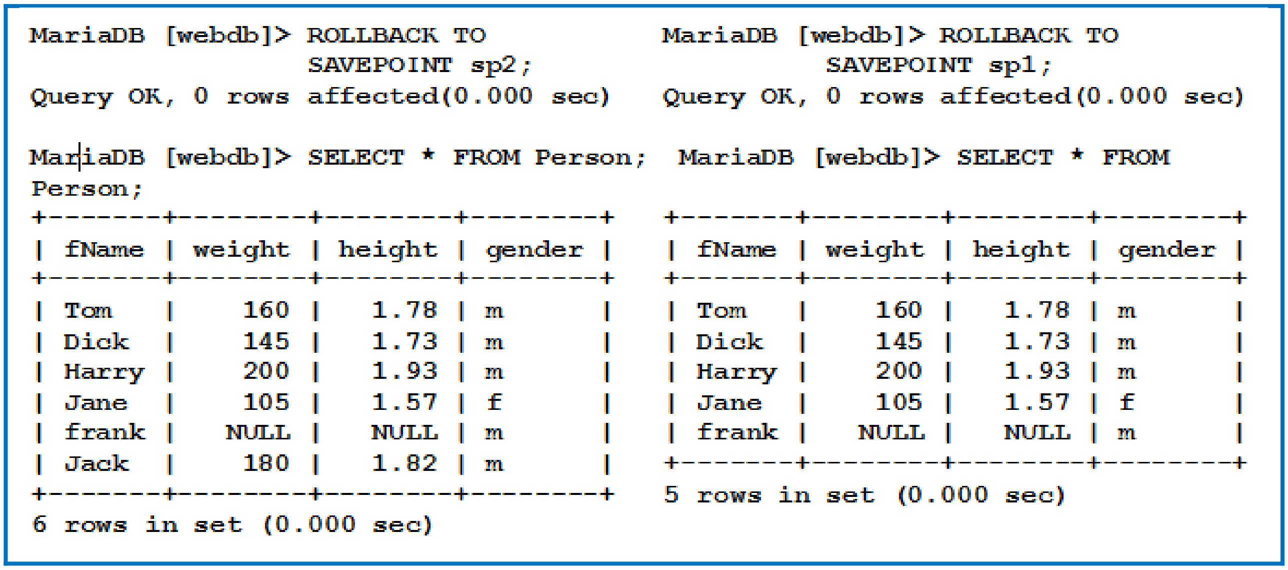
Figure 3.25 shows these commands and the resulting status of the table Person.



```
MariaDB [webdb]> SAVEPOINT sp1;
Query OK, 0 rows affected (0.000 sec)

MariaDB [webdb]> INSERT INTO Person
        VALUES ('Jack',180, 1.82, 'm');
Query OK, 1 row affected (0.000 sec)

MariaDB [webdb]> SAVEPOINT sp2;
Query OK, 0 rows affected (0.000 sec)

MariaDB [webdb]> INSERT INTO Person
        VALUES ('Jill', 115, 1.63, 'f');
Query OK, 1 row affected (0.000 sec)
```

```
MariaDB [webdb]> SELECT * FROM
Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
| Jack  |    180 |   1.82 | m      |
| Jill  |    115 |   1.63 | f      |
+-------+--------+--------+--------+
7 rows in set (0.000 sec)
```

Figure 3.25 DCL command: SAVEPOINT



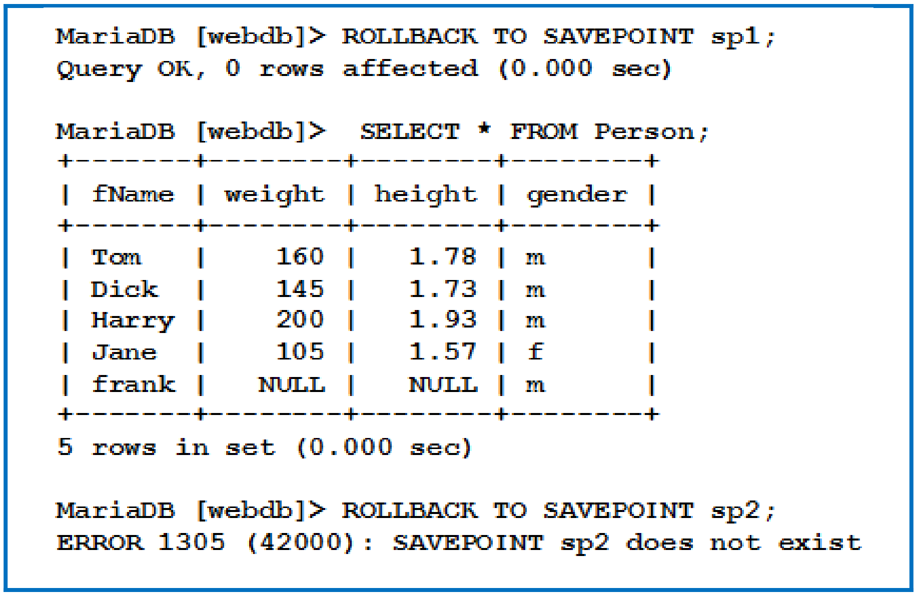
```
MariaDB [webdb]> ROLLBACK TO
                 SAVEPOINT sp2;
Query OK, 0 rows affected(0.000 sec)

MariaDB [webdb]> SELECT * FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
| Jack  |    180 |   1.82 | m      |
+-------+--------+--------+--------+
6 rows in set (0.000 sec)
```

```
MariaDB [webdb]> ROLLBACK TO
                 SAVEPOINT sp1;
Query OK, 0 rows affected(0.000 sec)

MariaDB [webdb]> SELECT * FROM
Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)
```

Figure 3.26 DCL command: ROLLBACK TO SAVEPOINT

If a number of SAVEPOINTs are created, the order of the creation and roll back is important; the order should be last created, first used. If an earlier (older) SAVEPOINT is used, later (younger) SAVEPOINTs are no longer valid. If an attempt is made to rollback to the first (older) savepoint sp1, followed by an attempt to rollback to the second (younger) savepoint, sp2, it would lead to an error since once sp1 is used, sp2 would not exist because it was created after sp1. Figure 3.26 demonstrates the use of ROLLBACK and SAVEPOINT using the last saved, first used principal.

An attempt to use a more recent (younger) savepoint after using a less recent (older) savepoint results in an error; the more recent savepoint having been deleted due their use in an inconsistent order, as demonstrated in Figure 3.27.



```
MariaDB [webdb]> ROLLBACK TO SAVEPOINT sp1;
Query OK, 0 rows affected (0.000 sec)

MariaDB [webdb]>  SELECT * FROM Person;
+-------+--------+--------+--------+
| fName | weight | height | gender |
+-------+--------+--------+--------+
| Tom   |    160 |   1.78 | m      |
| Dick  |    145 |   1.73 | m      |
| Harry |    200 |   1.93 | m      |
| Jane  |    105 |   1.57 | f      |
| frank |   NULL |   NULL | m      |
+-------+--------+--------+--------+
5 rows in set (0.000 sec)

MariaDB [webdb]> ROLLBACK TO SAVEPOINT sp2;
ERROR 1305 (42000): SAVEPOINT sp2 does not exist
```

Figure 3.27 DCL command: Incorrect order of rollbacks

3.8.2 Privileges

The access to tables created by one user and given to another user is controlled by the security features of SQL which are also called privileges. Security is enforced by the use of the GRANT and REVOKE commands. The owner of a table or the database administrator can allow access to data in the table to specific users by the GRANT command and remove (disallow) the access privilege by the REVOKE command. The owner of a table is the user who created the table. The owner has all privileges for the table. The database administrator has all privileges for all tables of all users.

3.8.2.1 GRANT

Syntax:

GRANT *privilege_type* ON *tableName* TO *user*;

The value of *user*, is *userName@hostName.* The user, *userName*, is a synonym for *userName@*' %', where % is the wildcard for zero more characters.

The following GRANT command gives the user with name, user1, access to add data (using INSERT) to table, Person.

```
GRANT INSERT ON Person TO user1;
```

3.8.2.2 REVOKE

Syntax:

REVOKE *privilege_type* ON *tableName* FROM *user*|PUBLIC;

The following REVOKE command disallows the access given by the GRANT command above:

```
REVOKE INSERT ON Person FROM user1;
```

4 SQL Constraints

4.1 Introduction

SQL constraints restrict the data that can be put into a database, in order to control the integrity of the database. Integrity in this context means reliability or correctness. Constraints ensure that the data is accurate and consistent. When data is added or changed in the tables of a database, the constraints only allow valid data. If the data violates the constraints (data is invalid), the addition or change to the database is rejected by the constraints.

Valid data is described by statements expressed in words which are sometimes called business rules. Then the rules are applied to the database as either schema-based constraints or application-based constraints.

All integrity constraints can be demonstrated by one table except for one constraint: the referential integrity constraint. This constraint relates two tables by means of a foreign key which is an attribute that links the two tables. So the database which is used to demonstrate the constraints in this chapter has two tables. The two tables are related by a many-to-one relationship. SQL relationships are the subject of the next chapter.

4.2 Classification

The most general classification of constraints divides constraints into three categories.

4.2.1 Model-based constraints

Model-based constraints are characteristics of the tables in the relational model.
For example, there cannot be duplicate tuples (rows) in a table (although this is permitted in commercial implementations of SQL).

4.2.2 Schema-based constraints

Schema-based constraints are SQL constraints which are expressed by DDL (data definition language).
They are restrictions on the values of attributes and are included in the CREATE TABLE command together with the definitions of the attributes.

4.2.3 Application-based constraints

Application-based constraints are enforced by application programs. They must be used for constraints which cannot be expressed by DDL although application programs can be used to enforce all constraints. DDL can be used to enforce only some constraints. However, it is simpler to use schema-based constraints than application-based constraints. Also if more than one application accesses the database, a schema-based constraint applies to all applications, and it is not necessary to remember to include the constraint in each individual application. A constraint should always be enforced by the schema instead of by an application, when possible.

4.3 Database Concepts

The relational database is introduced in Chapter 1. This section provides a review and adds some more information about relational database concepts.

4.3.1 Entity

An entity is a thing which can be a person, place, other physical object, concept, organization, event.
Examples of entities are Student (person), Car (physical object), Employer (organization).

Note that there is a constraint called an entity integrity constraint.

4.3.2 Entity set

An entity set is a collection of entities of the same kind.
Examples of entity sets are students enrolled in Computer Science, cars for sale at a motor vehicle dealership, employers who hire programmers.
An entity set could be empty or contain one entity, but in general, entity sets contain many entities. For example, the entity set, Student, could contain all of the students in a database course.

4.3.3 Attribute

An attribute is a property or characteristic of an entity.
Examples of attributes of entity, Student, are firstName and weight. Examples of attributes of entity, Car, are color and weight. Examples of attributes of entity, Employer, are name, address and telephone.
For example one entity, Student, could have firstName, Tom, and weight, 160.
By convention an entity name is capitalized and an attribute name is not capitalized.

4.3.4 Domain

A domain is a constraint on the value of an attribute.
The domain corresponds to a data type in a programming language, except a domain both specifies the data type and restricts the value of the attribute. A domain is mandatory for every attribute in a database.
Examples of domains of the attributes of entity, Student, are the following.

Attribute: firstName	Domain: string, 15 characters or less
Attribute: weight	Domain: integer, 3 digits or less

Note that the domain for weight is from -999 to 999. So additional constraints should be imposed on weight, which can be done by application programming.

4.3.5 Schema

A schema is the structure of an entity which is specified by the name of the entity followed, in brackets, by the list of attributes of the entity.
Examples of schemas are the following.

Student(firstName, weight)
Car(color, weight)

Employer(name, address, telephone)

4.3.6 Tuple

A tuple is one entity of an entity set. So the terms, entity and tuple, have the same meaning.

4.3.7 Table

A table is a representation of an entity set where the columns of the table are the attributes of the entity set and the rows of the table are the tuples of the entity set.
A table is also called a relation. So the three terms, entity set, relation and table, have the same meaning.
A table has a name, which is the name of the entity.
Each column of the table has a name which is the name of the attribute.
Rows of a table do not have names; rows are identified by the values in one or more columns of a row.
An example of a table of the entity set, Student, with two rows is the following.

firstName	weight
Tom	160
Dick	145

4.4 Foreign Key

4.4.1 Definition

A foreign key is one (or more) attributes in one table, A, which also must be in a related table, B. The linkage (connection) between the two tables, A and B, is created by the matching values of the foreign key in the two tables. The foreign key in A must reference either the primary key or a unique key in B, but usually references the primary key in B.

4.4.2 Parent and Child Tables

In a relationship between two tables, the table with the foreign key is called the child table and the other table is called the parent table. Also the words, parent and child, can be used as nouns when it is clear from the context that they refer to tables.

The child table depends on the parent table which means that the parent table must exist before the child table can be created. Therefore, when the tables are created, the parent tables must be created first, and when the tables are removed, the child tables must be removed first. Also, when the tables are populated with data, the parent tables must be populated first, and when the contents of tables are deleted, the contents of the child tables must be deleted first.

Tables that participate in more than one relationship could be a parent table in one relationship and a child table in another relationship. The order of creation and removal of these tables and the order of population and deletion of the tables must be determined for each particular case.

The terminology, parent table and child table, is well-chosen because the parent table must be created before the child table, which is the same order for creation of people as well as tables. Also, a person can be both a parent in one relationship and (always) a child in another relationship, just as a table can have both types of relationships. Furthermore, a person has one parent (the mother) but can have more than one child, just as a parent tuple can be related to more than one child tuple, but a child tuple is only related to one parent tuple.

4.4.3 Null

An attribute that has no value (the value is missing) is said to be null. Null means, has no value. So the statement, the attribute is null, means, the attribute has no value. Strictly speaking it is not correct to say, the value of an attribute is null, because null is not a value; it is the absence of a value.

An attribute with name, num1, and domain, number, which has value 0 is not null. An attribute with name, string1, and domain, char, which contains a zero-length string (represented by “”), is not null (although it was in earlier versions of Oracle). However, if both num1 and string1 do not have values, they are both null. Furthermore, there is only one kind of null; there is no distinction between the null in num1 and the null in string1. There is no domain associated with null.

There are three reasons that an attribute can be null:
(a) value is known but not recorded
(b) value is unknown
(c) value is not applicable
In the first two cases the value of the attribute exists. In the last case the value does not exist.
As an example consider entity, Student, with three attributes: first name, weight, date of graduation.
So the schema of Student is
Student (firstName, weight, dateOfGraduation)
Suppose an entity of Student is firstName: Tom, weight: 160, dateOfGraduation: . The date of graduation is missing. There are only two possibilities; Tom graduated or Tom has not graduated. If Tom graduated, either the date is known but not recorded or the date is unknown, but in both cases the date exists. If Tom has not graduated, the date is inapplicable, that is, the date does not exist.

Note that there is a constraint with keywords, NOT NULL.

4.5 Schema-based Constraints

This chapter covers schema-based constraints. All SQL schema-based constraints are declared by the DDL commands, CREATE TABLE or ALTER TABLE, that is, when the schema of the database is defined or changed. There is only one mandatory constraint and that is the domain which is imposed on all attributes. A domain is a data type with a restricted range of values. Domains are covered in the previous chapter in the section, Domain Constraints. Domains are not discussed further in this chapter.

The schema-based constraints (excluding the domain constraint) can be categorized in different ways. One logical method is to divide constraints into two categories: constraints which restrict the value of an attribute (value of one column of a table) and constraints which restrict the value of a tuple (entire row of a table). The categories that are used below are not labelled as attribute constraints or tuple constraints, but they are either one or the other.

The schema-based constraints are listed and briefly described in Table 4.1. They are divided into four categories in the table. The keywords are listed which are used in the declaration of the constraint in the DDL command. The first two groups, entity and referential integrity, are tuple constraints and the last two are attribute constraints. However, the CHECK constraint can also restrict the value of a tuple.

Table 4.1 Schema-based Constraints

Category	Keywords	Description
1. Entity integrity	PRIMARY KEY	different value for every tuple and cannot be null
	UNIQUE	different value for every tuple and can be null
2. Referential integrity	FOREIGN KEY	value in child table must exist in parent table
3. Missing data	NOT NULL	value of attribute cannot be missing
	DEFAULT	value of attribute given if not specified when tuple is inserted
4. Value	CHECK	value of attribute or tuple is restricted for insert and update
	DOMAIN	declaration of user-defined data type

4.5.1 Entity integrity constraint, also called key constraint

There are two kinds of entity integrity constraints with keywords:

(a) PRIMARY KEY
(b) UNIQUE

An entity integrity constraint applies to one table. It states that one or more attributes (columns) must have a different value or set of values for every tuple (row) of the table, and so it is a tuple constraint. The particular attribute or attributes is called a key. The purpose of the key is to identify each tuple in a table without ambiguity (uniquely). A key for a table is determined from the meaning of the attributes.

The declaration of one or more attributes as a primary key or as unique achieves the same result; no two tuples can have the same primary key or the same unique attribute (or attributes). However, there are two differences between PRIMARY KEY and UNIQUE.

(i) There can be only one primary key in a table but any number of unique declarations.

(ii) Each attribute of a primary key must have a value, that is, it cannot be null. Also, it is not necessary to declare the attribute of a primary key as NOT NULL. Attributes declared as unique can be null, unless declared as NOT NULL.

4.5.2 Referential integrity constraint

There is one kind of referential integrity constraint with keyword:

(a) FOREIGN KEY

A foreign key is one or more attributes in a table which are also in another table. The next section, Foreign Key, discusses foreign key in more detail.

A referential integrity constraint applies to two tables. It is declared in one table and the declaration specifies another table. It states that a particular attribute (or attributes) in a tuple in one table (the child table) must also exist in a tuple in another table (the parent table). So a referential integrity constraint is a tuple constraint. The particular attribute (or attributes) in the child table is called the foreign key. The attribute (or attributes) in the parent table which are referenced by the foreign key can be either the primary key or unique, but in almost all cases is the primary key. The foreign key can be null, unless it is declared to be NOT NULL.

The foreign key in the child table and the referenced primary key in the parent table do not need to have the same name, but often do have the same name. However,

(i) the number of attributes and
(ii) the domain of the attributes

which constitute the foreign key and referenced primary key must be the same.

A foreign key for a table is determined by the semantics (meaning) of the child and parent tables, the relationship between them and the attributes in the tables. The foreign key is the mechanism by which tables are linked together (related) in a database. The foreign key labels the entities (rows) in one table (child) that effectively belong to entities (rows) in another table (parent).

4.5.3 Missing data constraint

There are two kinds of missing data constraints with keywords:

(a) NOT NULL
(b) DEFAULT

Missing data in an attribute is permitted in a table, unless the attribute is part of a primary key. When data is missing from an attribute the attribute has no value and is said to be null. The declaration, NULL, is the default and it is not necessary to include NULL in the declaration of an attribute which can be null. If the value of an attribute cannot be null, it is declared with the keywords,

NOT NULL

The value of an attribute can have a default value, *value*, defined by,

DEFAULT *value*

on the same line as the declaration of the attribute. In this case, if a tuple is inserted in the database and the value of the attribute is not given, the attribute is assigned the default value.

4.5.4 Value constraint

There are two kinds of value constraints with keywords:

(a) CHECK
(b) DOMAIN

A check constraint is an additional restriction on the value of the attribute in addition to the restriction imposed by the domain. Also a CHECK constraint can specify a restriction on a tuple (as well as on an attribute). The CHECK constraint only applies for operations, INSERT and UPDATE. If a constraint can be violated by DELETE, for example, an assertion or trigger must be used, which are application-based constraints and are covered in a later chapter. The CHECK constraint and an assertion both evaluate a condition and return either true or false. If the result is false, the constraint is violated and the operation is rejected. If the result is true, the constraint is satisfied and the operation is permitted.

A domain constraint is the declaration of a user-defined data type which must include a data type defined by SQL and can also include additional constraints. This constraint is analogous to a type definition statement in a programming language (for example, typedef in C or C++).

4.5.5 Constraints applied to one table

All constraints can be demonstrated using one table, except for the referential integrity constraint. The table, Person, used in the last chapter does not have any constraints except NOT NULL for attribute, fName, and the mandatory constraint, domain, for every attribute. However, table, Person, can be used to give examples of all constraints, except the referential integrity constraint which requires two tables. Examples of the constraints using Person are in Table 4.2. From the last chapter, the description of the attributes of table, Person, displayed by the command, DESC Person, is the following.

```
Name                 Null?                Type
-----------------------------------------------------------------------
FNAME               NOT NULL              VARCHAR (10)
WEIGHT              DECIMAL(3)
HEIGHT              DECIMAL(3,2)
GENDER              CHAR(1)
```

From the last chapter, the contents of the table, Person, displayed by the command, SELECT * FROM Person, is the following.

```
FNAME               WEIGHT               HEIGHT               G
-------------------------------------------------------------------------------------
Tom                 160                  1.78                 m
Dick                145                  1.73                  m
Harry               1.93                                       m
Jane                105                  1.57                  f
```

A more formal terminology for the two tables in a relationship is referenced, instead of parent, and referencing, instead of child. However, a disadvantage of the two names, referenced table and referencing table, is that they sound almost the same. Secondly, the words referenced and referencing can only be used as adjectives. Another terminology that is used is owner table, instead of parent table, and dependent table, instead of child table. Note that the terms, parent and child, are used in Oracle documentation and, in particular, in error messages generated by Oracle.

Table 4.2 Examples of Schema-based Constraints

Description of attribute	Invalid data	Prevented by constraint
fName is 10 characters or less	fName = 'Tommmmmmmmy'	domain: VARCHAR (10)
fName is different in every row	fName = 'Tom' (in two rows)	PRIMARY KEY or UNIQUE
weight is a number	weight = abc	domain: DECIMAL(3)
weight is a 3-digit integer	weight = 1000	domain: DECIMAL(3)
height is a 1-digit real number	height = 10	domain: DECIMAL(3,2)
height is in meters	height = 9.5	CHECK (height < 2.2)
gender is one character	gender = 'red'	domain: CHAR(1)
gender is either m or f	gender = 'r'	CHECK (gender in ('m', 'f'))
gender must have a value	gender =	NOT NULL
gender is m if no value is entered	gender =	DEFAULT 'm'

4.5.6 Need for a Foreign Key

The need for a foreign key arises naturally from a situation such as the following. This situation leads to the Crs-Dpt relationship used as the sample database in this chapter. Crs is an abbreviation for course and Dpt is an abbreviation for department. Abbreviations are used as names in this chapter because the names, Course and Department, are used in the next chapter in a different database.

Suppose a university records the information about the departments and the courses offered by departments in two lists. The department list has columns: numerical code (dCode), name (dName), location (location). An example of two rows in the list is the following. The name of the list is Dpt.

The course list has columns: alphanumeric code (cNumber), course name (cName), number of credits (credits). An example of four rows in the list is the following. The name of the list is Crs.

Courses belong to departments. A course belongs to only one department and one department has many courses. This is an example of a many-to-one relationship; a department has many courses, a course belongs to only one department.

Dpt

dCode	dName	location
1	Computer Science	EV building
2	Physics	Science building

Crs

cNumber	cName	credits
COMP 228	System Hardware	3
COMP 229	System Software	3
PHYS 245	Mechanics	3
PHYS 253	Electricity & Magnetism	3

It is important to realize that all lists which the university keeps are not necessarily related to each other. For example, the university would keep a list of non-academic departments (Human Resources, Physical Resources, and so on) which would not be related to either of the lists, Dpt and Crs.

How can the department to which each course belongs be recorded? The answer should be obvious. Make a another list with two columns: all of the courses in the list, Crs, and the department to which each course belongs from list, Dpt. This list could be called CoursesGivenByDepartments, or GivenBy for short. The course in GivenBy is identified by the column, cNumber, which is the same column as in list, Crs. The department in GivenBy is identified by the column, dCode, which is the same column as in list, Dpt. Both, cNumber and dCode, in GivenBy are called foreign keys. The two foreign keys come from (reference) the columns of the same name in lists Crs and Dpt.

GivenBy

cNumber	dCode
COMP 228	1
COMP 229	1
PHYS 245	2
PHYS 253	2

Since a course belongs to only one department, cNumber of each course appears only once in list GivenBy, just as in list Crs. So instead of making the extra list GivenBy, the column, dCode, can be added to the list Crs. With this simple addition to the list of courses, courses are now related to departments. The column, dCode, in Crs (revised) is called a foreign key. The single foreign key comes from (references) the column of the same name in the list, Dpt.
So there are two methods to record the relationship between courses and departments:

(a) create an extra list, GivenBy, to record which department gives each course
(b) add an extra column to list, Crs, to record which department gives each course
It is desirable to minimize the number of lists and so for this reason the second method, (b), would be used.

Crs (revised)

dCode	cNumber	cName	credits
1	COMP 228	System Hardware	3
1	COMP 229	System Software	3
2	PHYS 245	Mechanics	3
2	PHYS 253	Electricity & Magnetism	3

Two rules pertain to the list, GivenBy, or the list, Crs (revised). Both rules are a consequence of one restriction which is based entirely on common sense: a course cannot be given by a department that does not exist. So a course cannot be added to the course list with a department code for a department that does not exist. Secondly, a department cannot be removed from the department list if there is a course that has the code of that department, that is, if one or more courses belong to that department. These restrictions are enforced by the declaration of a foreign key in a database.

4.6 Sample Database

4.6.1 ER diagram

The demonstrations of constraints in this chapter use a database that consists of two tables related by a many-to-one relationship. The ER (entity-relationship) diagram for the two tables and the relationship is shown in Figure 4.1. The two tables should be familiar to everyone. The table, Dpt, contains a tuple (row) for each department in a university. The table, Crs, contains a tuple (row) for each course that is given by a department. The relationship, GivenBy, expresses how the two tables are related: a course is given by a department. Crs and Dpt each have three attributes shown in the ER diagram. The attributes which are underlined are primary keys.

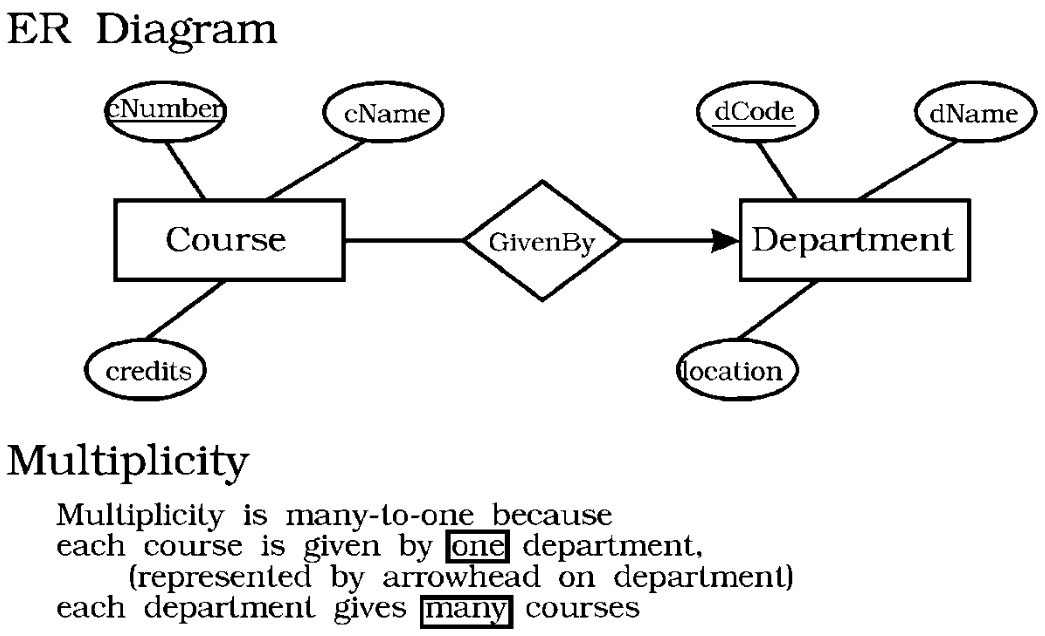
4.6.2 Multiplicity

The multiplicity of the relationship, many-to-one, is indicated by the single arrow in the ER diagram that points to table, Dpt. This notation conveys the meaning that is described in the Figure 4.1 below the ER diagram. Multiplicity of relationships is covered in the next chapter.

4.6.3 Schema

The ER diagram can be converted to either two or three tables, as illustrated in the last section. Usually the ER diagram is converted to two tables which is done here. The two tables have the following schemas. The attributes which are underlined are the primary keys. The attribute dCode in Crs is a foreign key which references the primary key in Dpt, indicated by (fk ⊞ Dpt).

```
Crs(cNumber, dCode(fk → Dpt), cName, credits)
Dpt(dCode, dName, location)
```



Multiplicity

Multiplicity is many-to-one because
each course is given by one department,
(represented by arrowhead on department)
each department gives many courses

Figure 4.1 Sample Database

The foreign key attributes are not shown on an ER diagram. The attributes of an entity shown on an ER diagram are only the properties of the entity. The foreign key of an entity is not a property of that entity but a property of another entity. In the sample database, dCode in entity, Crs, is a property of entity, Dpt. The foreign key attribute in Crs labels each course with the identification of the department to which the course belongs. Furthermore, it is possible to convert the ER diagram into three tables, as was done in the last section, and in that case the foreign keys would not belong to either of the entities.

4.6.4 Instance

An instance of a table is the data in a table at a particular time. An instance of each table, Crs and Dpt, is given in Table 4.3 and Table 4.4.

4.6.5 Create Database

The database is created without any constraints by the script, ct.sql. Then the database is populated by the script, in.sql. The name, ct, is an abbreviation for create table and name, in, stands for insert. The programming demonstrations start with the database without constraints and then add constraints one at a time. In this and the next chapter, MariaDB version of MySQL is used. It is compatible with MySQL.

4.6.5.1 Create tables

Use a script called, ct (for create tables), to create the two tables with the following schemas from Section 4.6.

```
Crs(cNumber, dCode(fk → Dpt), cName, credits)
Dpt(dCode, dName, location)
```

The tables are created without any constraints, except the domain of the attributes. The constraints will be added later.
Start the editor and type the script; once finished save it as ct.sql.

```
MariaDB [webdb]> system emacs;
```

Table 4.3 Crs Instance

Crs

cNumber	dCode	cName	credits
COMP228	10	System Hardware	3
COMP229	10	System Software	3
PHYS245	11	Mechanics	3
PHYS253	11	Electricity & Magnetism	3

Table 4.4 Dpt Instance

Dpt

dCode	dName	location
10	Computer Science	EV Building
11	Physics	Science Building

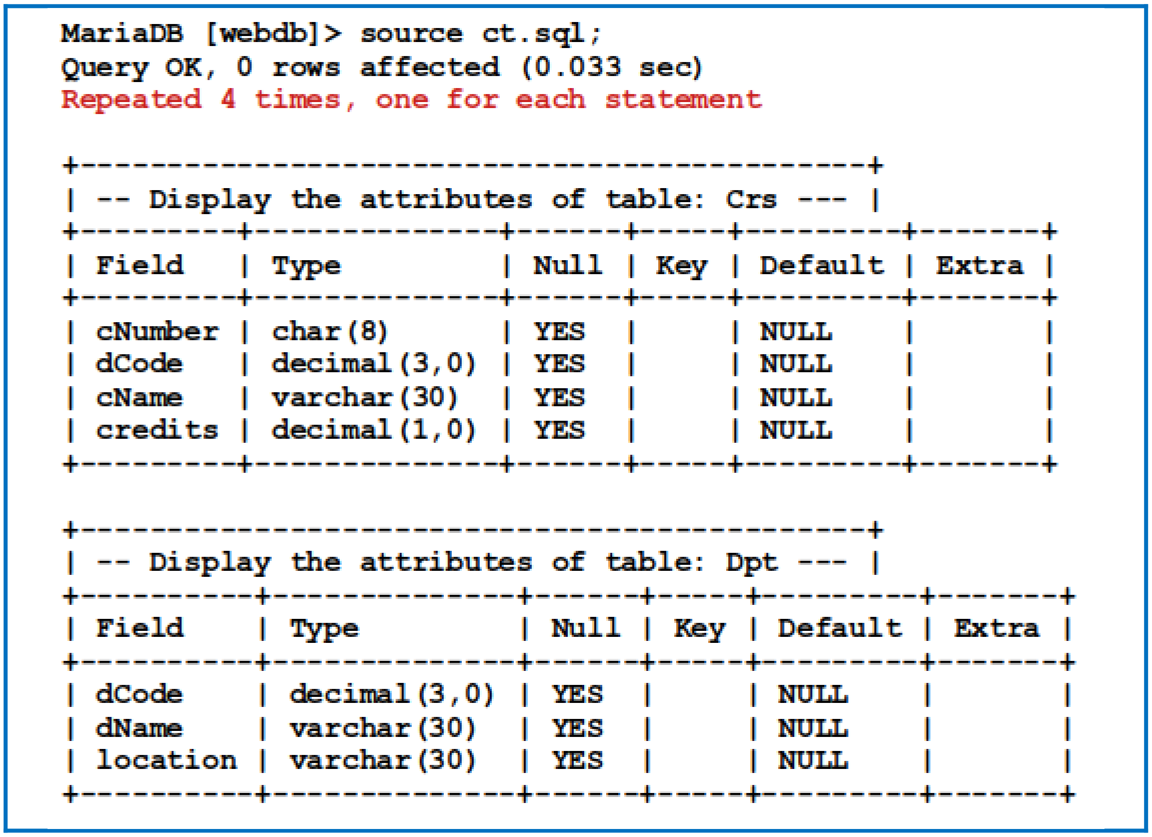
Script 4.1 Create Crs and Dpt: ct.sql
(Type the following script)

```
-- create many-to-one relationship
DROP TABLE IF EXISTS  Crs;      -- delete the table, Crs
DROP TABLE IF EXISTS  Dpt;      -- delete the table, Dpt
-- create the table, Crs
CREATE TABLE Crs (
     cNumber       CHAR(8),
     dCode         DECIMAL(3),
     cName         VARCHAR (30),
     credits       DECIMAL(1));
-- create the table, Dpt
CREATE TABLE Dpt (
     dCode         DECIMAL(3),
     dName         VARCHAR(30),
     location      VARCHAR(30));
select “” AS  “Description of attributes of table, Crs”;
DESC Crs;  -- display the attributes of the table, Crs
select “” AS  “Description of attributes of table, Dpt”;
DESC Dpt; -- display the attributes of the table, Dpt
```

(End of the script)

Save the script and terminate the editor. Control will return to MariaDB prompt.
Execute the script to create the tables.

```
MariaDB [webdb]> source ct.sql;
```



```
MariaDB [webdb]> source ct.sql;
Query OK, 0 rows affected (0.033 sec)
Repeated 4 times, one for each statement

+-----------------------------------------+
| -- Display the attributes of table: Crs --- |
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| cNumber | char(8)      | YES  |     | NULL    |       |
| dCode   | decimal(3,0) | YES  |     | NULL    |       |
| cName   | varchar(30)  | YES  |     | NULL    |       |
| credits | decimal(1,0) | YES  |     | NULL    |       |
+---------+--------------+------+-----+---------+-------+

+-----------------------------------------+
| -- Display the attributes of table: Dpt --- |
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| dCode    | decimal(3,0) | YES  |     | NULL    |       |
| dName    | varchar(30)  | YES  |     | NULL    |       |
| location | varchar(30)  | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
```

Figure 4.2 Script ct.sql, create tables with constraints: output

Figure 4.2 shows the output of script, ct.sql. When the script is executed for the first time, the two tables do not exist and there are two error messages advising that the tables do not exist. When the script is executed the second time and thereafter, two messages, Table dropped, will be displayed.
Note that the tables can be created in any order when there are no foreign keys declared. Crs, is created before table, Dpt, in script, ct.sql, but this will not be permitted when the foreign key in Crs is declared. This is demonstrated later.

4.6.5.2 Populate tables

Use a script called, in (for insert), to populate the tables with the instances in Tables 4.3 and 4.4.
Start the editor and type the script, once finished save it as in.sql..

```
MariaDB [webdb]>system emacs;
```

Script 4.2 Populate Crs and Dpt: in.sql
(Type the following script)

```
-- populate many-to-one relationship
```

```
DELETE FROM Crs;    -- delete the contents, if any, from Crs
DELETE FROM Dpt;    -- delete the contents, if any, from Dpt
-- populate the table, Crs
INSERT INTO Crs
     VALUES('COMP228', 10, 'System Hardware', 3);
INSERT INTO Crs
     VALUES('COMP229', 10, 'System Software', 3);
INSERT INTO Crs
     VALUES('PHYS245', 11, 'Mechanics', 3);
INSERT INTO Crs
     VALUES('PHYS253', 11, 'Electricity & Magnetism', 3);
-- populate the table, Dpt
INSERT INTO Dpt
     VALUES(10, 'Computer Science', 'Library Building');
INSERT INTO Dpt
     VALUES(11, 'Physics', 'Science Building');
SELECT “”  AS “Contents of Crs”;
SELECT * FROM Crs;  -- display the contents of the table, Crs
SELECT “”  AS “Contents of Dpt”;
SELECT * FROM Dpt;  -- display the contents of the table, Dpt
```

(End of the script)

Save the script and terminate the editor.
Execute the script to populate the tables.

```
MariaDB [webdb]> source in.sql;
```

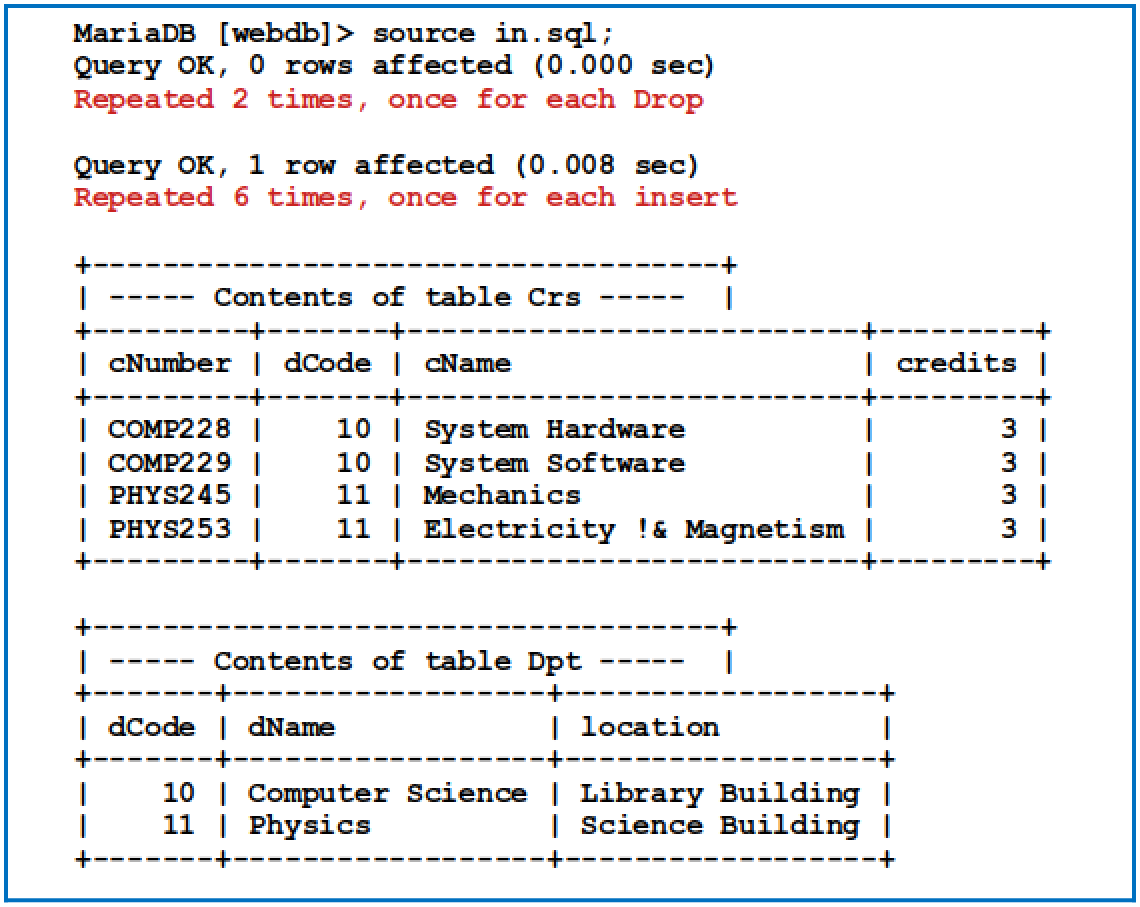
Figure 4.3 shows the output of script, in.sql. The output from the DB is edited in these figures to save space. Initailly the table just created by ct.sql did not have any data so deleting the contents would delete 0 rows..

The tables can be populated in any order when there are no foreign keys declared. Crs, is populated before table, Dpt, in script, in.sql, but this will not be permitted when the foreign key in Crs is declared. This is demonstrated later.

4.7 Semantics

Semantics is a branch of linguistics (the study of language) which deals with the study of meaning. However, in Computer Science the word, semantics, is a synonym for the word, meaning. The semantics (meaning, interpretation or representation) of the components of a database is the starting point for the design of a database.

There are two entities and one relationship in the database in Figure 4.1. The entities are Crs and Dpt which are related by the relationship, GivenBy. A course is given by one department. A department can give many courses. This relationship is called many-to-one.



```
MariaDB [webdb]> source in.sql;
Query OK, 0 rows affected (0.000 sec)
Repeated 2 times, once for each Drop

Query OK, 1 row affected (0.008 sec)
Repeated 6 times, once for each insert

+------------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+-------------------------+---------+
| cNumber | dCode | cName                   | credits |
+---------+-------+-------------------------+---------+
| COMP228 |    10 | System Hardware         |       3 |
| COMP229 |    10 | System Software         |       3 |
| PHYS245 |    11 | Mechanics               |       3 |
| PHYS253 |    11 | Electricity !& Magnetism |       3 |
+---------+-------+-------------------------+---------+

+------------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
```

Figure 4.3 Script in.sql, populate tables: output

The meaning of the two entities and their attributes leads to rules which are enforced by constraints. Some of the rules which are a consequence of the semantics are the following. The constraint which enforces the rule is given after the rule.

- Every course has a number which is different from all other courses.
  Enforced by entity integrity constraint, PRIMARY KEY
- Every department has a numerical code which is different from all other departments.
  Enforced by entity integrity constraint, PRIMARY KEY
- The name of every department is different from all other departments.
  Enforced by entity integrity constraint, UNIQUE
- The name of every course in each department is different from all other courses in the department.
  Enforced by entity integrity constraint, UNIQUE
- A course belongs to a department (a department does not belong to a course) and a course cannot belong to a department that does not exist.
  Enforced by referential integrity constraint, FOREIGN KEY, declared in course
- The credits of a course must have the value 3 or 4.
  Enforced by value constraint, CHECK
- The credits of a course cannot be missing.
  Enforced by missing data constraint, NOT NULL

- The credits of a course has a default value of 3, since most courses have a credit value of 3.
  Enforced by missing data constraint, DEFAULT

4.8 Constraints for Sample Database

The schema-based constraints are described in a previous section. The semantics of the sample database lead to the rules in the last section. The rules are applied by the constraints.

4.8.1 Entity integrity constraint - PRIMARY KEY

The primary key in each relation is underlined in the ER diagram in Figure 4.1 and in the schema. The attribute, cNumber, in relation, Crs, is different for every course and so cNumber can be declared as PRIMARY KEY. Also, the attribute, dCode, in relation, Dpt, is an identification number for each department which will be different for each department and so dCode can be declared as PRIMARY KEY. It is possible, in general, that more than one attribute or group of attributes in the same table can be the primary key, and in this case one of the them is chosen.

4.8.2 Entity integrity constraint - UNIQUE

The name of the department, dName in Dpt, should be different for every department. So dName is another choice for a primary key. When there are two or more choices for a primary key, the choices are called candidate keys. Since dCode is chosen as primary key, dName, must be declared as UNIQUE, since dName is different for every department.

The name of a course, cName in Crs, should be different for every course in a each department but could be the same for courses in two different departments. For example, there could be a course with the name, Introduction to Programming, in both Computer Science and Physics. So cName alone should not be declared UNIQUE, but cName together with dCode should be declared as UNIQUE.

4.8.3 Referential integrity constraint - FOREIGN KEY

Every course must belong to a department which exists, from the understanding that every course is the responsibility of some department and is offered by that department. So there cannot be a course in table, Crs, which has a dCode of a department that does not exist in table, Dpt. This constraint is enforced by a foreign key.

There are two consequences of the constraint imposed by the foreign key in table, Crs. A course cannot be added to table, Crs, which belongs to a department that does not exist. Secondly, a department cannot be removed from the table, Dpt, which has courses in table, Crs, since the courses would then belong to a department that no longer exists.

The attribute, dCode, in the schema of relation, Crs, is a foreign key which references the primary key, dCode, in relation, Dpt. It may be possible that a course in table, Crs, could have no value of dCode, that is, dCode is null and in this case the relationship between tuples in Dpt and Crs is optional for Crs (many of many-to-one). This could occur, for example, because a new course is proposed but not approved and so does not yet belong to a department. However, if every course must belong to a department (a course cannot have a dCode of null), the foreign key in relation, Crs, must be declared NOT NULL and then the relationship between tuples in Dpt and Crs is mandatory for Crs (many of many-to-one). The two possibilities, optional and mandatory, are called the participation of entities in a relationship which is covered in the next chapter.

Note that foreign keys, dCode in this case, are not shown on the ER diagram, Figure 4.1, but are in the schema. In a many-to-one relationship between two relations, the arrow in the ER diagram always points to the relation without the foreign key and so the foreign key is always in the relation without the arrowhead. So with this knowledge, from the ER diagram it can be deduced that there is a foreign key in the relation, Crs, which references the primary key, dCode, in relation, Dpt, and therefore the foreign key in relation, Crs, must be dCode.

The relationship between courses and departments is many-to-one because of the meaning of the entities, Crs and Dpt. A department administers a group of similar courses; a department can have many (more than one) courses and a course is in only one department. The relation between courses and departments would be one-to-one if each department could have only one course, which does not correspond to the usual meaning of course and department. However, if this was the case, the one-to-one relationship could be enforced simply by declaring the foreign key in table, Crs, to be UNIQUE.

4.8.4 Missing data constraint - NOT NULL

Each course should have a name and so cName in Crs should be declared NOT NULL.
If the location of a department is not required, then attribute, location, in Dpt can be null, which is the default.

4.8.5 Missing data constraint - DEFAULT

Most courses have a value of 3 credits, and so the default value of credits can be declared, DEFAULT 3.
When a new course is inserted in the database and the value of credits is not given, the value 3 will be stored. If credits in Crs must have a value, it must also be declared as NOT NULL. Otherwise the default value of 3 could be deleted after the course is inserted into the relation.

4.8.6 Domain constraint -CHECK

If courses can have a value of only 3 or 4 credits, the value of credits in Crs can be constrained to one of the two values by the constraint, CHECK (credits IN (3,4).

4.9 Sample Database with Different Table, Crs

This example illustrates composite primary keys, and strong and weak entities.

Consider the following change to the table, Crs, in Figure 4.1. A highly-paid university administrator decides that it would be simpler if courses are identified only by numbers and that the course numbers in each department start at 1. He (or she) agrees to prefix the course numbers with the letter, C, for course. So the course numbers in Computer Science would be C01, C02, . . . and the course numbers in physics would also be C01, C02, . . . The table, Crs, in Table 4.3 would become the table, Table 4.5.

The attributes of table, Crs, are exactly the same as before. There is only a semantic change (a change of meaning) of one of the attributes, cNumber. However, this leads to a significant change to the relationship between Crs and Dpt.

Table 4.5 Revised Table, Crs

cNumber	dCode	cName	credits
C01	10	System Hardware	3
C02	10	System Software	3
C01	11	Mechanics	3
C02	11	Electricity & Magnetism	3

4.9.1 Primary key

The course number, cNumber, in the revised table, Crs, cannot be the primary key because cNumber is not different for every course. Also, dCode, cannot be the primary key for the same reason. But the combination of cNumber and dCode are different for every course and so the primary key can be cNumber and dCode together. The primary key in this case is called a composite primary key which means that it consists of more than one attribute.

4.9.2 Foreign key

The foreign key for table, Crs, is still the same as for the table when cNumber alone was the primary key.

The attribute, dCode, is still the foreign key even though now dCode is part of the primary key of table, Crs.

4.9.3 Strong and weak entities

There is a fundamental difference between the two cases of the table, Crs. In Table 4.3, Crs, has the primary key, cNumber, which is an attribute that does not appear in table, Dpt. This means that the entity, Crs, can exist independently of the entity, Dpt. The course table can exist without the department table, because the attribute, dCode, in Crs can be null. Crs in this case is called a strong entity. Dpt, as well, is a strong entity.

In Table 4.5, Crs has a composite primary key which consists of two attributes, cNumber and dCode. One of the primary key attributes of Crs, dCode, is also the primary key of table, Dpt. Also, dCode, is a foreign key in Crs which references the primary key, dCode, in Dpt and so every value of dCode in Crs must also be in Dpt. Furthermore, dCode in Crs cannot be null because it is part of the primary key in Crs. This means that the existence of entity, Crs, depends on the existence of entity, Dpt. The table, Crs, cannot exist unless table, Dpt, exists because the attribute, dCode, in Crs must be a value of the primary key in Dpt. Crs in this case is called a weak entity. Also the relationship, GivenBy, is called weak.

One notation in ER diagrams to indicate weak entities and relationships is to enclose them by a double line instead of a single line. The weak entity, Crs, in the example above has the attribute, cNumber, as part of the primary key. The attribute (or attributes) of the weak entity that are part of the primary key is called the partial key. It is underlined by a dashed line (instead of a solid line) in the ER diagram. When a weak entity is related to a strong entity, the weak entity is sometimes called a dependent entity (dependent on the strong entity) and the strong entity is called an owner entity (owner of the weak entity).

4.10 DDL (Data Definition Language) Commands for Constraints

All of the SQL commands in MySQL are defined by syntax diagrams in Chapter 7 of the reference, SQL Reference, in the last section of this chapter. The syntax of commands for constraints in this section is derived from the syntax diagrams. The syntax of each command is a simplified version of the general syntax.

The meaning of the syntax diagrams are for the most part obvious. However, the description of syntax diagrams is given in Appendix A in the same reference, SQL Reference, and should be consulted, if necessary, to clarify the meaning of the diagrams.

The database created in this chapter is used in the examples. Names in uppercase are key words.

4.10.1 Create constraints: CREATE TABLE *tableName* . . .

4.10.1.1 Column constraint: constraint included in definition of column

Syntax:

CREATE TABLE *tableName* (*columnName columnDomain columnConstraint*, . . .)

Examples:

CREATE TABLE Dpt (dCode DECIMAL(3) PRIMARY KEY, . . .)

columnConstraint: PRIMARY KEY

CREATE TABLE Dpt (dName VARCHAR(30) UNIQUE, . . .)

columnConstraint: UNIQUE

CREATE TABLE Crs (credits DECIMAL(1) NOT NULL, . . .)

columnConstraint: NOT NULL

4.10.1.2 Table constraint: definition of constraint after definition of column

Syntax:

CREATE TABLE *tableName* (*tableConstraint*, . . .)

Examples:

The following two examples and the first example above are three different ways to declare a primary key. The best way is the last way, where the constraint is named.

CREATE TABLE Dpt (dCode DECIMAL(3), PRIMARY KEY (dCode), . . .)

tableConstraint: PRIMARY KEY (dCode)

CREATE TABLE Dpt (dCode DECIMAL(3), CONSTRAINT pkDpt-dCode PRIMARY KEY (dCode), . . .)

tableConstraint: CONSTRAINT pkDpt(dCode) PRIMARY KEY (dCode)
where pkDpt-dCode is chosen as the name of the constraint

4.10.2 Change constraints: ALTER TABLE *tableName*

4.10.2.1 ADD: add a new constraint

Syntax:

ALTER TABLE *tableName* ADD (*columnName columnDomain columnConstraint*)
ALTER TABLE *tableName* ADD (*tableConstraint*)

Examples:

The first example is for the case when the column does not exist. The second and third examples are for the case when the column does exist. The syntax is the same after the keyword, ADD, as for CREATE TABLE.

ALTER TABLE Dpt ADD (dCode DECIMAL(3) PRIMARY KEY)
columnConstraint: PRIMARY KEY
ALTER TABLE Dpt ADD (PRIMARY KEY (dCode))
tableConstraint: PRIMARY KEY (dCode)
ALTER TABLE Dpt ADD (CONSTRAINT pkDpt-dCode PRIMARY KEY (dCode))
tableConstraint: CONSTRAINT pkDpt(dCode) PRIMARY KEY (dCode)
where pkDpt-dCode is chosen as the name of the constraint

4.10.2.2 MODIFY: modify an existing constraint

Syntax:

ALTER TABLE *tableName* MODIFY *columnName columnConstraint*

Example:

ALTER TABLE Dpt MODIFY (dCode PRIMARY KEY)
columnConstraint: PRIMARY KEY

4.10.2.3 DROP: drop an existing constraint

Syntax:

ALTER TABLE *tableName* DROP *constraint* [CASCADE]

Example:

If the optional keyword, CASCADE, is used, all foreign keys which reference the primary key are also dropped. Without CASCADE only the primary key is dropped. In the database, a foreign key in table, Crs, references the primary key in table, Dpt.

ALTER TABLE Dpt DROP PRIMARY KEY [CASCADE]

4.10.2.4 ENABLE|DISABLE: enable or disable an existing constraint

Enable a constraint means to allow an existing constraint to be applied. Disable a constraint means to disallow an existing constraint to be applied without dropping the constraint.

Syntax:

ALTER TABLE *tableName* ENABLE|DISABLE *constraint* [CASCADE]

Examples:

If the optional keyword, CASCADE, is used, all foreign keys which reference the primary key are also enabled or disabled. Without CASCADE only the primary key is enabled or disabled. In the database, a foreign key in table, Crs, references the primary key in table, Dpt.

ALTER TABLE Dpt ENABLE|DISABLE PRIMARY KEY [CASCADE]
ALTER TABLE Dpt ENABLE|DISABLE CONSTRAINT pkDpt(dCode) [CASCADE]

where pkDpt-dCode is the name of the constraint

4.11 Programming Demonstrations

The programming demonstrations of constraints use the sample database in Figure 4.1. The tables have been created without constraints using a script (SQL file) and populated using another script, which are both executed by sourcing the respective files at the Mariadb prompt. The constraints are added one at a time by changing the script to create the tables and then executing the script again, which deletes both tables and then creates them again. The script to populate the tables is also changed each time a constraint is added and then executed to test the constraints.

The first script to create the tables is given the name, ct.sql, and the first script to populate the tables is given the name, in.sql. The name, ct, is an abbreviation for create table and name, in, stands for insert. Each time a script is changed, it is given a new name so that all versions of the scripts are stored on the hard disk. The new names are letters, ct or in, followed by the last two or three digits of the section number.

4.11.1 **Entity Integrity Constraint**

The entity integrity constraint is one or more attributes declared as either a primary key or as unique. A composite primary key or unique declaration (a constraint of more than one attribute) can have a maximum of 16 attributes.

4.11.1.1 PRIMARY KEY

A primary key identifies a tuple (row) of a table and is different for every tuple (row). It is declared by keywords, PRIMARY KEY. A primary key is one or more attributes (columns) of a table. There can be only one primary key in each table. The attributes of a primary key are automatically not null, that is, they do not need to be declared not null with the NOT NULL constraint.

A common primary key which consists of one attribute is an identification number. For example, if the table is Person, with the four attributes, name, weight, height and gender, name is not a good choice for a primary key because two different persons can have the same name. So a fifth attribute can be created with the name, personID for example, and domain, number. Then each person can be given a different number using personID and so every tuple will have a different personID. For example, every person in Canada has a different 9-digit identification number called the SIN (social insurance number).

In the table, Crs, there is only one choice for the primary key, cNumber, since it is possible that two courses in different departments could have the same name, cName, and more than one course will have the same value of credits, credits.

In the table, Dpt, there are two choices for the primary key: dCode and dName. The attribute, dCode, is chosen as the primary key. So the attribute, dName, must be declared as unique. dName also should be declared as not null, if it cannot be missing from a tuple.

(a) Without constraint

Add a tuple to table, Crs, which contains the same cNumber as in a tuple already in the table. Also add a tuple to table, Dpt, which contains a duplicate value of dCode.
Edit the script, in, and call the changed script, in21.

```
MariaDB [webdb]>system emacs  in.sql;
```

Script 4.3 Insert two tuples which violate primary key: in21.sql
add:

```
INSERT INTO Crs
     VALUES('COMP228',10,'Introduction to Programming',3);
INSERT INTO Dpt
     VALUES(11, 'Chemistry', 'Science Building');
```

Save the script using the name, in21,SQL and terminate the editor.

In emacs the following keyboard commands can be used for file handling. The notation, Ctrl+x, means press Ctrl and x at the same time.

(a) To create a copy of the file, in.sql, with the name, in21.sql, type Ctrl+x Ctrl+w, and then type the new name, in21.sql, followed by Enter. (Corresponds to Save As . . . in Windows)
(b) To save a file, type Ctrl+x Ctrl+s. (Corresponds to Save in Windows)
(c) To exit emacs, type Ctrl+x Ctrl+c. (Corresponds to Close in Windows)

Execute the script to populate the tables.

```
MariaDB [webdb]> source in21.sql;
```

The value of cNumber in table, Crs, is the same for two different tuples. Also the value of dCode in table, Dpt, is duplicated. This can be prevented by using a primary key for each table.
Figure 4.4 shows the last part of the output of script, in21.sql, without the constraint.

(b) With constraint

Add the constraint, PRIMARY KEY, for cNumber in table, Crs, and the constraint, PRIMARY KEY, for dCode in table, Dpt. There are two ways to declare a primary key.

(a) When the primary key consists of one attribute, the primary key can be declared on the same line as the declaration of the attribute. The syntax is,
`cNumber  CHAR(8)` becomes `cNumber  CHAR(8) PRIMARY KEY`

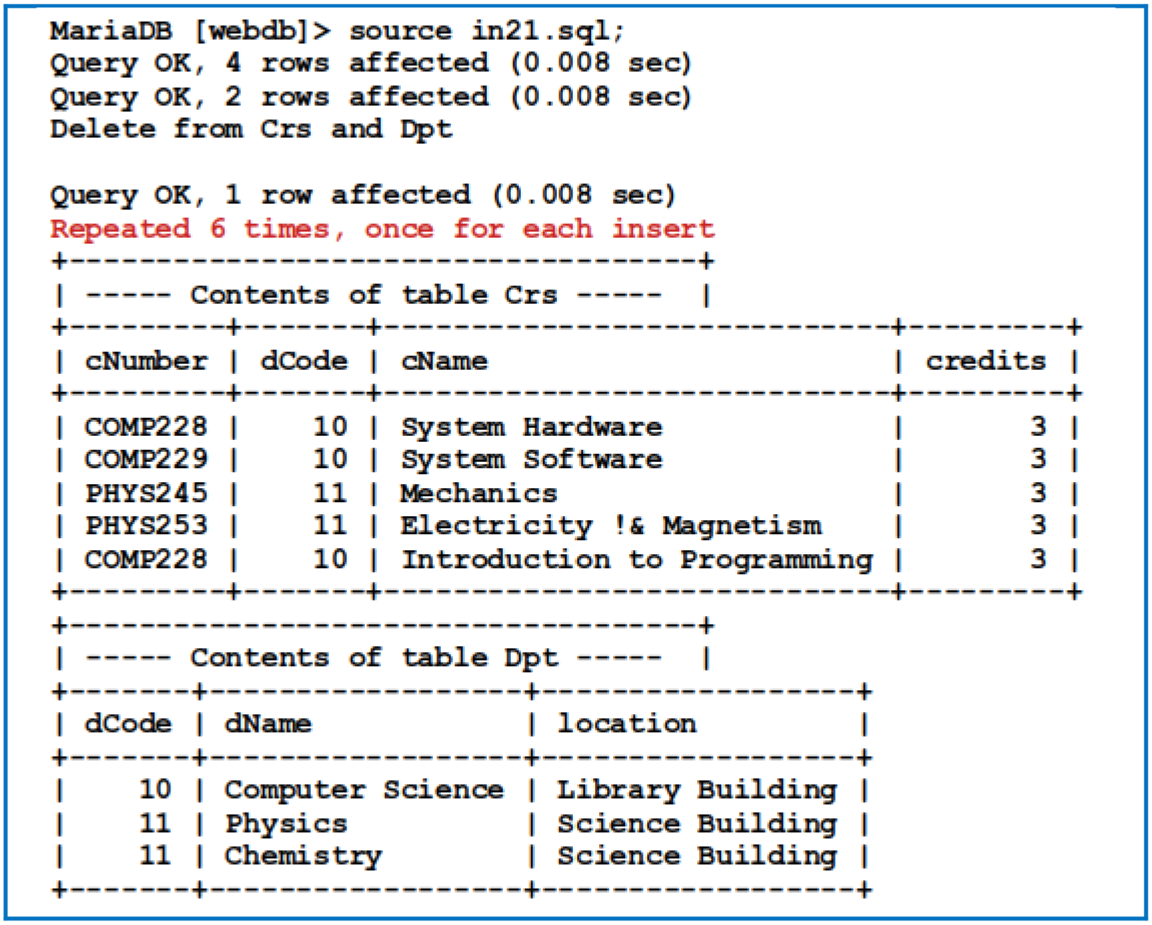
(b) When the primary key consists of more than one attribute, it must be declared on a separate line. A primary key that consists of only one attribute can also be declared on a separate line. The syntax is,

```
PRIMARY KEY (cNumber)
```

The last method will always be used

Edit the script, ct.sql, and call the changed script, ct21.sql.

```
MariaDB [webdb]>system emacs  ct.sql;
```



```
MariaDB [webdb]> source in21.sql;
Query OK, 4 rows affected (0.008 sec)
Query OK, 2 rows affected (0.008 sec)
Delete from Crs and Dpt

Query OK, 1 row affected (0.008 sec)
Repeated 6 times, once for each insert
+-----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+-----------------------------+---------+
| cNumber | dCode | cName                       | credits |
+---------+-------+-----------------------------+---------+
| COMP228 |    10 | System Hardware             |       3 |
| COMP229 |    10 | System Software             |       3 |
| PHYS245 |    11 | Mechanics                   |       3 |
| PHYS253 |    11 | Electricity !& Magnetism    |       3 |
| COMP228 |    10 | Introduction to Programming |       3 |
+---------+-------+-----------------------------+---------+
+-----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
|    11 | Chemistry        | Science Building |
+-------+------------------+------------------+
```

Figure 4.4 Script in21.sql, primary key: last part of output without constraint

Script 4.4 Add two primary keys: ct21.sql
add at the bottom of the CREATE TABLE command for Crs, inside the closing bracket:

```
   PRIMARY KEY (cNumber)
```

add at the bottom of the CREATE TABLE command for Dpt, inside the closing bracket:

```
   PRIMARY KEY (dCode)
```

Save the script using the name, ct21.sql, and terminate the editor.
Execute the script to create the updated tables.

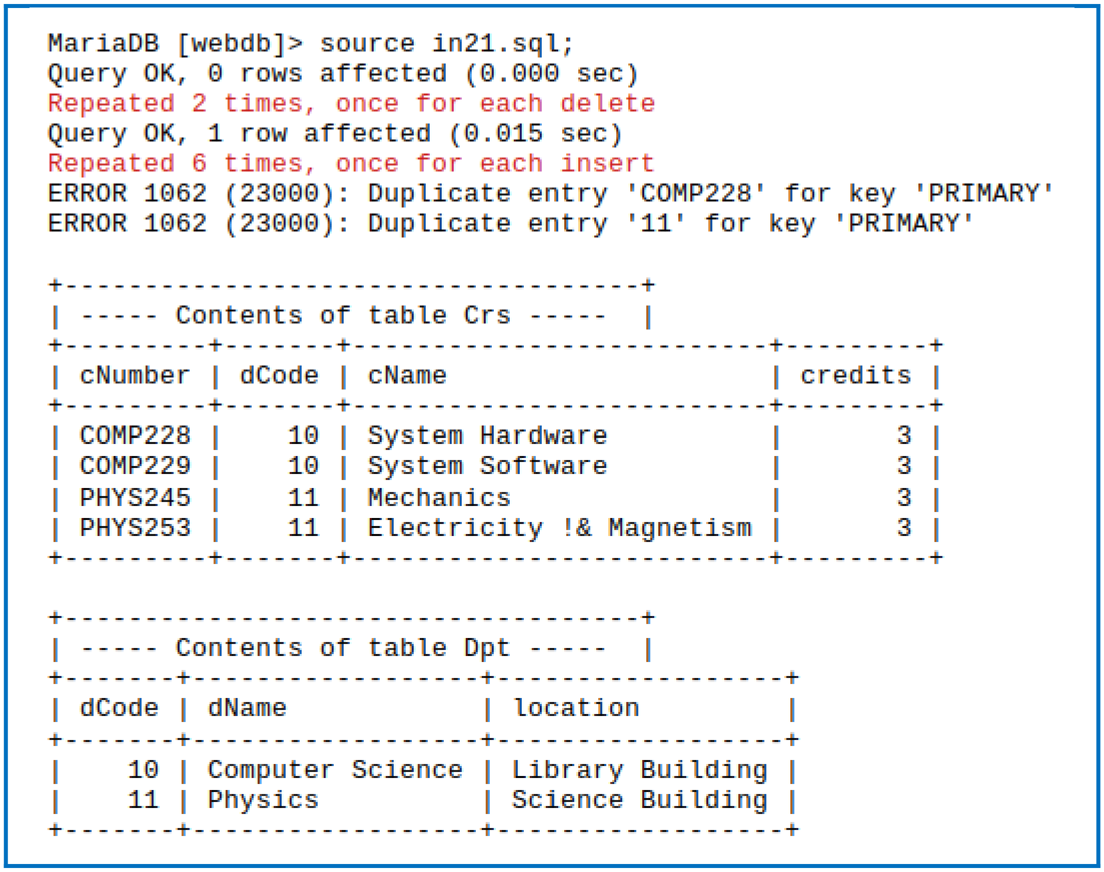
```
   MariaDB [webdb]> source ct21.sql;
```

Execute the script, in21.sql, again, to populate the tables.

```
   MariaDB [webdb]> source in21.sql;
```

An error message is displayed for both tables, Crs and Dpt: unique constraint violated. The two tuples which violate the primary key constraint were not inserted into the tables.

Figure 4.5 shows the last part of the output of script, in21.sql, with the constraint.



```
MariaDB [webdb]> source in21.sql;
Query OK, 0 rows affected (0.000 sec)
Repeated 2 times, once for each delete
Query OK, 1 row affected (0.015 sec)
Repeated 6 times, once for each insert
ERROR 1062 (23000): Duplicate entry 'COMP228' for key 'PRIMARY'
ERROR 1062 (23000): Duplicate entry '11' for key 'PRIMARY'

+-----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+------------------------+---------+
| cNumber | dCode | cName                  | credits |
+---------+-------+------------------------+---------+
| COMP228 |    10 | System Hardware        |       3 |
| COMP229 |    10 | System Software        |       3 |
| PHYS245 |    11 | Mechanics              |       3 |
| PHYS253 |    11 | Electricity !& Magnetism |       3 |
+---------+-------+------------------------+---------+

+-----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
```

Figure 4.5 Script in21.sql, primary key: last part of output with constraint

4.11.1.2 UNIQUE

The unique constraint specifies that every value of an attribute (or combination of values of more than one attribute) in a tuple is different, that is, there can be no duplicates of the attribute or (attributes). It is declared by keyword, UNIQUE. If there is more than one candidate key (choice for primary key) in a table, one key is chosen as the primary key and the alternate keys are declared to be UNIQUE. Every attribute that is declared as UNIQUE can be null unless it is also declared as NOT NULL. In contrast, an attribute declared to be a primary key (or part of a primary key) is automatically NOT NULL.

More than one UNIQUE constraint can be used in the same table. But, only one PRIMARY KEY constraint can be used in the same table. The use of the UNIQUE constraint together with the NOT NULL constraint is equivalent to the use of the PRIMARY KEY constraint.

4.11.1.3 FOREIGN KEY . . . ON DELETE

The clause ON DELETE in the declaration of a table specifies the action to be taken for foreign key violations caused by delete operation of a tuple involved. Tuples can be deleted in either the child table (table with foreign key) or parent table (table referenced by the foreign key). In the child table tuples can be deleted without restriction. However, in the parent table a tuple which is related to one or more tuples in the child table cannot be deleted, unless an ON DELETE option is specified in the declaration of the foreign key.

An operation that attempts to delete a value of the primary key in the parent table that has matching tuples in the child table will result in one of three actions.
(1) Default: The delete operation is rejected; the tuple with the primary key is not deleted. This is the default, when ON DELETE is not specified.
(2) ON DELETE CASCADE: The tuple with the primary key in the parent table is deleted and also all related tuples in the child table are deleted.
(3) ON DELETE SET NULL: The tuple with the primary key is deleted in the parent table and also the foreign key of all related tuples in the child table is set to null. If the foreign key is declared, NOT NULL, the delete operation is rejected; the tuple with the primary key is not deleted.

(a) Without constraint
In the table, Dpt, the name of a department, dName, must be different for each department. Add a tuple to table, Dpt, with a name which is already in the table.

Also, in the table, Crs, the name of a course in each department must be different, but the same name for a course can be used in different departments. Add a tuple to table, Crs, with a course name which is already in the same department. Edit the script, in.sql, and call the changed script, in22.sql.

```
MariaDB [webdb]>system emacs in.sql;
```

Script 4.5 Insert tuples which violates unique key: in22.sql
add:

```
INSERT INTO Dpt
     VALUES(12, 'Physics', 'Science Building');
INSERT INTO Crs
     VALUES('COMP230', 10, 'System Software', 3);
```

Save the script using the name, in22,sql and terminate the editor.

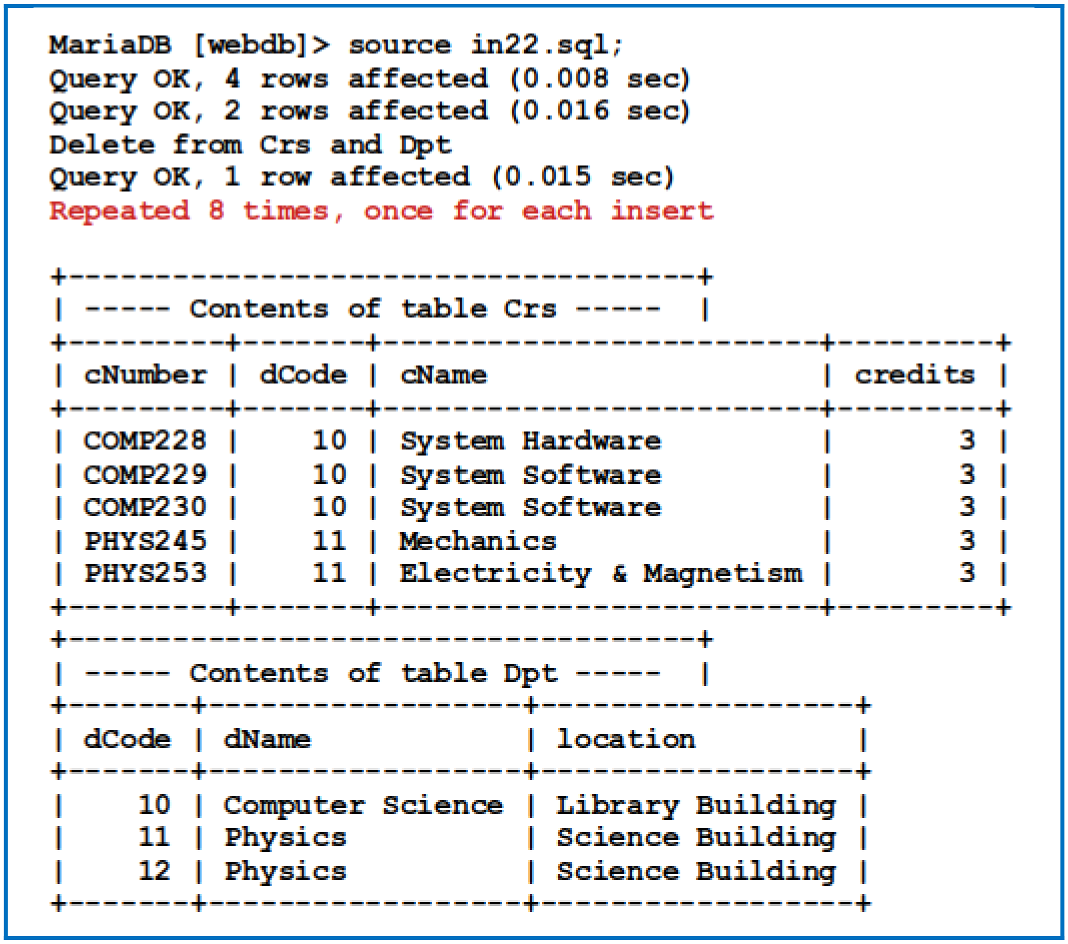
Execute the script to populate the tables:

```
MariaDB [webdb]> source in22.sql;
```

The name of two different departments is the same and the name of two different courses in the same department is the same. This can be prevented by using the unique constraint.

Figure 4.6 shows the last part of the output of script, in22.sql, without the constraint.

(b) With constraint
Edit the script, ct.sql, and call the changed script, ct22.sql.



```
MariaDB [webdb]> source in22.sql;
Query OK, 4 rows affected (0.008 sec)
Query OK, 2 rows affected (0.016 sec)
Delete from Crs and Dpt
Query OK, 1 row affected (0.015 sec)
Repeated 8 times, once for each insert

+-----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+------------------------+---------+
| cNumber | dCode | cName                  | credits |
+---------+-------+------------------------+---------+
| COMP228 |    10 | System Hardware        |       3 |
| COMP229 |    10 | System Software        |       3 |
| COMP230 |    10 | System Software        |       3 |
| PHYS245 |    11 | Mechanics              |       3 |
| PHYS253 |    11 | Electricity & Magnetism |       3 |
+---------+-------+------------------------+---------+
+-----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
|    12 | Physics          | Science Building |
+-------+------------------+------------------+
```

Figure 4.6 Script in22.sql, unique: last part of output without constraint

Script 4.6 Add unique constraint: ct22.sql

```
MariaDB [webdb]>system emacs  ct.sql;
```

change:

```
dName      VARCHAR(30)
```

to:

```
dName      VARCHAR(30) UNIQUE
```

add at the bottom of the CREATE TABLE command for Crs, inside the closing bracket:

```
UNIQUE (cName, dCode)
```

Save the script using the name, ct22.sql, and terminate the editor.

Execute the script to create the tables.

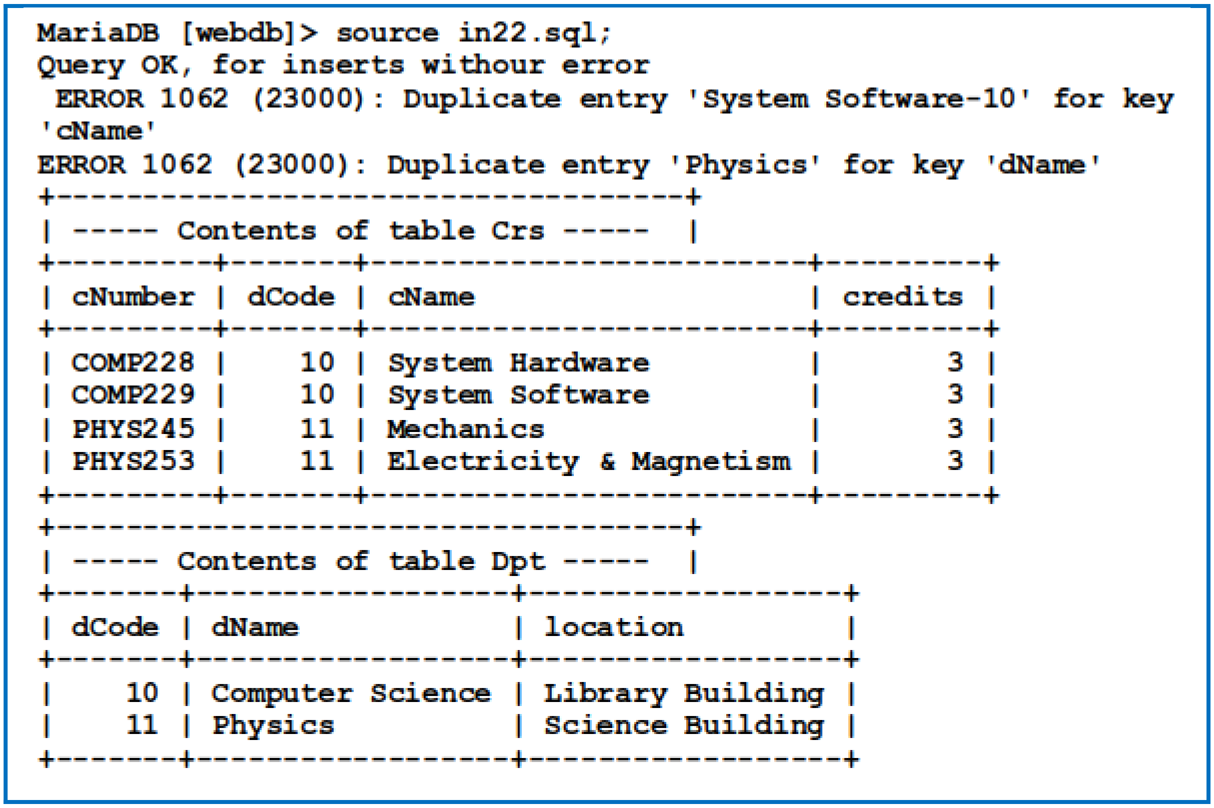
```
MariaDB [webdb]> source ct22.sql;
```

Execute the script, in22, again, to populate the tables.

```
MariaDB [webdb]> source in22.sql;
```

Two error messages are displayed: unique constraint is violated. The two tuples which violate the unique constraint were not inserted.

Figure 4.7 shows the last part of the output of script, in22.sql, with the constraint.



```
MariaDB [webdb]> source in22.sql;
Query OK, for inserts withour error
 ERROR 1062 (23000): Duplicate entry 'System Software-10' for key
'cName'
ERROR 1062 (23000): Duplicate entry 'Physics' for key 'dName'
+-------------------------------------+
| ----- Contents of table Crs -----   |
+---------+-------+-------------------------+---------+
| cNumber | dCode | cName                   | credits |
+---------+-------+-------------------------+---------+
| COMP228 |    10 | System Hardware         |       3 |
| COMP229 |    10 | System Software         |       3 |
| PHYS245 |    11 | Mechanics               |       3 |
| PHYS253 |    11 | Electricity & Magnetism |       3 |
+---------+-------+-------------------------+---------+
+-------------------------------------+
| ----- Contents of table Dpt -----   |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
```

Figure 4.7 Script in22.sql, unique: last part of output with constraint

4.11.2 Referential Integrity Constraint

A referential integrity constraint is specified by a foreign key, which can be one or more attributes of a table. It is declared by FOREIGN KEY.

A foreign key relates (or links or connects) two tables. The foreign key in one table is related to a primary key in the other table. Suppose the primary key is in table, Tablepri, and the foreign key is in table, Tablefor. The declaration of the foreign key is in Tablefor and the declaration specifies (a) the name of the attribute (or attributes) in Tablefor, (b) the name of the related table, Tablepri, and (c) the name of the attribute (or attributes) in Tablepri. The attribute (or attributes) in Tablepri is the primary key. It is not necessary to specify the name of the primary key if it is the same as the name of the foreign key.

The terminology, parent table and child table, is often used for two tables which are related by a foreign key. In the case where tables, Tablepri and Tablefor, are related and the foreign key is in Tablefor, Tablepri is the parent table and Tablefor is the child table.

The parent table must be created before the child table is created, which is easy to remember since this is the same order of creation when the parent and child are people. Also tuples in the parent table must be inserted before related tuples in the child table are inserted

4.11.2.1 FOREIGN KEY

A foreign key is one or more attributes in one table which reference the primary key in another table. The primary key must be the same attributes or attributes as the foreign key and so the two tables have the same attribute or attributes in common. The foreign key relates a tuple (row) in the table with the foreign key to a tuple (row) in the other table. In the database in Figure 4.1, the two tables, Crs and Dpt, are related by one attribute, dCode. The attribute, dCode, is the primary key in Dpt and will become the foreign key in Crs, when it is declared.

(a) Without constraint

Add a tuple to table, Crs, which contains a department code, dCode, which is not in the table, Dpt.

Edit the script, in, and call the changed script, in31.sql.

```
MariaDB [webdb]>system emacs  in.sql
```

Script 4.7 Insert tuple which violates foreign key: in31.sql

add:

```
INSERT INTO Crs
      VALUES('CHEM217', 12, 'Analytical Chemistry', 3);
```

Save the script using the name, in31.sql, and terminate the editor.

Execute the script to populate the tables.

```
MariaDB [webdb]> source in31.sql
```

A tuple has been entered in table, Crs, for a department that does not exist in table, Dpt. This is undesirable. This can be prevented by a foreign key. Figure 4.8 shows the last part of the output of script, in31.sql, without the constraint.

(b) With constraint

Add the declaration of the foreign key, dCode, in table, Crs. The attribute, dCode, appears in both tables and is the means by which tuples in the two tables are connected.

Edit the script, ct21,sql and call the changed script, ct31.sql Script, ct21,sql contains declarations for the primary keys of tables, Crs and Dpt.

```
MariaDB [webdb]>system emacs  ct21.sql
```

Script 4.8 Add foreign key: ct31.sql

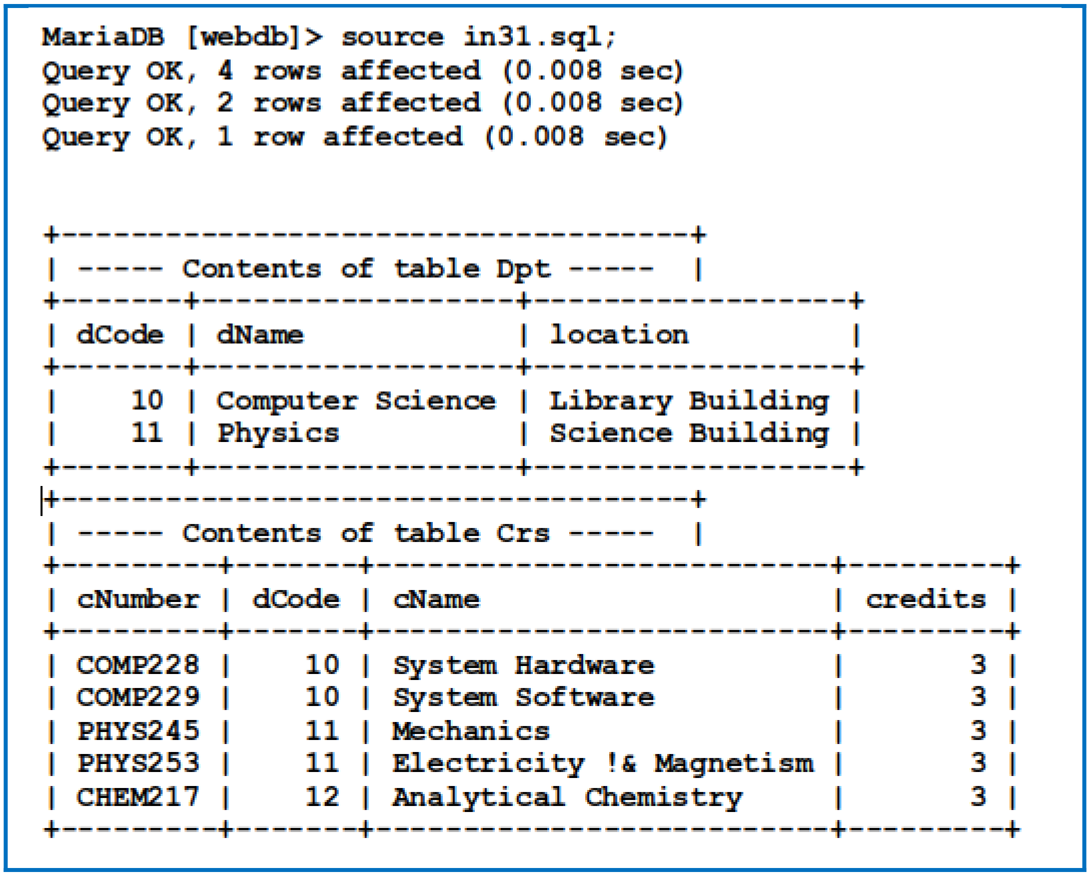
add at the bottom of the CREATE TABLE command for Crs, inside the closing bracket:

```
FOREIGN KEY (dCode) REFERENCES Dpt
```

Save the script using the name, ct31,sql and terminate the editor.

Execute the script to create the tables: MariaDB [webdb]> source ct31.sql

The foreign key references table, Dpt, which does not exist, because it is dropped at the beginning and then created after table, Crs. Dpt is the parent table of Crs, the child table. The parent table must be created first. Edit script, ct31.sql, by creating the table, Dpt, before table, Crs.



```
MariaDB [webdb]> source in31.sql;
Query OK, 4 rows affected (0.008 sec)
Query OK, 2 rows affected (0.008 sec)
Query OK, 1 row affected (0.008 sec)


+-------------------------------------+
| ----- Contents of table Dpt -----   |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
+-------------------------------------+
| ----- Contents of table Crs -----   |
+---------+-------+--------------------------+---------+
| cNumber | dCode | cName                    | credits |
+---------+-------+--------------------------+---------+
| COMP228 |    10 | System Hardware          |       3 |
| COMP229 |    10 | System Software          |       3 |
| PHYS245 |    11 | Mechanics                |       3 |
| PHYS253 |    11 | Electricity !& Magnetism |       3 |
| CHEM217 |    12 | Analytical Chemistry     |       3 |
+---------+-------+--------------------------+---------+
```

Figure 4.8 Script in31.sql, foreign key: last part of output without constraint

.

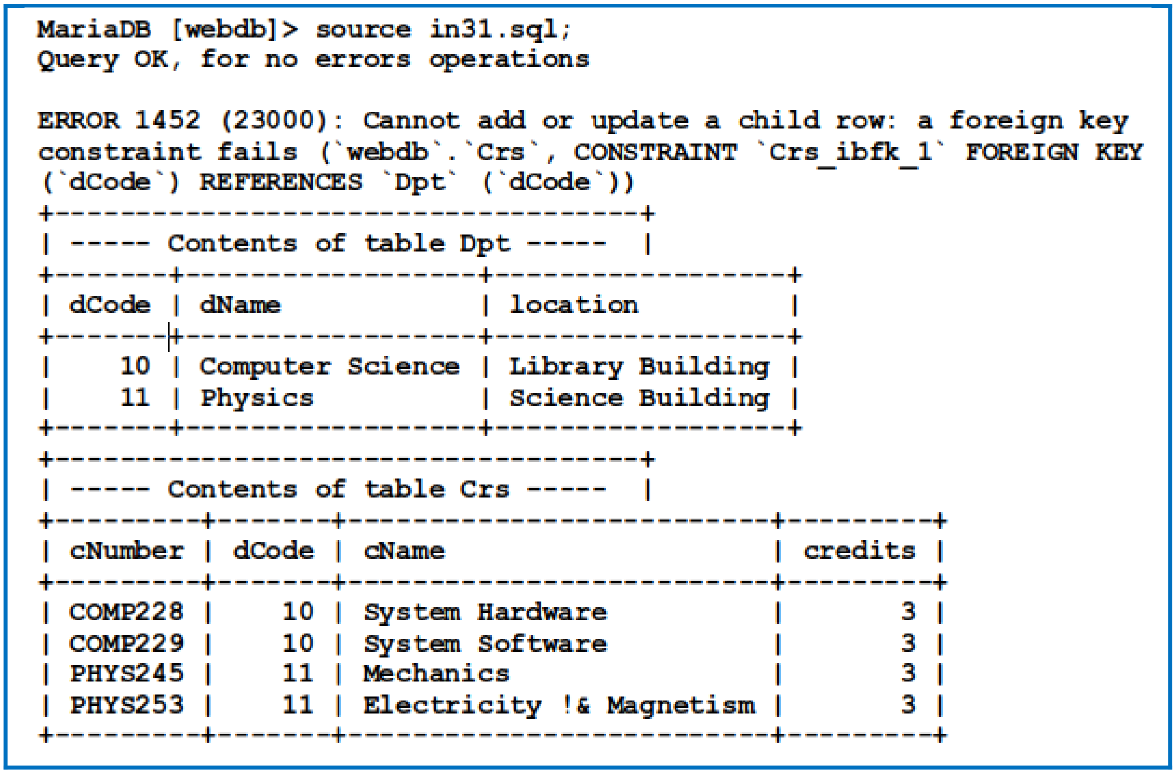
`MariaDB [webdb]>system emacs  ct31.sql` (put the create table command for Dpt before the create table command for Crs)

Save the script with the name, ct31r.sql and terminate the editor.
Execute the script to create the tables: `MariaDB [webdb]> source ct31r.sql`
Execute the script, in31.sql, again, to populate the tables:
`MariaDB [webdb]> source in31.sql`

The data in table, Crs, is inserted before the data in table, Dpt, and so none of the department codes, dCode, exist which are referenced by table, Crs. None of the data is inserted into Crs. Edit script, in31.sql, by inserting data into table, Dpt, before table, Crs.

`MariaDB [webdb]>system emacs  in31.sql` (put the insert commands for Dpt before the insert commands for Crs)

Save the script with the same name, in31.sql, and terminate the editor.



```
MariaDB [webdb]> source in31.sql;
Query OK, for no errors operations

ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (`webdb`.`Crs`, CONSTRAINT `Crs_ibfk_1` FOREIGN KEY
(`dCode`) REFERENCES `Dpt` (`dCode`))
+-----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
+-----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+-------------------------+---------+
| cNumber | dCode | cName                   | credits |
+---------+-------+-------------------------+---------+
| COMP228 |    10 | System Hardware         |       3 |
| COMP229 |    10 | System Software         |       3 |
| PHYS245 |    11 | Mechanics               |       3 |
| PHYS253 |    11 | Electricity !& Magnetism |       3 |
+---------+-------+-------------------------+---------+
```

Figure 4.9 Script in31.sql, foreign key: last part of output with constraint

Execute the script to populate the tables: MariaDB [webdb]> source in31.sql;
Now only one tuple is not inserted into table, Crs, the one for dCode = 12, which does not exist in table, Dpt. This was prevented by the foreign key.

Figure 4.9 shows the last part of the output of script, in31.sql, with the constraint.

4.11.2.1.1 Default

The option, ON DELETE, is not specified.
The script, ct31.sql, contains the declaration of the foreign key for table, Crs.
Execute the script, ct31sql, to create the tables.

```
   MariaDB [webdb]> source ct31.sql;
```

Remove the tuple from table, Crs, which contains the department code, dCode, which is not in the table, Dpt.
Edit the script, in31, and call the changed script, in32.

```
   MariaDB [webdb]>system emacs  in31.sql;
```

Script 4.9 Remove tuple: in32.sql
remove:

```
     INSERT INTO Crs
          VALUES('CHEM217',12,'Analytical Chemistry',3);
```

Save the script using the name, in32, and terminate the editor.

Execute the script, in32sql, to populate the tables.

```
      MariaDB [webdb]> source in32.sql;
```

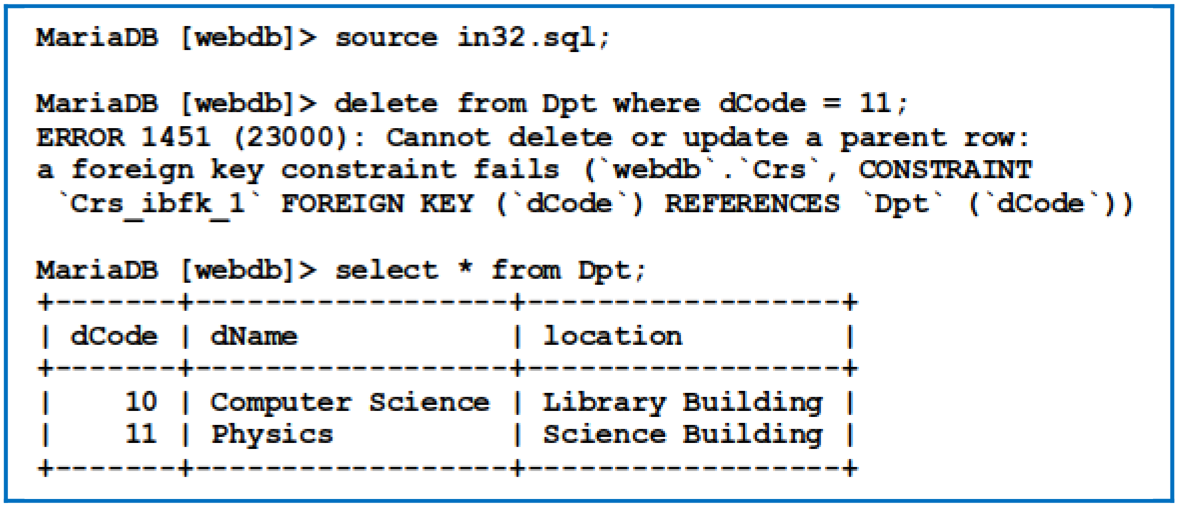
Attempt to delete the tuple in table, Dpt, with primary key, dCode = 11. Two tuples in the child table, Crs, are related to this primary key and so the deletion will be disallowed.

```
      MariaDB [webdb]> DELETE FROM Dpt
            WHERE dCode = 11;
```

An error message is displayed: integrity constraint violated - child record found. (Record is another name for tuple.)

Display the table, Dpt, to verify that no tuple was deleted.

```
      MariaDB [webdb]> SELECT * FROM Dpt;
```



```
MariaDB [webdb]> source in32.sql;

MariaDB [webdb]> delete from Dpt where dCode = 11;
ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails (`webdb`.`Crs`, CONSTRAINT
 `Crs_ibfk_1` FOREIGN KEY (`dCode`) REFERENCES `Dpt` (`dCode`))

MariaDB [webdb]> select * from Dpt;
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
```

Figure 4.10 Violation of foreign key constraint

Figure 4.10 shows the error message after the command to delete a row from table, Dpt, which is referenced by a row in table, Crs. Then the contents of the table, Dpt is displayed to show that no rows were deleted.

4.11.2.1.2 ON DELETE CASCADE

Edit the script, ct31, and call the changed script, ct322.sql.

```
      MariaDB [webdb]>system emacs  ct31.sql;
```

Script 4.10 Add foreign key option, on delete cascade: ct322.sql

change: `FOREIGN KEY (dCode)REFERENCES Dpt`

to: `FOREIGN KEY (dCode)REFERENCES Dpt ON DELETE CASCADE`

Save the script using the name, ct322.sql, and terminate the editor.

Execute the script, ct322, to create the tables.

```
      MariaDB [webdb]> source ct322.sql
```

Execute the script, in32.sql, to populate the tables.

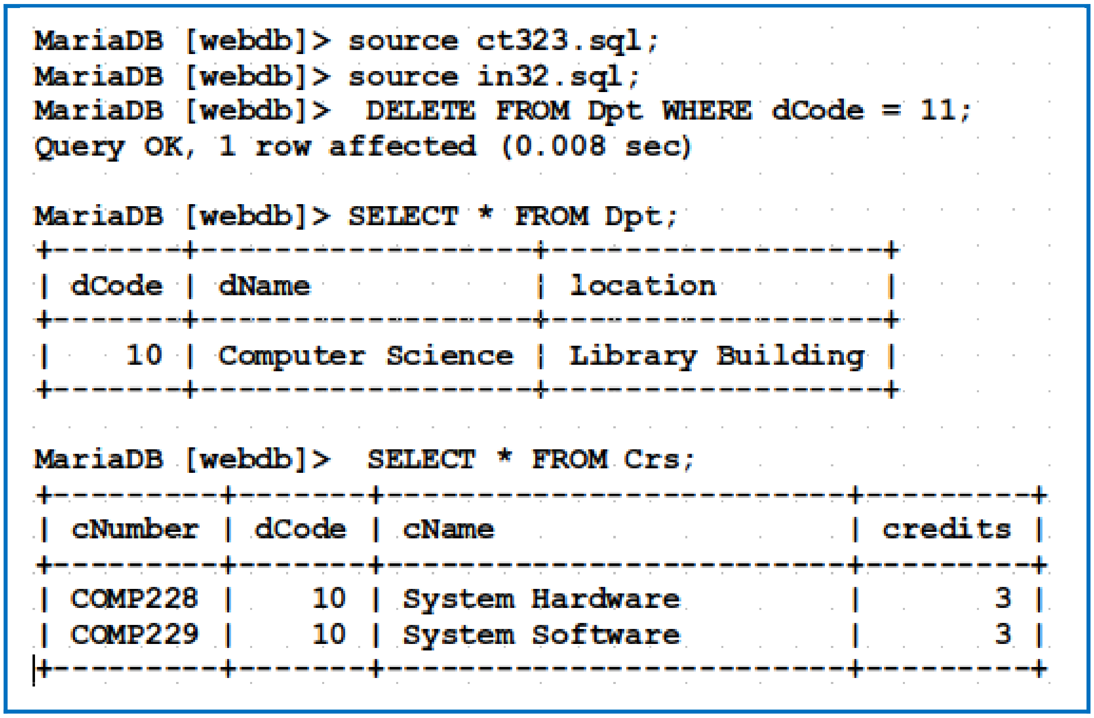
```
      MariaDB [webdb]> source in32.sql;
```

Delete the tuple in table, Dpt, with primary key, dCode = 11. Two tuples in the child table, Crs, are related to this primary key and so they will also be deleted.

```
MariaDB [webdb]> DELETE FROM Dpt
     WHERE dCode = 11;
```

Display the two tables to observe that both the tuple in Dpt and the two related tuples in Crs were deleted.

```
MariaDB [webdb]> SELECT * FROM Dpt;
MariaDB [webdb]> SELECT * FROM Crs;
```



```
MariaDB [webdb]> source ct323.sql;
MariaDB [webdb]> source in32.sql;
MariaDB [webdb]>  DELETE FROM Dpt WHERE dCode = 11;
Query OK, 1 row affected (0.008 sec)

MariaDB [webdb]> SELECT * FROM Dpt;
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
+-------+------------------+------------------+

MariaDB [webdb]>  SELECT * FROM Crs;
+---------+-------+------------------------+---------+
| cNumber | dCode | cName                  | credits |
+---------+-------+------------------------+---------+
| COMP228 |    10 | System Hardware        |       3 |
| COMP229 |    10 | System Software        |       3 |
+---------+-------+------------------------+---------+
```

Figure 4.11 Action of foreign key with ON DELETE CASCADE

Figure 4.11 shows the command to delete a row from table, Dpt. Then the contents of both tables are displayed to show the changes. The row in table, Dpt, was deleted and the two related rows in table, Crs, were also deleted – deletion is cascaded.

4.11.2.1.3 ON DELETE SET NULL

Edit the script, ct31.sql, and call the changed script, ct323.sql.

```
MariaDB [webdb]>system emacs  ct31.sql;
```

Script 4.11 Add foreign key option, on delete set null: ct323.sql

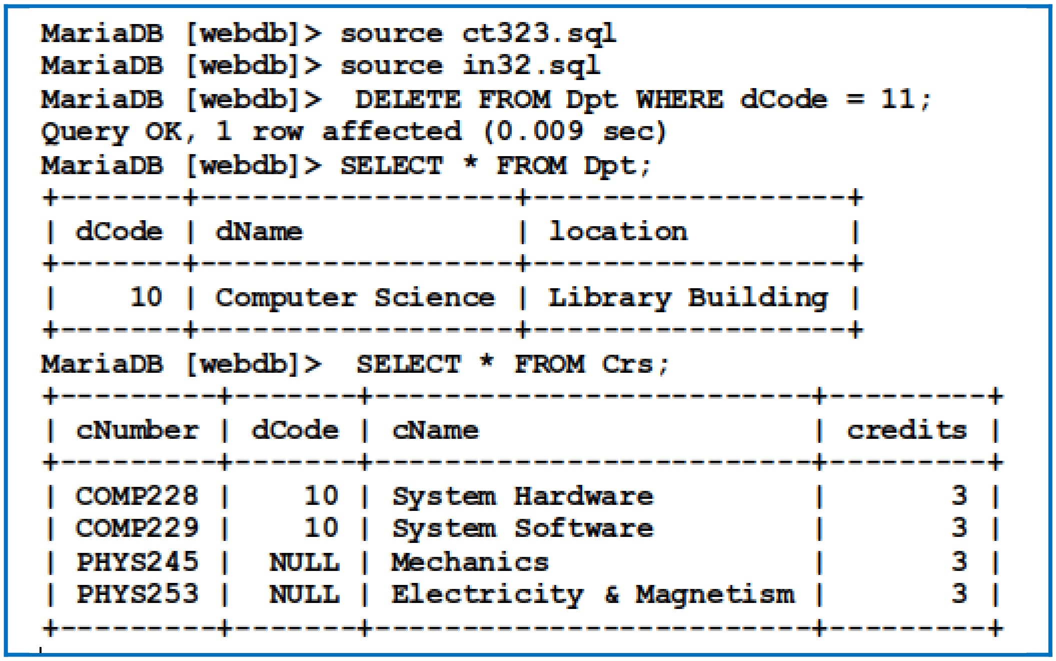
change: `FOREIGN KEY (dCode)REFERENCES Dpt`

to: `FOREIGN KEY (dCode)REFERENCES Dpt ON DELETE SET NULL`

Save the script using the name, ct323.sql, and terminate the editor.

Execute the script, ct323.sql, to create the tables.

```
MariaDB [webdb]> source ct323.sql;
```



```
MariaDB [webdb]> source ct323.sql
MariaDB [webdb]> source in32.sql
MariaDB [webdb]>  DELETE FROM Dpt WHERE dCode = 11;
Query OK, 1 row affected (0.009 sec)
MariaDB [webdb]> SELECT * FROM Dpt;
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
+-------+------------------+------------------+
MariaDB [webdb]>  SELECT * FROM Crs;
+---------+-------+-------------------------+---------+
| cNumber | dCode | cName                   | credits |
+---------+-------+-------------------------+---------+
| COMP228 |    10 | System Hardware         |       3 |
| COMP229 |    10 | System Software         |       3 |
| PHYS245 |  NULL | Mechanics               |       3 |
| PHYS253 |  NULL | Electricity & Magnetism |       3 |
+---------+-------+-------------------------+---------+
```

Figure 4.12 Action of foreign key with ON DELETE SET NULL

Execute the script, in32.sql, to populate the tables.

```
     MariaDB [webdb]> source in32.sql;
```

Delete the tuple in table, Dpt, with primary key, dCode = 11. Two tuples in the child table, Crs, are related to this primary key so their foreign keys will be set to null.

```
     MariaDB [webdb]> DELETE FROM Dpt
          WHERE dCode = 11;
```

Display the two tables to observe that the tuple in Dpt was deleted and the foreign key in the two related tuples in Crs was set to null.

```
     MariaDB [webdb]> SELECT * FROM Dpt;
     MariaDB [webdb]> SELECT * FROM Crs;
```

Figure 4.12 shows the command to delete a row from table, Dpt. Then the contents of both tables are displayed to show the changes. The row in table, Dpt, was deleted and in the two related rows in table, Crs, the foreign key was set to null – not cascaded.

4.11.2.1.4 FOREIGN KEY . . . ON UPDATE

FOREIGN KEY . . . ON UPDATE - specifies action for foreign key violations caused by insert or update operations. In Oracle there is no ON UPDATE option which corresponds to the ON DELETE option. So in the parent table the value of the primary key of a tuple which is related to one or more tuples in the child table cannot be changed.
The actions of cascade and set null for update can be implemented using triggers. For ON UPDATE CASCADE, an update operation would change the primary key value in the parent table and also change the related foreign key values in the child table. For ON UPDATE

SET NULL, the update operation would change the primary key value in the parent table and also set to null the related foreign key values in the child table, unless the foreign key is declared, NOT NULL.

4.11.3 **Missing Data Constraint**

4.11.3.1 NOT NULL

The constraint, NOT NULL, specifies that an attribute must have a value. Data cannot be missing from an attribute in any tuples of a table if the attribute is declared, NOT NULL.

(a) Without constraint
In the table, Crs, assume that there is a rule that there must be a value for credits for every course.

Add a tuple to table, Crs, with a missing value (null) for attribute, credits.
Edit the script, in32.sql, and call the changed script, in41.sql.

```
MariaDB [webdb]>system emacs  in32.sql;
```

Script 4.12 Insert tuple with null for credits: in41.sql
add:

```
INSERT INTO Crs
VALUES('COMP248',10,'Introduction to Programming', null);
```

Save the script using the name, in41.sql, and terminate the editor.

Execute the script to populate the tables.

```
MariaDB [webdb]> source in41.sql;
```

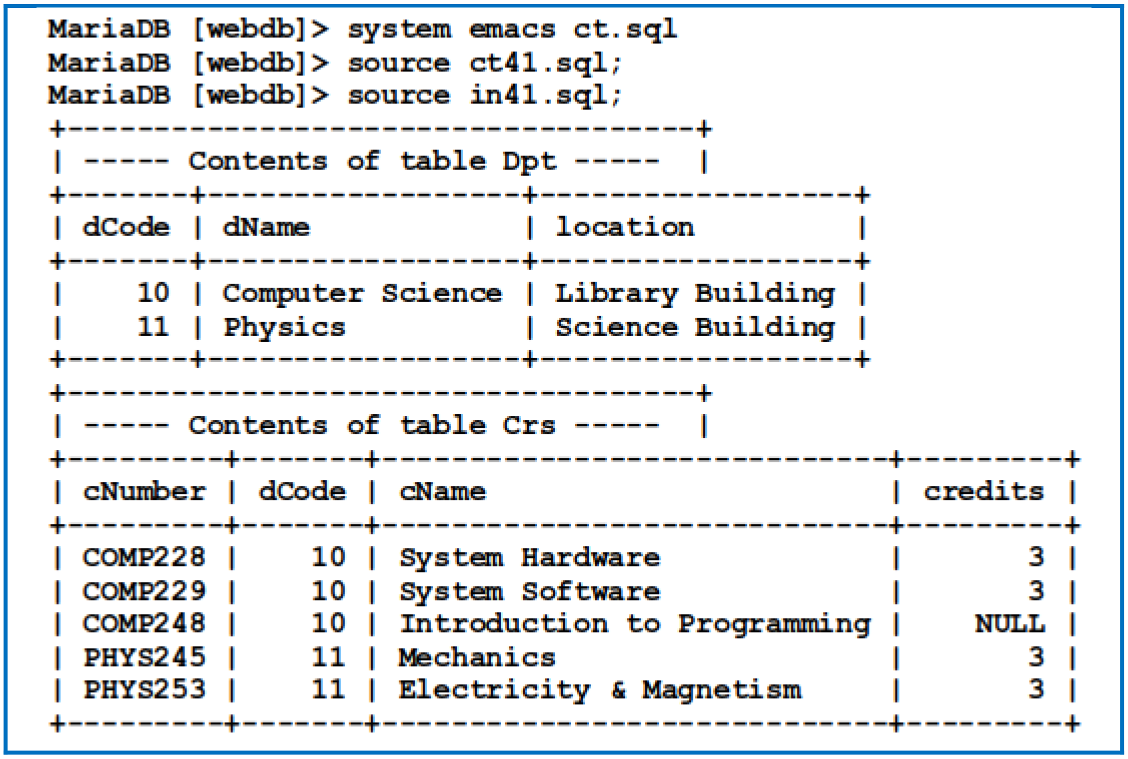
The value of the credits for the new course is missing. This can be prevented by the NOT NULL constraint.

Figure 4.13 shows the last part of the script, in41.sql, without the constraint

(b) With constraint
Change the constraint for attribute, credits, so that it cannot be null.

Edit the script, ct31.sql, and call the changed script, ct41.sql.

```
MariaDB [webdb]>system emacs  ct31.sql;
```



```
MariaDB [webdb]> system emacs ct.sql
MariaDB [webdb]> source ct41.sql;
MariaDB [webdb]> source in41.sql;
+------------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
+------------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+-----------------------------+---------+
| cNumber | dCode | cName                       | credits |
+---------+-------+-----------------------------+---------+
| COMP228 |    10 | System Hardware             |       3 |
| COMP229 |    10 | System Software             |       3 |
| COMP248 |    10 | Introduction to Programming |    NULL |
| PHYS245 |    11 | Mechanics                   |       3 |
| PHYS253 |    11 | Electricity & Magnetism     |       3 |
+---------+-------+-----------------------------+---------+
```

.Figure 4.13 Script in41.sql, not null: last part of output without constraint

Script 4.13 Add not null constraint: ct41.sql
change:

```
credits DECIMAL(1)
```

to:

```
credits DECIMAL(1) NOT NULL
```

Save the script using the name, ct41.sql, and terminate the editor.
Execute the script to create the tables.

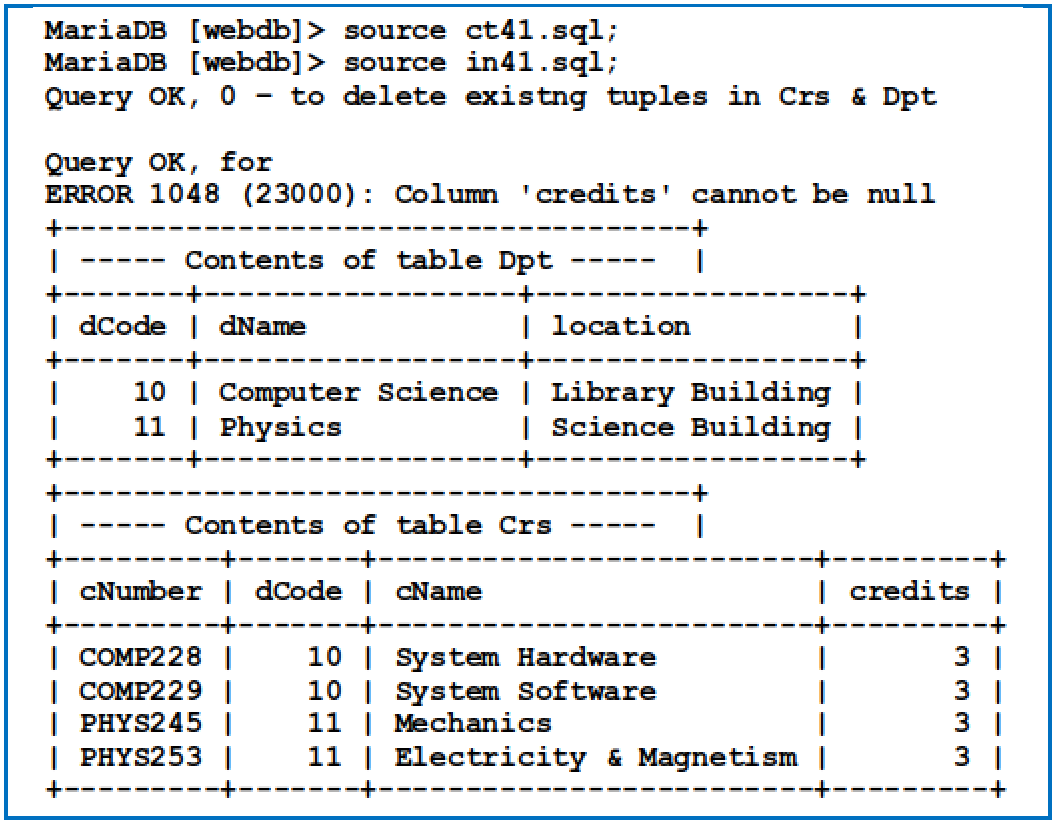
```
MariaDB [webdb]> source ct41.sql;
```

Execute the script, in41, again, to populate the tables.

```
MariaDB [webdb]> source in41.sql;
```

An error message is displayed: cannot insert null into course.credits. This was prevented by the NOT NULL constraint. The tuple which violates the constraint was not inserted into table, Crs.

Figure 4.14 shows the last part of the script, in41.sql, with the constraint.



```
MariaDB [webdb]> source ct41.sql;
MariaDB [webdb]> source in41.sql;
Query OK, 0 - to delete existng tuples in Crs & Dpt

Query OK, for
ERROR 1048 (23000): Column 'credits' cannot be null
+------------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
+------------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+-------------------------+---------+
| cNumber | dCode | cName                   | credits |
+---------+-------+-------------------------+---------+
| COMP228 |    10 | System Hardware         |       3 |
| COMP229 |    10 | System Software         |       3 |
| PHYS245 |    11 | Mechanics               |       3 |
| PHYS253 |    11 | Electricity & Magnetism |       3 |
+---------+-------+-------------------------+---------+
```

Figure 4.14 Script in41.sql, not null: last part of output with constraint

4.11.3.2 DEFAULT

The constraint, DEFAULT, specifies the value of an attribute if the value is not given by the INSERT command. If the INSERT command specifies any value, including null, that value is entered instead of the default value.

(a) Without constraint

For each course the number of credits must be specified.
Add a tuple to table, Crs, which does not include the value of credits.
Edit the script, in32.sql, and call the changed script, in42sql.

```
MariaDB [webdb]>system emacs in32.sql;
```

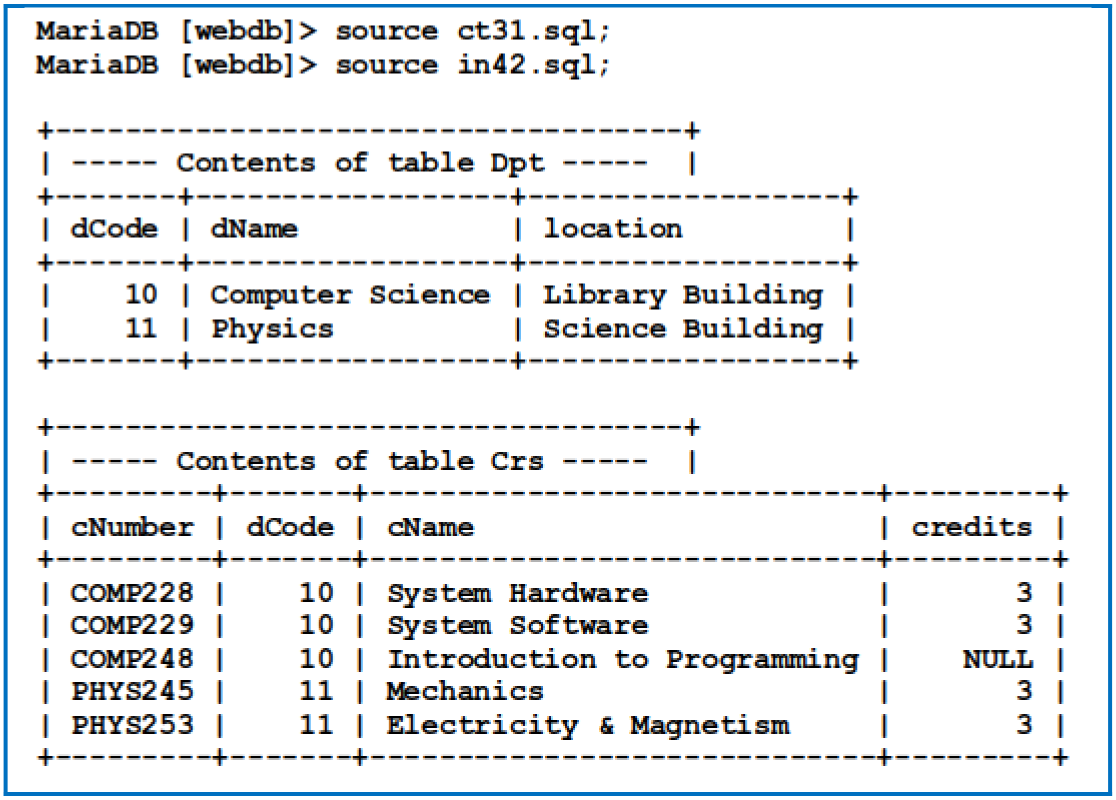
Script 4.14 Insert tuple without value for credits: in42.sql

add:

```
INSERT INTO Crs (cNumber, dCode, cName)
     VALUES('COMP248', 10, 'Introduction to
     Programming');
```

Save the script using the name, in42.sql, and terminate the editor.

First execute the script, ct31.sql, to create the tables without the not null constraint applied to credits.



```
MariaDB [webdb]> source ct31.sql;
MariaDB [webdb]> source in42.sql;

+----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+-----------------+-----------------+
| dCode | dName           | location        |
+-------+-----------------+-----------------+
|    10 | Computer Science | Library Building |
|    11 | Physics         | Science Building |
+-------+-----------------+-----------------+

+----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+---------------------------+---------+
| cNumber | dCode | cName                     | credits |
+---------+-------+---------------------------+---------+
| COMP228 |    10 | System Hardware           |       3 |
| COMP229 |    10 | System Software           |       3 |
| COMP248 |    10 | Introduction to Programming |    NULL |
| PHYS245 |    11 | Mechanics                 |       3 |
| PHYS253 |    11 | Electricity & Magnetism   |       3 |
+---------+-------+---------------------------+---------+
```

Figure 4.15 Script in42.sql, default: last part of output without constraint

```
    MariaDB [webdb]> source ct31.sql;
```

Execute the script to populate the tables.

```
    MariaDB [webdb]> source in42.sql;
```

The value of credits is missing for the new course. This can be prevented by the default constraint.

Figure 4.15 shows the last part of the script, in42.sql, without the constraint.

(b) With constraint

In the table, Crs, the value of credits is usually 3. If a value of credits is not specified use a default value of 3.
Change the constraint for attribute, credits, so that if it is not specified, its value is 3.
Edit the script, ct31, and call the changed script, ct42.sql.

```
  MariaDB [webdb]>system emacs  ct31.sql;
```

Script 4.15 Add default constraint: ct42.sql
change:

```
     credits DECIMAL(1)
```

to:

```
     credits DECIMAL(1) DEFAULT 3
```

Save the script using the name, ct42.sql, and terminate the editor.

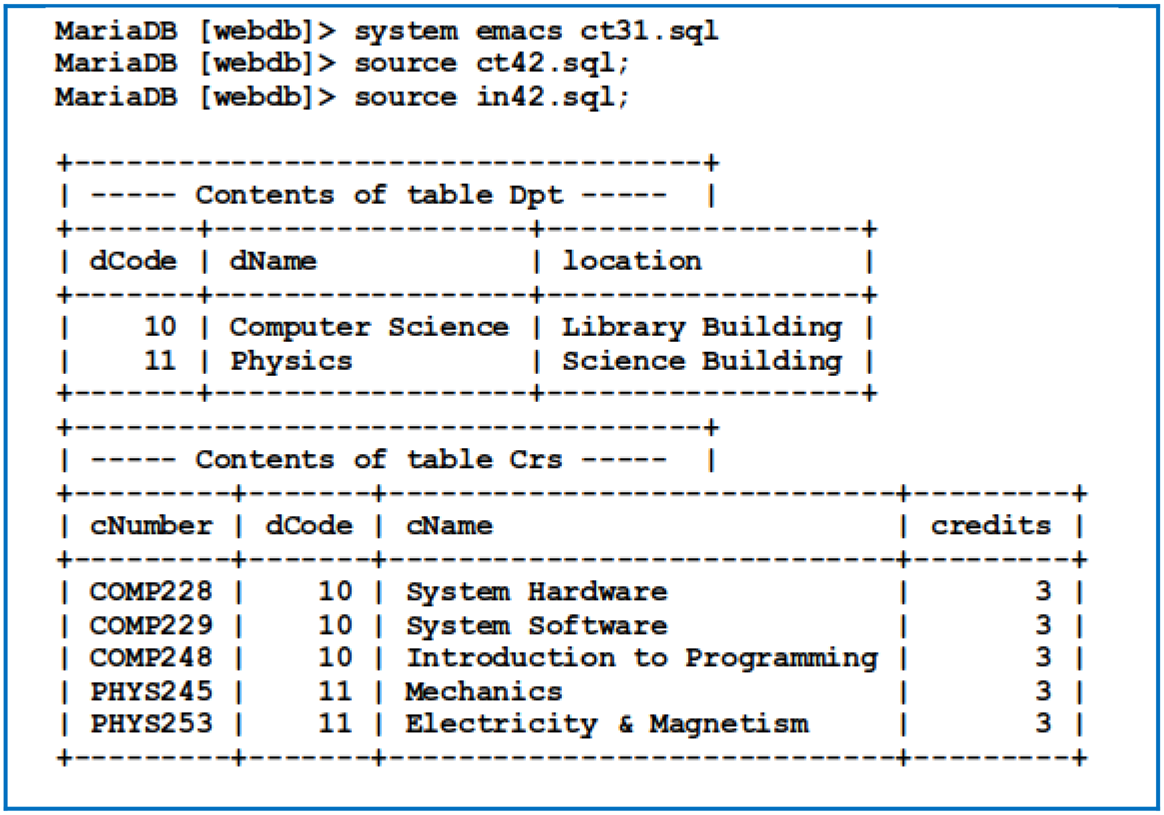
Execute the script to create the tables.

```
MariaDB [webdb]> source ct42.sql;
```

Execute the script, in42.sql, again, to populate the tables.

```
MariaDB [webdb]> source in42.sql;
```

The value of credits for the new course (COMP248) is 3, even though it wasn't specified by the INSERT command.



```
MariaDB [webdb]> system emacs ct31.sql
MariaDB [webdb]> source ct42.sql;
MariaDB [webdb]> source in42.sql;

+-----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
+-----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+----------------------------+---------+
| cNumber | dCode | cName                      | credits |
+---------+-------+----------------------------+---------+
| COMP228 |    10 | System Hardware            |       3 |
| COMP229 |    10 | System Software            |       3 |
| COMP248 |    10 | Introduction to Programming |       3 |
| PHYS245 |    11 | Mechanics                  |       3 |
| PHYS253 |    11 | Electricity & Magnetism    |       3 |
+---------+-------+----------------------------+---------+
```

Figure 4.16 Script in42.sql, default: last part of output with constraint

Figure 4.16 shows the last part of the script, in42.sql, with the constraint.

An important use of DEFAULT, which is not illustrated by either of the tables in this chapter, is to automatically record the date and time that a row in a table is inserted. This data would be stored in another column of the table that could be named, dateInserted. Also, the name of the user or the ID of the user who inserted the data can be automatically stored in the table in additional columns.

The built-in SQL function, SYSDATE, returns the date and time. The table, dual, is a dummy table to display one row of data where there is no table involved in the query. It is used to denote a dummy table as is done in the following examples in MySQL

```
MariaDB [webdb]>  SELECT
     SYSDATE()  AS DateTime
```

```
          FROM dual;

                     +---------------------+
                     | DateTime            |
                     +---------------------+
                     | 2019-05-23 14:50:46 |
                     +---------------------+
```

Another example of the use of dual is as follows: .

```
     MariaDB [webdb]> SELECT sqrt(144) from dual;
```

4.11.4 **Value Constraint**

4.11.4.1 CHECK - attribute values which are permitted

The constraint, CHECK, restricts the values which an attribute can have.In the table, Crs, the value of the credits has data type, number, with one digit, that is, the domain of credits is -9, -8 . . . -1, 0 , 1 . . . 8, 9. However, the credit value of a course certainly cannot be negative or 0. It is assumed that the credit value is either 3 or 4.

(a) Without constraint

Add a tuple to table, Crs, which has a value of 0 credits.
Edit the script, in32sql, and call the changed script, in51.sql.

```
  MariaDB [webdb]>system emacs  in32.sql;
```

Script 4.16 Insert tuple with value, 0, for credits: in51.sql
add:

```
  INSERT INTO Crs
  VALUES('COMP248', 10, 'Introduction to Programming', 0);
```

Save the script using the name, in51.sql, and terminate the editor.

Execute the script to populate the tables.

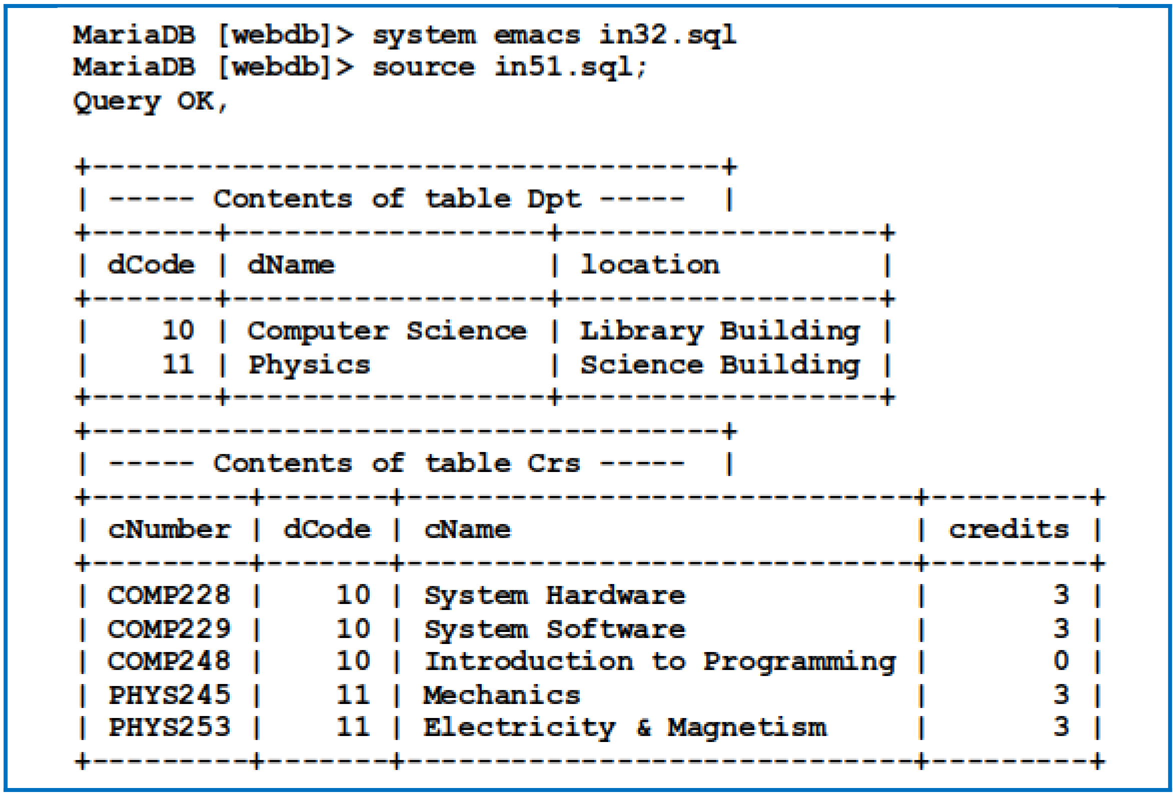
```
  MariaDB [webdb]> source in51.sql;
```

The value of the credits for the new course is 0, which is not a valid number of credits. This can be prevented by the check constraint.
Figure 4.17 shows the last part of the script, in51.sql, without the constraint.

(b) With constraint

Change the constraint for attribute, credits, so that the value of credits can be only 3 or 4.
Edit the script, ct31.sql, and call the changed script, ct51.sql.

```
  MariaDB [webdb]>  system emacs  ct31.sql;
```



```
MariaDB [webdb]> system emacs in32.sql
MariaDB [webdb]> source in51.sql;
Query OK,

+-----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+-----------------+-----------------+
| dCode | dName           | location        |
+-------+-----------------+-----------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+-----------------+-----------------+
+-----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+----------------------------+---------+
| cNumber | dCode | cName                      | credits |
+---------+-------+----------------------------+---------+
| COMP228 |    10 | System Hardware            |       3 |
| COMP229 |    10 | System Software            |       3 |
| COMP248 |    10 | Introduction to Programming |       0 |
| PHYS245 |    11 | Mechanics                  |       3 |
| PHYS253 |    11 | Electricity & Magnetism    |       3 |
+---------+-------+----------------------------+---------+
```

Figure 4.17 Script in51.sql, check: last part of output without constraint

Script 4.17 Add check constraint: ct51.sql
change:

```
   credits     DECIMAL(1)
```

to:

```
   credits     DECIMAL(1) CHECK (credits IN (3,4))
```

Save the script using the name, ct51, and terminate the editor.

Execute the script to create the tables.

```
   MariaDB [webdb]> source ct51.sql;
```

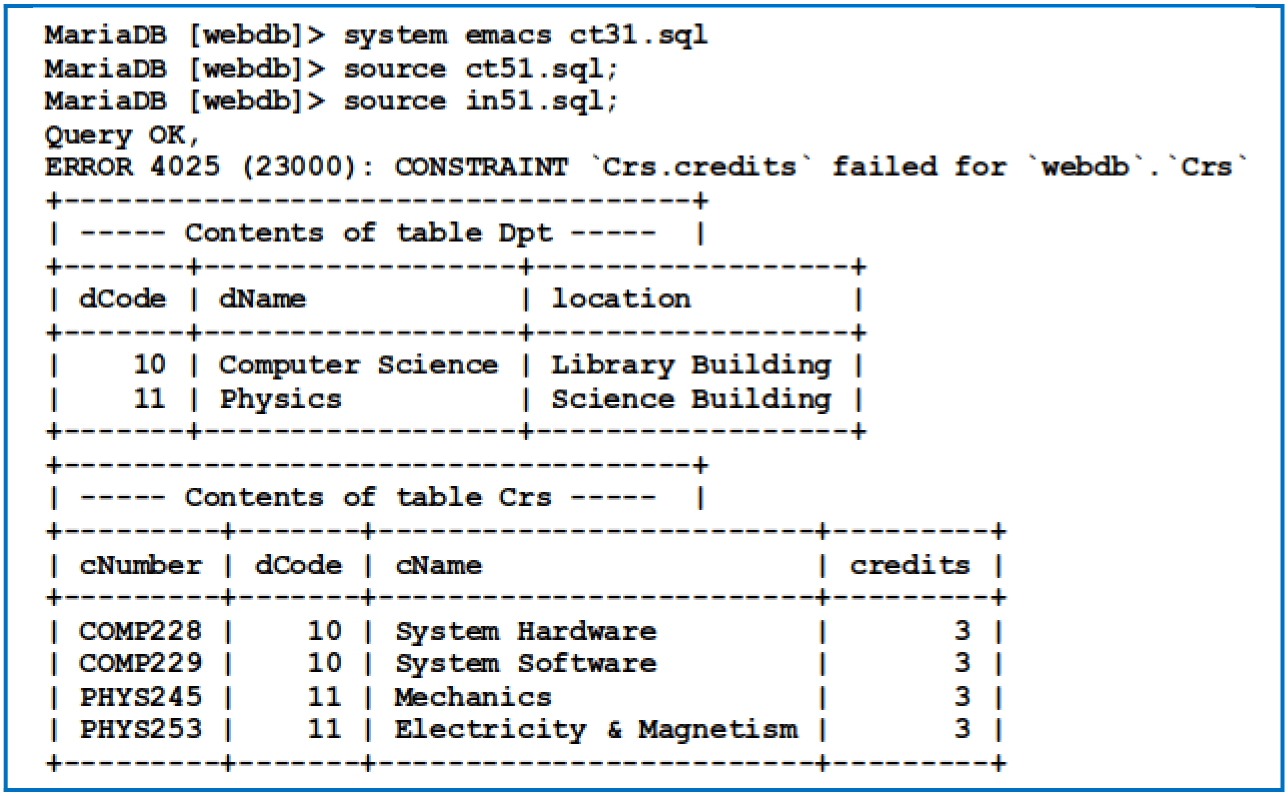
Execute the script, in51.sql, again, to populate the tables.

MariaDB [webdb]> source in51.sql;

An error message is displayed: check constraint violated. The tuple which violates the constraint (the tuple with credit value of 0) was not inserted into table, Crs.
Figure 4.18 shows the last part of the script, in51.sql, with the constraint.

4.11.4.2 DOMAIN - define a data type

The domain constraint defines a data type. This constraint is part of the SQL standard but is not supported at present by Oracle.



```
MariaDB [webdb]> system emacs ct31.sql
MariaDB [webdb]> source ct51.sql;
MariaDB [webdb]> source in51.sql;
Query OK,
ERROR 4025 (23000): CONSTRAINT `Crs.credits` failed for `webdb`.`Crs`
+-----------------------------------+
| ----- Contents of table Dpt -----  |
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
+-----------------------------------+
| ----- Contents of table Crs -----  |
+---------+-------+-------------------------+---------+
| cNumber | dCode | cName                   | credits |
+---------+-------+-------------------------+---------+
| COMP228 |    10 | System Hardware         |       3 |
| COMP229 |    10 | System Software         |       3 |
| PHYS245 |    11 | Mechanics               |       3 |
| PHYS253 |    11 | Electricity & Magnetism |       3 |
+---------+-------+-------------------------+---------+
```

Figure 4.18 Script in51.sql, check: last part of output with constraint

A domain is a set of allowed (legal) values for an attribute. The data types defined in SQL (VARCHAR and DECIMAL, for example) specify a domain. The DOMAIN constraint uses the SQL data types and, optionally, the constraints given by DEFAULT and CHECK to define a special data type based on the SQL data types

For each course credits must have a value (cannot be null), that can be either 3 or 4 and usually is 3. To satisfy these conditions, a domain can be created which specifies that credits is not null, gives a default value of 3 to credits and checks that the value of credits is either 3 or 4. The name of the domain (user-defined data type) is chosen to be CreditsType.

Create the domain for credits. Note that this is not supported in the current versions of Oracle, MySQL or MariaDB. The example below uses PostgreSQL, another free database which has additional features. The commands are slightly different as illustrated below.

```
[postgres@BP ~]$  pg_ctl -D /var/pgsql/data -l logfile start
[postgres@BP ~]$ psql WebDB
psql (11.5)
Type "help" for help.
WebDB=# CREATE DOMAIN CreditType as DECIMAL(1)
WebDB-# NOT NULL
WebDB-# DEFAULT 3
WebDB-#  CHECK (VALUE IN (1,2,3,6));
CREATE DOMAIN
```

```
WebDB=# CREATE TABLE Crs(
WebDB(# cNumber char(8),
WebDB(# dCode DECIMAL (3),
WebDB(# cName VARCHAR (30),
WebDB(# credits CreditType);
CREATE TABLE
WebDB-# \d Crs
Table "public.crs"

--------+--------------------------------+---------+---------
Column  |         Type         |Collation| Nullable| Default
--------+--------------------------------+---------+---------
cnumber | character(8)         |         |         |
dcode   | numeric(3,0)         |         |         |
cname   | character varying(30 |         |         |
credits | credittype           |         |         |
--------+--------------------------------+---------+---------

WebDB=# INSERT INTO CRS (cnumber,dcode ,cname,credits)
VALUES('comp333', 11, 'errorware', 5);
ERROR:  value for domain credittype violates check constraint
"credittype_check"

WebDB=# INSERT INTO CRS (cnumber,dcode ,cname,credits)
VALUES('comp333', 11, 'errorware', 6);
INSERT 0 1
WebDB=#  select * from crs;

             ----------+-------+-----------+---------
              cnumber  | dcode |   cname   | credits
             ----------+-------+-----------+---------
              comp333  |    11 | errorware |       6
             ----------+-------+-----------+---------
```

5 SQL Relationships

5.1 Introduction

A relationship means a connection or association, when the word is used in common everyday language. In the context of databases, a relationship has a similar but more precise meaning. A relationship (more formally, relationship set) in databases is an association or connection between two or more entity sets. The entity sets are like bricks and the relationship is like the mortar (material to bond bricks together).

The most common relationship is a binary relationship which is a relationship between two entity sets. A ternary relationship is a relationship between three entity sets. A quaternary relationship is a relationship between four entity sets. An n-ary relationship is a relationship between n entity sets where n is an integer which has the value 2 (for binary) or greater. There are rarely any relationships for n > 4, that is, among more than four entity sets. Incidently, in any of these relationship types, the same entity set may participate more than once. This chapter covers binary relationships.

Relationships are categorized by their multiplicity, which is based on the number of entities associated with each other from each of the entity sets in the relationship. For a binary relationship, there are three different types of multiplicities: many-to-many, many-to-one and one-to-one. So, in general, there could be three types of relationships, which use the same names as the multiplicities. The multiplicities one-to-many and many-to-one are equivalent since one-to-many can be converted to many-to-one by interchanging the two entity sets. Hence there are only three distinct cases for binary relationships instead of four. This chapter is concerned with the explanation of the three binary multiplicities.

The many-to-one relationship between two entity sets can be considered as the fundamental relationship. The other two types of relationships are variants of the many-to-one relationship. The many-to-many relationship may be visualized as two many-to-one relationships. The one-to-one relationship is a special case of a many-to-one relationship.

A relationship is implemented by the use of primary keys. When the primary key of one relation (table), sometimes referred as the parent, is used in another relation (table), sometimes called the child, it is said to be a foreign key in the second relation. The common value in the two tables, the foreign key and the corresponding primary key, establishes the relationship. In the case of a one-to-one relationship the foreign key is declared to be unique by the constraint, UNIQUE.

The multiplicity, many-to-one, was used to introduce the integrity constraints in the last chapter, and the same entity sets and relationship sets for many-to-one are used in this chapter.

An entity set corresponds to a relation in the relational model. Furthermore, a relationship is also modelled as a relation in the relational model.

A relation is called a table when it is implemented in a relational DBMS (database management system) and is created using the create table statement of SQL. So the terms, entity set, relation and table, have similar meaning and are sometimes used interchangeably

A relation (entity set) and relationship in databases are two very different concepts. Using the analogy above, a relation is like a brick and a relationship is like mortar (a material similar to concrete used to bond bricks together). However, in everyday language, relation and relationship are usually synonyms and either word can be used. For example, one can say, “There is always a relation between cause and effect” or “There is always a relationship between cause and effect”, and the meaning is the same. A beginner in the field of databases must be careful to distinguish between the two terms, relation and relationship.

5.2 Relationship Characteristics

There are two characteristics of a binary relationship between two entity sets: multiplicity and participation. These characteristics are determined solely from the semantics of the applications involving the entity sets and their relationship.

5.2.1 Multiplicity

One characteristic of a relationship between two entity sets is the multiplicity. The type of the relationship is implied by the name used to describe it.
There are three types of binary relationships:

(1) one-to-one
(2) many-to-one
(3) many-to-many

Table 5.1 Multiplicity

Type	Upper limit of the number of tuples in one relation that can be related to tuples in the other relation
one-to-one	at most one tuple of one relation is associated with one tuple of the other relation
many-to-one	one or more (many) tuples of one relation is associated with one tuple of the other relation
many-to-many	one or more (many) tuples of one relation is associated with one or more (many) tuples of the other relation

The multiplicity is a constraint which specifies the maximum number of entities from one side of the relationship that can be associated with an entity on the other side. The multiplicity is the upper limit of the number of instances in one entity set that can be related to instances of the other entity set. Multiplicity, one, means that the upper limit is one; only one instance (one tuple) of the entity set (relation) on the one side can be associated with an instance of the entity set (relation) on the other side. Multiplicity, many, means that the upper limit is more than one; many instances (tuples) of the entity set (relation) on the many side could be associated with one instance (tuple) of the entity set (relation) on the other side. The types of multiplicity and their meanings are shown in Table 5.1.

5.2.2 Participation

The other characteristic of a relationship is the participation . There are two types of participation for each type of multiplicity:

(1) optional (or partial)
(2) mandatory (or total)

The participation is a restriction which specifies the minimum number of relationships in which an instance of an entity set on one side of the relationship (a tuple of a relation), side A, can participate with the entity on the other side, side B. The participation is the lower limit of the number of tuples in one table that can be related to tuples in the other table. Participation, optional, means that the lower limit is zero; it means that some instances of the entity set on side A may not participate with any instance of the entity set on the other side, side B. Participation, mandatory, means that the lower limit is one; it means that each instance of the entity set on side A must participate with at least one instance of the entity set on the other side, side B.

The types of participation and their meanings are tabulated in Table 5.2.

Table 5.2 Participation

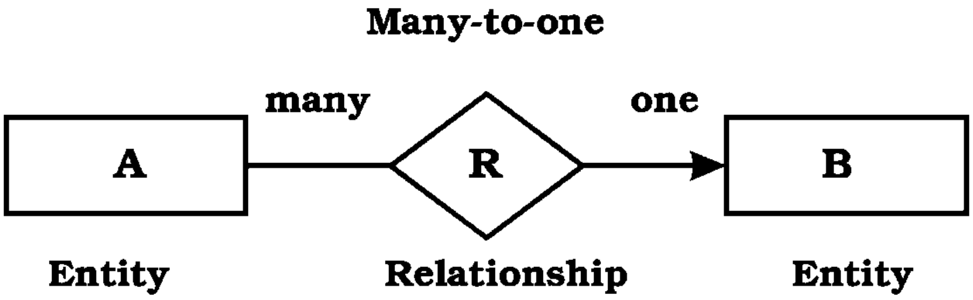
Type	Lower limit of the number of tuples in a relation that can be related to tuples in the other relation
optional or partial	lower limit is zero for a relation
mandatory or total	lower limit is one for a relation

The participation is enforced by schema-based constraints (constraints declared in the CREATE TABLE command of SQL) only in the table that contains the foreign key (child table).

To enforce the participation in the table referenced by the foreign key (parent table), programming must be used. It should be noted that the participation in the parent table is always optional when the tables are created because the parent table must be created before the related child table. So initially, tuples in the parent table will be related to zero tuples in the child table since the child table does not exist. After a tuple in the parent table is related to one or more tuples in the child table, programming can be used to enforce mandatory participation, that is, to prevent the removal of a tuple in the child table if it is the only tuple that is related to a tuple in the parent table.

The characteristics of a many-to-one relationship and related information are summarized in Figure 5.1. The example in the figure is the many-to-one relationship used in the last chapter: Course-GivenBy-Department. Note that A is a child table only if, in the conversion of the ER model to the relational model, the relationship R is combined with entity set A, the many side of the relationship. This is an example of a database design decision, which depends on how A is involved in other relationships.

ER Diagram



Example	Course	GivenBy	Department
Terminology	child (when R combined with A)		parent
Multiplicity	many (no arrow)		one (arrow)
Keys	foreign key	references →	primary key
Participation	total - foreign key not null partial - foreign key null		
Order of Tables			
Create	last		first
Remove	first		last
Populate	last		first
Delete Contents	first		last

Figure 5.1 Characteristics of a many-to-one relationship

5.2.3 Combinations of Multiplicity and Participation

For each multiplicity, there are four possible combinations of participation (optional and mandatory on each side of a binary relationship). Since there are three types of multiplicity, there are a total of 3 x 4 = 12 combinations of multiplicity and participation. The actual multiplicity and participation for a particular case is one of these 12 possibilities which depends on the semantics of the application being modelled by the ER diagram, that is, the meaning of the entities, their attributes and the relationship being modelled.

The number of combinations of participation for each multiplicity can be understood as follows.

On the one side of a relationship there are two possibilities for the participation.
(a) one side (maximum = 1): optional (minimum = 0)
(b) one side (maximum = 1): mandatory (minimum = 1)
So the one in a relationship means either (a) 0 or 1 (at most 1), or (b) 1 or 1 (exactly 1).

On the many side of a relationship there are the same two possibilities for the participation.
(c) many side (maximum > 1): optional (minimum = 0)
(d) many side (maximum > 1): mandatory (minimum = 1)
So the many in a relationship means either (c) 0 or more (at least 0), or (d) 1 or more (at least 1)

Using many-to-one as an example, if the participation on the one side is optional, (a), the participation on the many side can be either optional, (c), or mandatory, (d), which is two combinations. If the participation on the one side is mandatory, (b), the participation on the many side can also be either optional, (c), or mandatory, (d), which is two more combinations. So for the many-to-one multiplicity, there are four possible combinations.
(1) many is optional, one is optional
(2) many is optional, one is mandatory
(3) many is mandatory, one is optional
(4) many is mandatory, one is mandatory

By the same reasoning there are also four possible combinations for each of the one-to-one multiplicity and the many-to-many multiplicity.

5.2.4 Notation on ER Diagram

The multiplicity in an ER diagram is indicated by an arrowhead, meaning one, or the absence of an arrowhead, meaning many. The notation for many-to-one is shown in Figure 5.1.

The participation in an ER diagram is indicated in more than one way. One way is to indicate optional participation by a thin line and mandatory participation by a thick line or a double line. Another way to indicate mandatory participation on the one side is to use a rounded arrowhead instead of a pointed arrowhead. Using either method, the participation shown in Figure 5.1 is optional on both sides.

5.3 Relationship Instances

An instance of a relationship is an association of an instance from each of the entity sets involved in the relationship. Examples of relationship instances for a many-to-one relationship are shown in Figure 5.2. The participation is optional for both entity sets. This means that zero or more entities from both the one side and many side may participate in a relationship. The many-to-one relationship in Figure 5.2 is the same as in Figure 4.1 in the last chapter where entity A corresponds to Course, entity B corresponds to Department and relationship R corresponds to GivenBy. Attributes for the entities are not shown in order to simplify the diagram.

There are five tuples of A: a1, a2, a3, a4, a5, which correspond to courses in Figure 4.1 of the last chapter. There are four tuples of B: b1, b2, b3, b4 which correspond to departments. There are four relationship instances of R shown in Figure 5.2: r1, r2, r3, r4, which correspond to courses given by departments. Each course is given by (is related to) 0 (a5 is not given) or one (a1, a2, a3, a4) department. Each department gives (is related to) 0 (b2 does not give any courses) or more (b1 gives 3 courses, b3 gives 1 course) courses.

The multiplicity and participation of the relationship are shown in Figure 5.2 using the notation of the Unified Modelling Language: 0..* and 0..1. The notation has the form, x..y. The x is the participation, the minimum number of relationships in which a tuple can participate and is either 0, which is optional participation, or 1, which is mandatory participation. The y is the multiplicity, the maximum number of relationships in which a tuple can participate, which is either multiplicity, 1, or multiplicity, many (more than 1), represented by *.

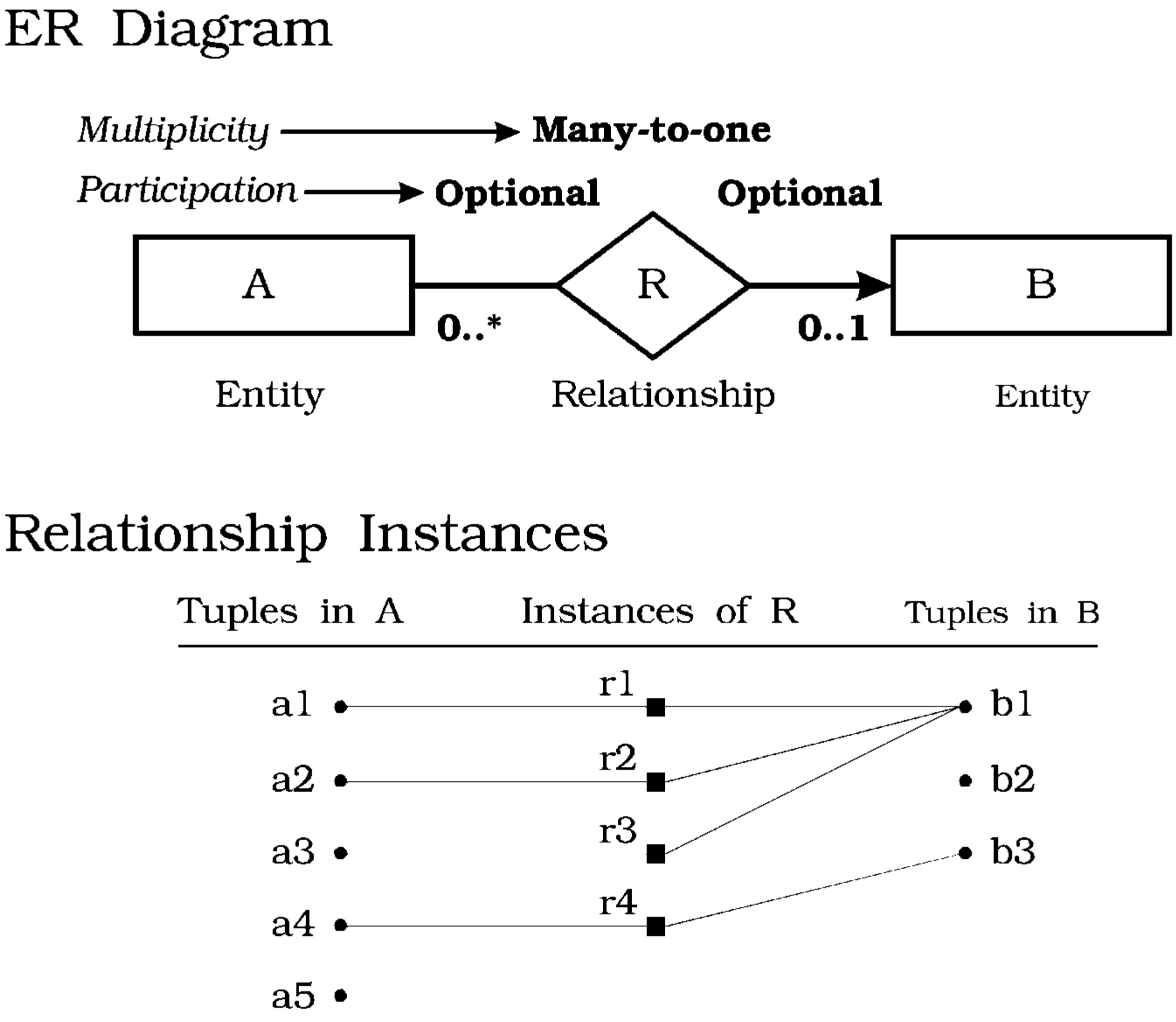


Figure 5.2 Relationship instances

In the ER diagram the notation on the left, 0..*, signifies that the multiplicity is many (*) and the participation is optional (0), and so 0 or more tuples in the table on the left can be related to a particular tuple in the table on the right. The notation on the right, 0..1, signifies that the multiplicity is one (1) and the participation is optional (0), and so 0 or 1 tuple in the table on the right can be related to a particular tuple in the table on the left.

5.4 Table Names

The name of every table in one database must be different from all other tables. There are no divisions of a user space which correspond to directories in a file system. In the previous chapter, the tables, Course and Department, were created as in this chapter but they were given the names, Crs and Dpt. To display all of the user tables in the database, at the database prompt use the command: MariaDB [webdb]> show tables;

Suppose that the tables in the last chapter were given the same names, Course and Department, as in this chapter. Should it be necessary to keep the tables from the previous chapter in the database, the names of the tables must be changed before the tables are recreated in this chapter. For example, change the name of table, Course, to Cars with the command:

```
ALTER TABLE Course RENAME TO Cars;
```

5.5 Constraint Names

It is desirable to give names to each constraint at the same time as the constraints are created by the CREATE TABLE command. When a constraint is violated, the DBMS displays an error message which gives the name of the constraint. The name of the constraint should be chosen so that it includes the name of the table for which the constraints is defined, which is the table name in the CREATE TABLE command. Also, the name of the constraint should include the constraint type. The name of every constraint in a database must be different, that is, a constraint name must not only be different than other constraint names in the same table but also in all other tables.

If a name of a constraint is not given by the programmer, the DBMS generates a name for the constraint which identifies the constraint by a number (described below). This is the name used by the DBMS in an error message. This DBMS-generated name is not useful because it contains no information about the nature of the constraint, and so it is advisable that the programmer gives a name to each constraint. In the last chapter, SQL Constraints, the constraints were not named by the programmer, which demonstrates that DBMS-named constraints may not be intuitive or informative.

The constraint is named by the programmer in the CREATE TABLE command. In this chapter primary key, foreign key and unique constraints are named. Therefore, both DBMS names for constraints (NOT NULL, CHECK, DEFAULT) and programmer names for constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE) are displayed in this chapter.

5.5.1 Defining Constraints

A constraint is named in the CREATE TABLE command using the keyword, CONSTRAINT. The syntax is

CONSTRAINT *constraint_name constraint_declaration*
The last argument, *constraint-declaration*, is the declaration of the constraint.

For example, if the attribute, cNumber, is the primary key of a table, Course, the named constraint is

```
CONSTRAINT CoursePK PRIMARY KEY(cNumber)
```

where, CoursePK, is chosen as the name of the constraint, *constraint_name*, because it starts with the name of the table, Course, and is followed by the type of constraint, PK (for primary key). The declaration of the constraint, *constraint_declartion*, is PRIMARY KEY(cNumber).

If the constraint is unnamed, then the declaration of the primary key is

```
PRIMARY KEY(cNumber)
```

and DBMS will give a name for the constraint, which may be of the form

Error no yyyy or SYS_C*xxxxxx*

where the x and y are decimal digits. An example of an Oracle-named constraint is SYS_C001234 and a MySQL/MariaDB error is 1451.

5.5.2 Listing of Defined Constraints

The Oracle system table, user_constraints, contains information about each constraint in all user tables. The description of the attributes of table, user_constraints, is displayed by the command,

```
DESC user_constraints
```

There are 20 attributes in this table. The four most important attributes are table_name, constraint_name, constraint_type and search_condition. The list of all user constraints using the four most important attributes are displayed by the command,

```
SELECT table_name,
     constraint_name,
     constraint_type,
     search_condition
FROM user_constraints;
```

The constraints for a particular table, Course for example, can be displayed by adding the WHERE clause after the FROM clause (before the ;) in the above command:

```
WHERE table_name = 'COURSE';
```

The names of the tables are capitalized in the table, user_constraints, and so must be capitalized in the WHERE clause.

There are five types of constraints, constraint_type, and each is designated by a single letter. The types are the following.

P - primary key
U - unique
R - foreign key (R stands for reference)
C - check, not null
V - check for views

Note that C is the type for both check and not null constraints. The reason for this is that NOT NULL is actually the check constraint, CHECK (*attributeName* IS NOT NULL).
The constraint, default, is not included in the table, user_constraints.

There is a search condition, search_condition, for only types C (check, not null) and V (check for views). Since there is no search condition for P (primary key), U (unique) and R (foreign key), at least these constraints should be named by the programmer, as done in this chapter.

In MySQL/MariaDB the constraints may be displayed using a query on the special table called information schema; an example is the following.

```
Select table_name,column_name,referenced_table_name,referenced_column_name
from    information_schema.key_column_usage
Where
 (referenced_table_name is not NULL
   or referenced_column_name is not NULL)
   and table_schema = 'webdb'
   and (table_name =   'Professor' OR table_name = 'Department');
```

5.6 Models

There are two models which are used to design a database and then a third stage to implement the database. The two models are the entity-relationship (ER) model and the relational model. The third stage converts the relational model to database tables using SQL. The three stages of database design and implementation are shown in Figure 5.3.

First the ER model is used to create the initial design of the database. The ER model is represented by a diagram, appropriately called an ER diagram. Then the relational model is derived from the ER model. The relational model is represented by schemas. A schema is the name of a relation (entity set) followed in brackets by the names of the attributes of the relation where each attribute is defined on an underlying domain. Finally, the relational schemas are used to implement relational database tables using the CREATE TABLE command of SQL.

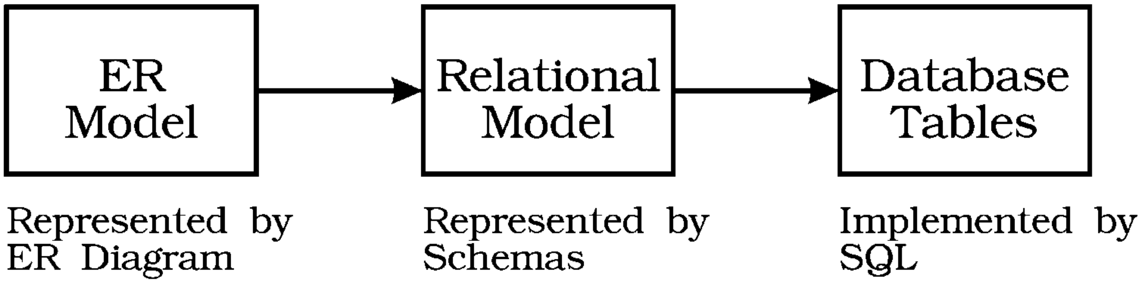


Figure 5.3 Three stages of database design and implementation

5.6.1 Entity-Relationship (ER) Model

Elements of the ER Model

(1) entity set - rectangle
(2) attribute - oval
(3) relationship set - diamond
(4) association between an entity set and its attributes - line

(5) association between an entity set and a relationship set - line with or without arrows to indicate the type of relationship (multiplicity)

The simplest case of a database is one that consists of only one entity set and in this case there is no relationship in the database. The ER model and the relational model are the same for one entity set, except for the method to represent the model. So it is necessary to consider a database that has at least two entity sets and at least two of the entity sets must be related to each other.

The ER model will be illustrated by two entities, A and B, which are related by relationship, R. There are three types of binary relationships and the notation, R1, R2 and R3 will be used for the three types as follows.

R1: many-to-many
R2: many-to-one
R3: one-to-one

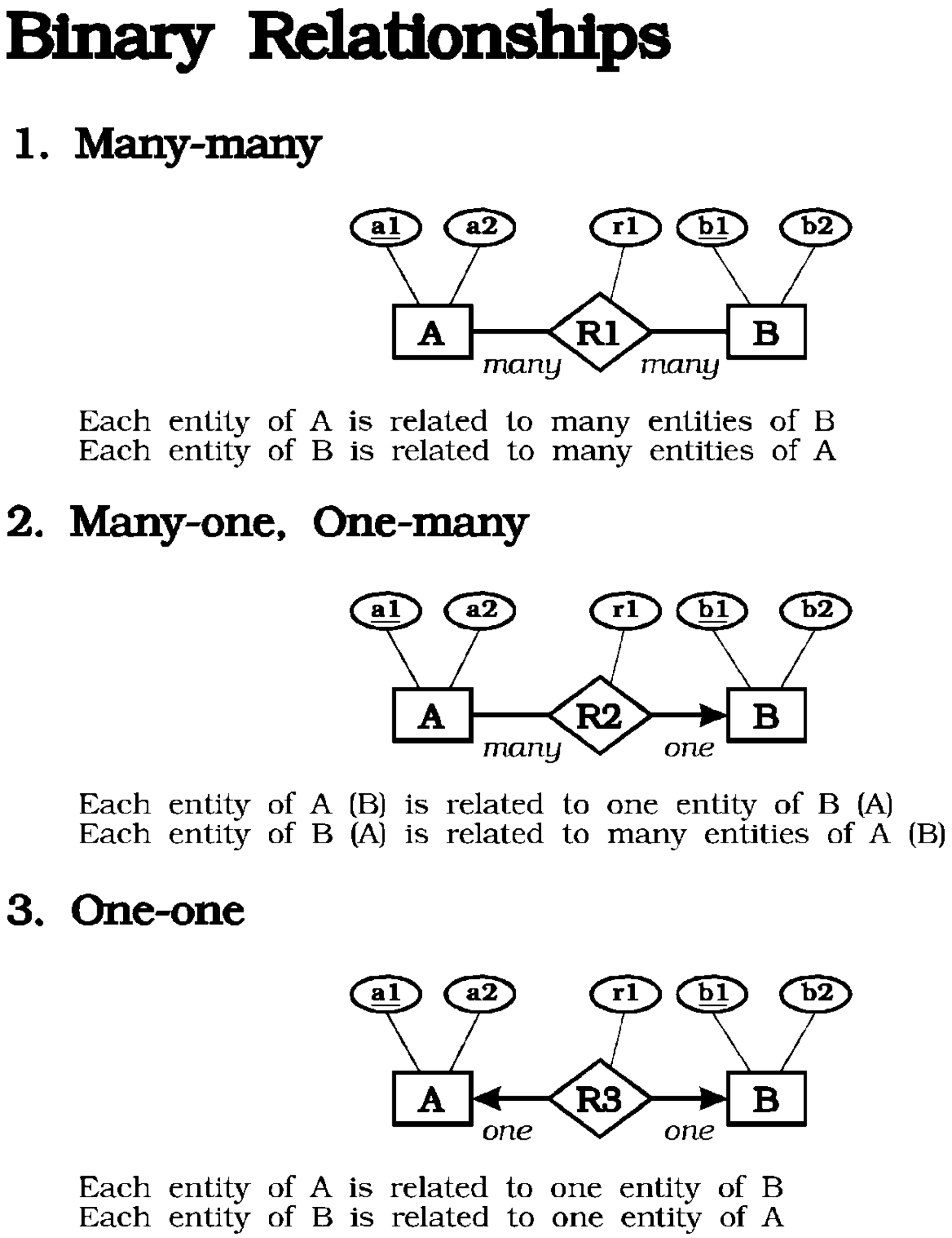
Figure 5.4 shows the ER diagram for each case. The only differences between the three diagrams is the symbol for the relationship and the notation that is used to indicate each of the three types of relationships. When the association between the entity set and relationship is many, the two are connected by a line without an arrow. When the association is one, the two are connected by a line with an arrow. Each entity set has two attributes. The attribute which is the primary key is underlined. Entity set, A, has attributes a1 and a2, and a1 is the primary key. Entity set, B, has attributes b1 and b2, and b1 is the primary key. The relationship, R, has one attribute, r1.

The entities in each case in Figure 5.4 could be the same in all three cases or they could be different. For example, if A is the entity set, Student, and B is the entity set, Project, the same entity sets could be related in three different ways. In each case the relationship could have the same name, Does, for example, although the relationship in each case is different. The relationship is one-to-one if each student does one project and each project is done by one student. This is often the case in courses where the project is small and no two students can do the same project. The relationship is many-to-one if each student does one project and each project is done by many (more than one) students, and no two groups of students can do the same project. This would be the case if the project in a course is large, for example, a large database project. Finally the relationship would be many-to-many if each student does many (more than one) projects and each project is done by many (more than one) students, for example if there are large projects from more than one course.

5.6.2 Relational Model

Basics of the Relational Model

In the relational model of a database, the data is stored in one or more relations. A relation can be visualized as a two-dimensional table. The tables corresponding to the relations in the database contain the data of the database. The columns of the table are the attributes of the relation and the rows of the table are the tuples of the relation. Both the entity sets and relationship sets of the ER model are represented by relations. The attributes of the relation (columns of the table) are the attributes of entity sets or relationship sets. The tuples of a relation (rows of the table) contain the data for each instance of an entity set or a relationship set.



Binary Relationships

1. Many-many

Each entity of A is related to many entities of B
Each entity of B is related to many entities of A

2. Many-one, One-many

Each entity of A (B) is related to one entity of B (A)
Each entity of B (A) is related to many entities of A (B)

3. One-one

Each entity of A is related to one entity of B
Each entity of B is related to one entity of A

Figure 5.4 ER diagrams for a binary relationship

A domain D is a set of values. Given a set of domains, D1, D2, ... , Dn, a relation R(A1, A2, ... An) is a n-ary tuple which is a subset of the cartesian product of the domains D1* D2* ... Dn. Since the relation is a set of tuples, there are no duplicates. Hence a subset of the set of attributes uniquely identify a tuple in the relation. Such a subset is called the primary key of the relation.

A foreign key establishes referential integrity in the relational model. If a relation R on a schema R has a subset of attributes X which is the primary key of another relation S on the relation schema S, then the referential integrity constraint requires that if a tuple r in R has the value x for X then a tuple s in S must exist with x as the primary key. X is called a foreign key of R.

Conversion of ER Model to Relational Model

In general, each entity in the ER model is converted to a relation and also each relationship in the ER model is converted to a relation. However, in the case of many-to-one and one-to-one relationships both an entity and its associated relationship can also be merged during the conversion into one relation and this is often, but not always, done.

Relation Schema

A relation is defined by a schema. A schema is the name of a relation (table) followed in brackets by the names of the attributes of the relation (columns of the table). If a relation (table) has name, T, and has attributes with names: t1, t2, t3, . . ., then the schema for this relation is written as: T(t1, t2, t3, . . .)

Additional notation is used in a schema to identify primary and foreign keys, and the unique constraint applied to a foreign key.

(a) primary key - underlined

(b) foreign key - in brackets by, fk ⇌ *tablename(pkname)*, where the foreign key, fk, references the primary key with name, *pkname*, of the table with name, *tablename*

(c) unique constraint - given in brackets by, unique, for a foreign key

If the name of the foreign key is the same as the name of the primary key which it references, then *pkname* can be omitted in the designation of the foreign key, (b). This shortens the notation but is not done in this chapter.

The schema with a foreign key is called a child. The schema which is referenced by the foreign key is called a parent.

Rules for Conversion of a Relationship to a Relation

Let the relation, R, be the relationship between n entity sets, E1, E2, E3, . . . , En.

1. Attributes of R are the primary keys of the n entity sets, and any attributes of the relationship.
2. The foreign keys of R are all the primary keys of the n entity sets.
3. The components of the primary key of R include only the primary keys of those entity sets which are on the many side of the relationship, except if the relationship is all ones (1-1-1 . . .-1).
4. If the relationship is all ones (1-1-1- . . .-1), the primary key from any one of the n entity sets is the primary key of R and the primary keys from the other n-1 entity sets are declared, UNIQUE.

Types of binary relationships

The three types of binary relationships are shown in Figure 5.4.

5.6.2.1 Many-to-many

There is only one choice in the relational model for a many-to-many relationship in Figure 5.4.

The relationship is converted to a relation, R1. The attributes of R1 are the primary keys of the two entity sets, A and B, and the attributes, if any, of the relationship. The foreign keys of R1 are all of the primary keys of the entity sets, A and B. The primary keys of R1 are also all of the primary keys of the entity sets since both sides of the relationship is many.

(1.1) Relationship is converted to a relation

A($\underline{\text{a1}}$, a2)	parent
R1($\underline{\text{a1}}$(fk ⇌ A(a1)), $\underline{\text{b1}}$(fk ⇌ B(b1)), r1)	child
B($\underline{\text{b1}}$, b2)	parent

5.6.2.2 Many-to-one

There are two choices in the relational model for a many-to-one relationship.

Relational model (2.1) below would be used if the relationship has one or more attributes whose values vary with time, that is, if r1 varied with time, since entity sets should be constructed without time-varying attributes. Also relational model (2.1) would be used if the entity set on the many side participates in many different relationships on the many side, in order to separate the attributes of the relationships from the attributes of the entity set. Otherwise, relational model (2.2) is used.

Note that the only difference between (1.1) and (2.1), below, is that b1 in R1 is part of the primary key (underlined) in the many-to-many relationship.

(2.1) Relationship is converted to a relation

A($\underline{\text{a1}}$, a2)	parent
R2($\underline{\text{a1}}$(fk ⇌ A(a1)), b1(fk ⇌ B(b1)), r1)	child
B($\underline{\text{b1}}$, b2)	parent

(2.2) Relationship is absorbed in the relation on the many side

A($\underline{\text{a1}}$, b1(fk ⇌ B(b1)), a2, r1)	child
B($\underline{\text{b1}}$, b2)	parent

5.6.2.3 One-to-one

There are four choices in the relational model for modelling a one-to-one relationship.

Either of relational models (3.1) or (3.2), below, is used if the participation of B is partial. Relational model (3.2) is used unless the relationship has one or more attributes and the participation of A is partial, and also the entity set A participates in many other relationships. When the participation of A is partial, many of the values for the foreign key and the value for r1 (the attribute of the relationship) in the relation for A under (3.2) would be null. The choice used is usually a database design decision based on performance issues (speed of execution of various SQL operations).

Either of relational models (3.3) or (3.4) is used if the participation of A is partial. Relational model (3.4) is used unless the relationship has one or more attributes and the participation of B is partial, and also the entity set B participates in many other relationships. The other reasons for (3.2) also are valid for model (3.4).

(3.1) Relationship is converted to a relation with same primary key as A

A($\underline{\text{a1}}$, a2)	parent
R3($\underline{\text{a1}}$(fk ⇌ A(a1)), b1(fk ⇌ B(b1), unique), r1)	child

B(b1, b2)   parent

(3.2) Relationship is absorbed in the relation A
A(a1, b1(fk ⇌ B(b1), unique), a2, r1)   child
B(b1, b2)   parent

(3.3) Relationship is converted to a relation with same primary key as B
A(a1, a2)   parent
R3(a1(fk ⇌ A(a1), unique), b1(fk ⇌ B(b1)), r1)   child
B(b1, b2)   parent

(3.4) Relationship is absorbed in the relation B
A(a1, a2)   parent
B(b1, a1(fk ⇌ B(b1), unique), b2, r2)   child

5.6.3 Example of Conversion of ER Model to Relational Model

In a university database, the entity set for a course could be involved in a number of relationships. Some of the relationships between a course and other entity sets are the following six relationships.

One-to-many relationships
Many courses are offered by one department - CrsDep
Many courses are offered by one faculty - CrsFac
Many courses are coordinated by one professor - CrsProf
Many-to-one relationship
Many sections are given of one course - CrsSec
Many-to-many relationships
Many major programs require many courses - CrsMaj
Many courses are prerequisites for many other courses - CrsPre

The ER diagram of entity set, Course, and the entity sets to which it is related is shown in Figure 5.5. The name of each entity set in the diagram conveys the meaning of the entity set. The name of each relationship set in the diagram is an abbreviation for the names of the two entity sets which it relates (except for the relationship of Course with itself). Each of the six entity sets has two attributes, one of which is the primary key (underlined). For the sake of simplicity, none of the six relationships have any attributes.

The conversion of the ER model in Figure 5.5 to the relational model can be done in different ways, depending on whether the one-to-many relationships are implemented as a table or are combined with the entity set on the many side of the relationship: the attributes in the ER-model have been renamed in the relational models given below.

Relational Model 1

One choice is to combine the three one-to-many relationships, CrsDep, CrsFac and CrsProf, with the entity set, Course. There are eight tables in this relational model with the schemas which follow.

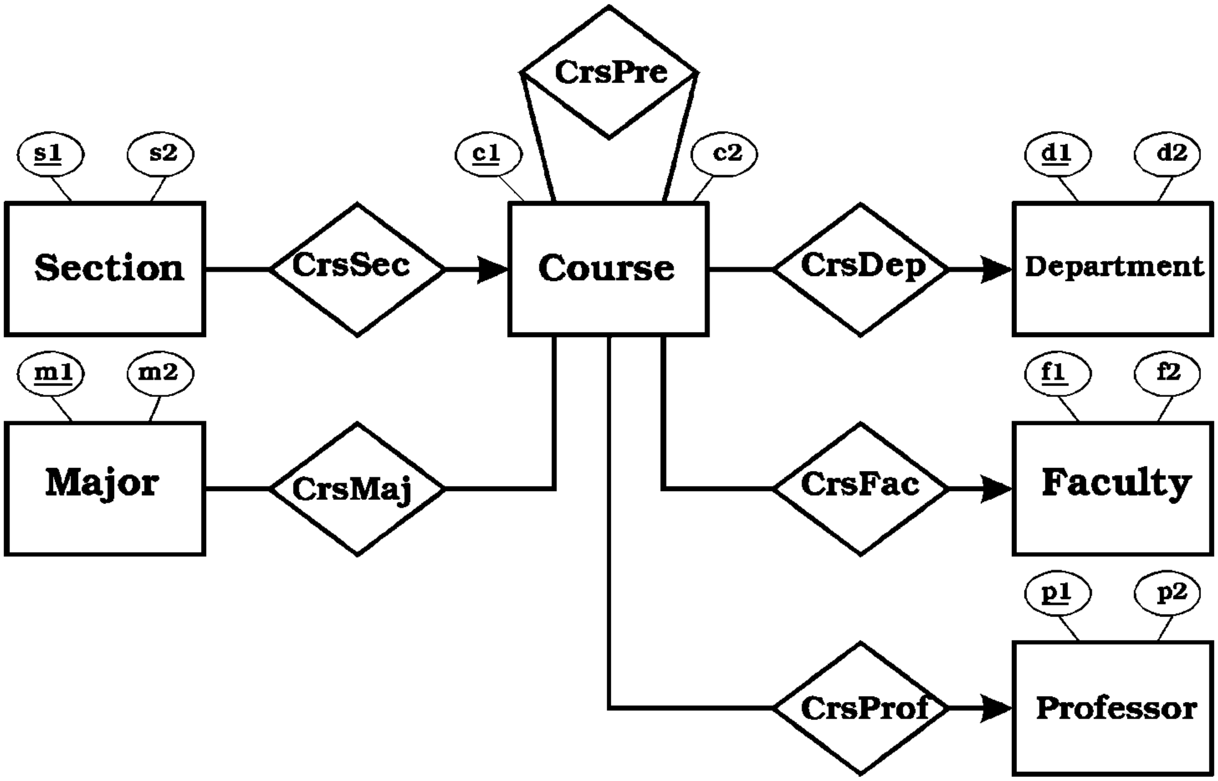


Figure 5.5 Example of ER model

Entity sets only

1. Department(d1, d2)
2. Faculty(f1, f2)
3. Professor(p1, p2)
4. Major(m1, m2)

Entity sets and relationships

5. Course(c1, c2, d1(fk ⇌ Department(d1)), f1(fk ⇌ Faculty(f1)), p1(fk ⇌ Professor(p1)))
6. Section(s1, c1(fk ⇌ Course(c1)))

Relationships only

7. CrsMaj(c1(fk ⇌ Course(c1)), m1(fk ⇌ Major(m1)))
8. CrsPre(cp1(fk ⇌ Course(c1)), cp2(fk ⇌ Course(c1)))

In this case the relation for Course has three more attributes than in Relation Model 2 which follows. However, one less join is required to get details for the department, faculty or professor for a given course. To get the details of the department offering a given course requires one join of the relation for course with the relation for department using the foreign key of department in course to do an equijoin with the key of the relation for department.

Relational Model 2

Another choice is to implement the three one-to-many relationships, CrsDep, CrsFac and CrsProf, as tables. There are eleven tables in this relational model with the schemas which follow.

Entity sets only

1. Department(d1, d2)
2. Faculty(f1, f2)
3. Professor(p1, p2)
4. Major(m1, m2)
5. Course(c1, c2)

Entity sets and relationships

6. Section(s1, c1(fk ⇋ Course(c1)))

Relationships only

7. CrsMaj(c1(fk ⇋ Course(c1)), m1(fk ⇋ Major(m1)))
8. CrsPre(cp1(fk ⇋ Course(c1)), cp2(fk ⇋ Course(c1)))
9. CrsDep(c1(fk ⇋ Course(c1)), d1(fk ⇋ Department(d1)))
10. CrsFac(c1(fk ⇋ Course(c1)), f1(fk ⇋ Faculty(f1)))
11. CrsProf(c1(fk ⇋ Course(c1)), p1(fk ⇋ Professor(p1)))

In this case there are three more tables than in the previous model, Relational Model 1. Also, for example, to get the details of the department of a given course, a join of course is required with the relation for CrsDep and then a join with the relation for Department.

Other Relational Models

There are other choices for relational models, in addition to Relational Model 1 and 2 above. For example, a third choice is to have no entity sets combined with relationships, that is, table 6 in Relational Model 2 would be replaced by two tables, one for Section alone and one for CrsSec.

The relational model that is chosen depends on the goals of the database design, which will not be discussed here.

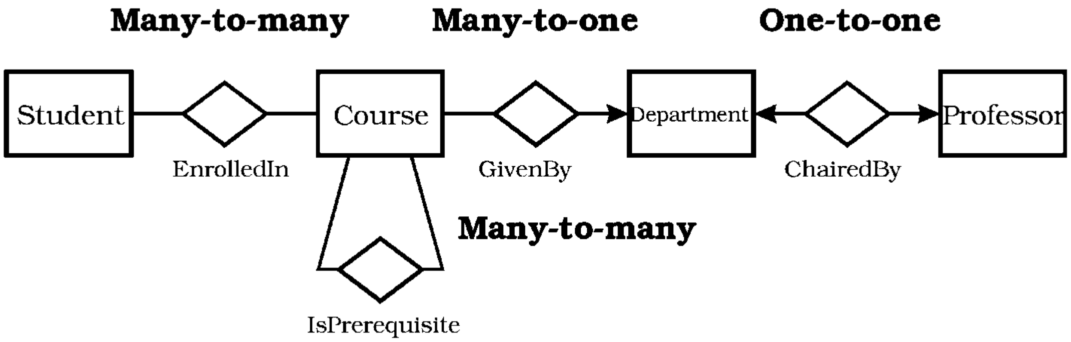
5.7 Sample Database

5.7.1 ER Diagram

The ER diagram in Figure 5.6 shows the database that is used to demonstrate the different relationships. Attributes are not shown in this figure to emphasize the relationships, but are shown in figures for the individual relationships which follow. The database contains four entity sets: Course, Department, Student, Professor. There are four different relationships among the four entity sets shown on the ER diagram. The relationships are chosen to demonstrate the three different multiplicities.

There could be more relationships among the entities than the four shown in Figure 5.6. For example, many professors work in one department and one department employs many professors. So there would be a many-to-one relationship, WorksIn, between Professor and Department, with Department on the one side. This illustrates that more than one relationship can coexist between the same two entities, ChairedBy and WorksIn in this case.

ER Diagram



Multiplicity

Many-to-many

Student-Course

each student is enrolled in many courses,
each course enrolls many students

Course-Course

each course has many courses as a prerequisite
each course is a prerequisite for many courses

Many-to-one

Course-Department

each department gives many courses
each course is given by one department,
(represented by arrowhead on department)

One-to-one

Department-Professor

each department is chaired by one professor
(represented by arrowhead on Professor)
each professor chairs one department
(represented by arrowhead on Department)

Figure 5.6 Sample database

5.7.2 Multiplicity

A statement of the multiplicity for each relationship is also given in Figure 5.6. An example of a many-to-one relationship and a one-to-one relationship is demonstrated. Two many-to-many relationships involving the table, Course, are given to demonstrate that a table (Course in this case) can be related to another table (Course-Student) or a table can be related to itself (Course-Course). A table can be related to itself, just as a table can be related to another table, by any one of the three basic types of relationships. From the semantics (meaning), the prerequisite relationship between the entity set, Course, with itself is many-to-many. The relationship between a table and itself is sometimes called a recursive relationship or unary relationship. It involves a relationship between different instances from the same entity set.

5.7.3 Order of Programming Demonstrations

First the four relationships are demonstrated individually. The many-to-one relationship, Course-Department, is demonstrated first because the other two types of relationships are derived from a many-to-one relationship. Next the many-to-many relationship, Student-Course, is demonstrated without the relationship between Course and Department. Then the one-to-one relationship, Department-Professor, is demonstrated without the relationship between Course and Department. Finally the many-to-many relationship, Course-Course, is demonstrated without the other relationships that involve Course. Then the four relationships are combined as shown in Figure 5.6.

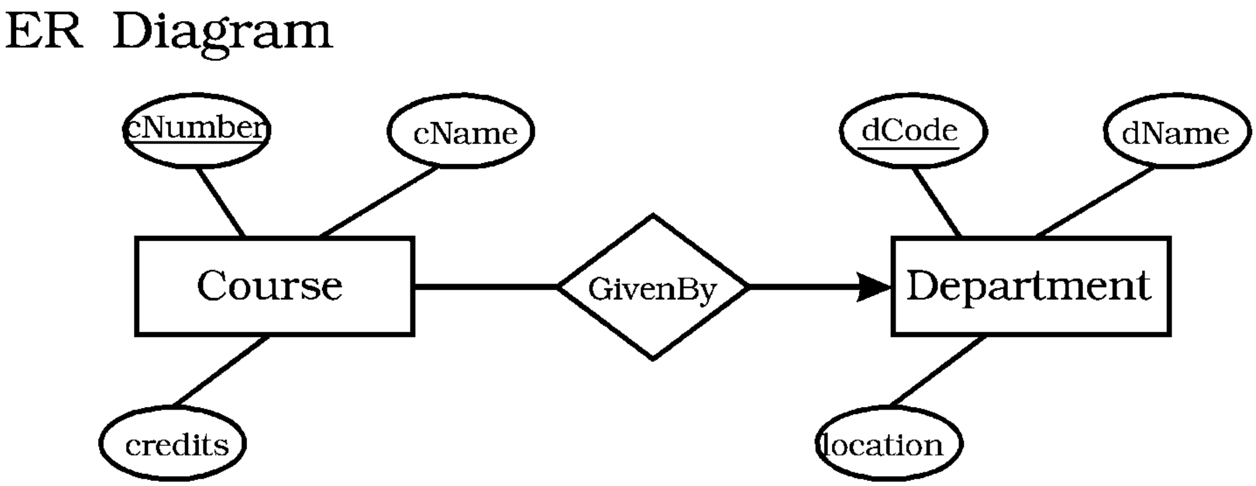


Figure 5.7 Many-to-one: Course-Department

5.8 Programming Demonstrations

5.8.1 Many-to-one: Course-Department

5.8.1.1 Semantics

Semantics is the meaning of the tables and relationship between the tables which is expressed by statements that are sometimes called business rules or just rules.

There are two entity sets, Course and Department, and one relationship set, GivenBy, in the ER diagram of Figure 5.7. A course is given by one department and a department can give many courses. Therefore, this relationship has multiplicity, many-to-one. This multiplicity, many-to-one, is indicated on the ER diagram by an arrow on the line from the diamond for the relationship pointing to the entity on the one side and no arrow on the line from the diamond to the entity on the many side.

The multiplicity in this case expresses the semantics or rule of the upper limit of the number of courses that can be offered by a department and the number of departments to which a course can belong. The upper limit, many, is the maximum number of courses a department can offer which is probably unspecified. The upper limit, one, is the maximum number of departments to which a course can belong, that is, a course is given by only one department.

The participation is the lower limit for the number of entities involved in a relationship. The lower limit of participation for the entity set on the many side can be either 0 (called optional or partial) or 1 (called mandatory or total). Probably the lower limit should be 0 for the case of a new department that does not yet have any courses. The lower limit of participation for the entity set on the one side can be either 0 or 1. The lower limit should be 0 for the case of a new course that is not yet approved or not yet assigned to a department. Therefore, the participation is optional on both sides of the relationship, that is, optional many and optional one.

In addition to multiplicity and participation, some of the other rules are the following. The type of constraints needed to enforce the rules are given in brackets.

- Every course has a number different from the course number for all other courses (primary key).
- Every department has a numerical code which is different from all other departments (primary key).
- The name of every course and the name of every department in the database must be recorded (not null).
- The name of every department must be different from all other departments (unique).
- The name of a course in a department must be unique and hence different from all other courses in the same department (unique). However, the name of a course in one department can be the same as the name of a course in another department (no constraint is needed).
- The credits of every course in the database must be recorded (not null).
- The credits of a course has a default value of 3, since most courses have a credit value of 3 (default).
- The credits of a course can only have the value 3 or 4 (check).

5.8.1.2 ER Diagram

The ER diagram is shown in Figure 5.7.

5.8.1.3 Schema

There are two choices for the schema, from subsection 5.6.2.2.
The schema with two tables is chosen, because the relationship has no attributes.

Course(cNumber, dCode (fk ⇌ Department), cName, credits) child
Department(dCode, dName, location) parent

The primary keys in each table are underlined.
The foreign key, dCode, is in Course and references the primary key, dCode, in Department.

5.8.1.4 Instance

Instances of Course and Department are given in Table 5.3 and Table 5.4.

Table 5.3 Course Instance

cNumber	dCode	cName	credits
COMP228	10	System Hardware	3
COMP229	10	System Software	3
PHYS245	11	Mechanics	3
PHYS253	11	Electricity & Magnetism	3

Table 5.4 Department Instance

dCode	dName	location
10	Computer Science	Library Building
11	Physics	Science Building

5.8.1.5 Create Tables

Use a script called, ct_mto1.sql (ct for CREATE TABLE, mto1 for many-to-one), to create the two tables in the schema with the constraints needed to enforce the rules described in the subsection, Semantics.

Start the editor and type the script.

```
MariaDB [webdb]> system emacs  ct_mto1.sql;
```

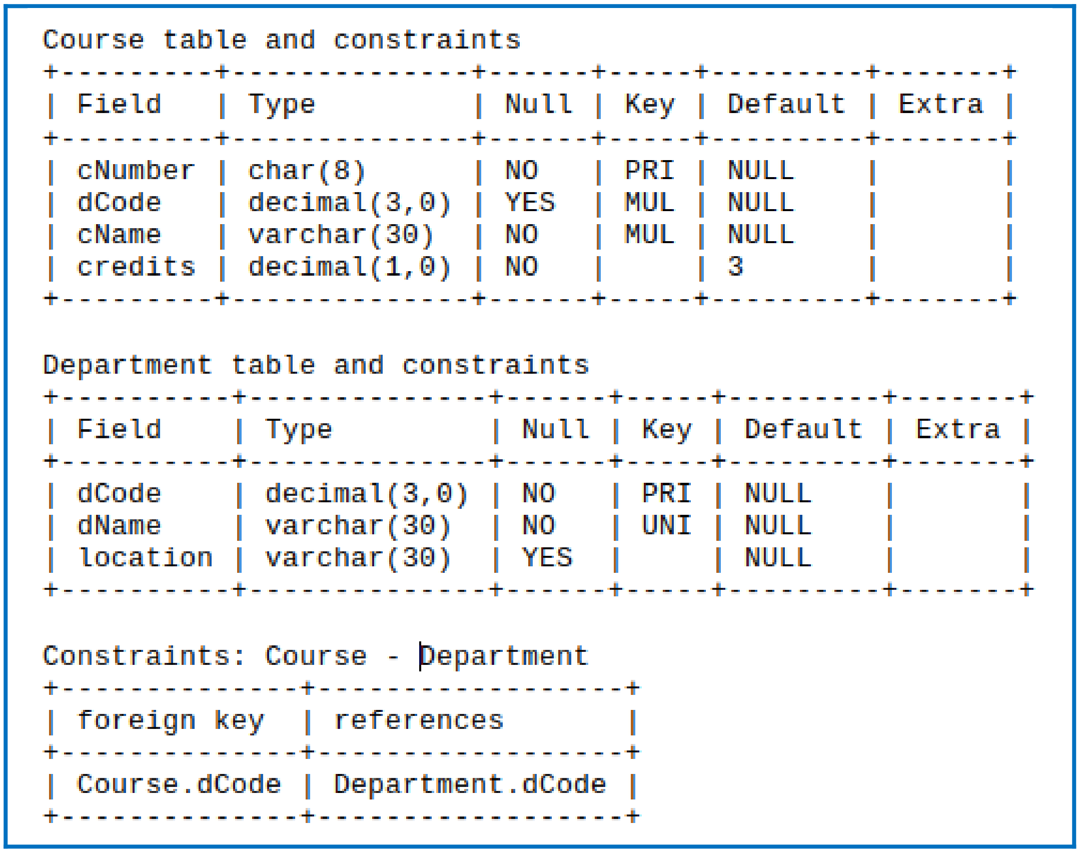
Script 5.1 Create Course-GivenBy-Department: ct_mto1.sql
(Start of script)

```
-- remove child tables first, then parent tables
DROP TABLE IF EXISTS Course;          -- child table
DROP TABLE IF EXISTS Department;      -- parent table
-- create parent tables first, then child tables
CREATE TABLE Department (
dCode        DECIMAL(3),
dName        VARCHAR(30) NOT NULL,
location     VARCHAR(30),
CONSTRAINT  DepartmentPK PRIMARY KEY(dCode),
CONSTRAINT  DepartmentNameUK UNIQUE (dName)) ;
CREATE TABLE Course (    cNumber            CHAR(8),
dCode        DECIMAL(3),
cName        VARCHAR(30) NOT NULL,
credits      DECIMAL(1) DEFAULT 3 NOT NULL CHECK (credits IN
(3,4)),
CONSTRAINT  CoursePK PRIMARY KEY(cNumber),
CONSTRAINT  CourseFK FOREIGN KEY(dCode) REFERENCES
Department(dCode),
CONSTRAINT  CourseNameindepartmentUK UNIQUE (cName, dCode));
```

```
-- display description of attributes of tables
select "" AS "   Specification for Course      ";
DESC Course;
select "" AS "   Specification for Department  ";
DESC Department;
-- display constraints
select "" AS "   Constraints in the database  ";
select
    concat(table_name, '.', column_name) as 'foreign key',
    concat(referenced_table_name, '.', referenced_column_name) as
'references'
from information_schema.key_column_usage
where (referenced_table_name is not NULL
    or referenced_column_name is not NULL)
    and table_schema = 'webdb'
    and (table_name =   'COURSE' OR table_name = 'DEPARTMENT');
```

(End of the script)

Save the script and terminate the editor.
Execute the script to create the tables.
MariaDB [webdb]> source ct_mto1.sql



```
Course table and constraints
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| cNumber | char(8)       | NO   | PRI | NULL    |       |
| dCode   | decimal(3,0)  | YES  | MUL | NULL    |       |
| cName   | varchar(30)   | NO   | MUL | NULL    |       |
| credits | decimal(1,0)  | NO   |     | 3       |       |
+---------+---------------+------+-----+---------+-------+

Department table and constraints
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| dCode    | decimal(3,0) | NO   | PRI | NULL    |       |
| dName    | varchar(30)  | NO   | UNI | NULL    |       |
| location | varchar(30)  | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+

Constraints: Course - Department
+--------------+------------------+
| foreign key  | references       |
+--------------+------------------+
| Course.dCode | Department.dCode |
+--------------+------------------+
```

Figure 5.8 Screen - Script ct_mto1.sql: last part of output

Figure 5.8 shows the last part of the output of script, ct_mto1.sql.

5.8.1.6 Populate Tables

Use a script called, in_mto1.sql (in for INSERT INTO, mto1 for many-to-one), to populate the tables, Course and Department, with the instances in Tables 5.3 and 5.4.
Start the editor and type the script.

MariaDB [webdb]> system emacs in_mto1.sql;

Script 5.2 Populate Course-GivenBy-Department: in_mto1.sql
(Start of script)

```
-- file name: in_mto1.sql
-- delete the contents of child tables first, then parent tables
DELETE FROM Course;         -- child table
DELETE FROM Department;   -- parent table
-- populate parent tables first, then child tables
INSERT INTO Department    -- parent table
VALUES(10, 'Computer Science', 'Library Building');
INSERT INTO Department
VALUES(11, 'Physics', 'Science Building');
INSERT INTO Course        VALUES('COMP228', 10, 'System
Hardware', 3);
INSERT INTO Course
VALUES('COMP229', 10, 'System Software', 3);
INSERT INTO Course
VALUES('PHYS245', 11, 'Mechanics', 3);
INSERT INTO Course
VALUES('PHYS253', 11, 'Electricity !& Magnetism', 3);
--display the contents of the tables
select "" AS "   Contents of table: Course     ";
SELECT * FROM Course;
select "" AS "   Contents of table: Department  ";
SELECT * FROM Department;
```

(End of the script)

Save the script and terminate the editor.
Execute the script to populate the tables.

MariaDB [webdb]> source in_mto1.sql

Figure 5.9 shows the last part of the output of script, in_mto1.sql.

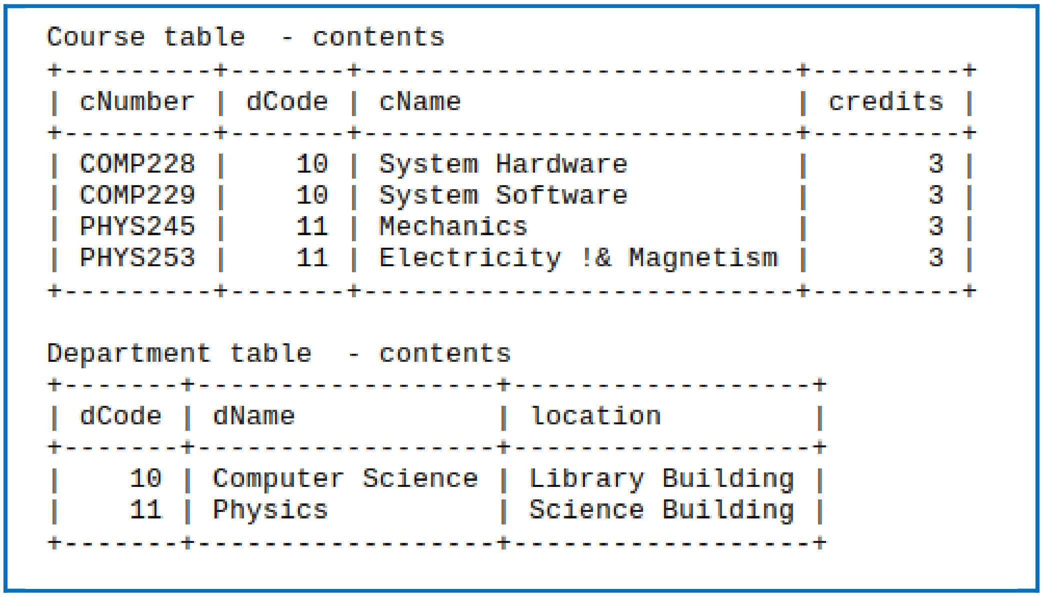
5.8.1.7 Integrity Constraints

The integrity constraints are in the CREATE TABLE commands for Department and Course.
(1) Department

PRIMARY KEY

```
CONSTRAINT  DepartmentPK PRIMARY KEY(dCode)
```

The department code, dCode, must be different for every department, that is, every tuple has a different value for dCode. Also, dCode cannot be null; every tuple has a value for dCode.



```
Course table  - contents
+---------+-------+--------------------------+---------+
| cNumber | dCode | cName                    | credits |
+---------+-------+--------------------------+---------+
| COMP228 |    10 | System Hardware          |       3 |
| COMP229 |    10 | System Software          |       3 |
| PHYS245 |    11 | Mechanics                |       3 |
| PHYS253 |    11 | Electricity !& Magnetism |       3 |
+---------+-------+--------------------------+---------+

Department table  - contents
+-------+------------------+------------------+
| dCode | dName            | location         |
+-------+------------------+------------------+
|    10 | Computer Science | Library Building |
|    11 | Physics          | Science Building |
+-------+------------------+------------------+
```

Figure 5.9 Screen - Script in_mto1.sql: last part of output

NOT NULL

```
dName      VARCHAR(30) NOT NULL
```

The name of the department, dName, must be given for every department, that is, every tuple has a value for dName.

UNIQUE

```
CONSTRAINT  DepartmentNameUK UNIQUE (dName)
```

The department name, dName, must be different for every department, that is, every tuple has different value for dName.

(2) Course

PRIMARY KEY

```
CONSTRAINT  CoursePK PRIMARY KEY(cNumber)
```

The course number, cNumber, must be different for every course, that is, every tuple has a different value for cNumber. Also, cNumber cannot be null; every tuple has a value for cNumber.

FOREIGN KEY

```
CONSTRAINT CourseFK FOREIGN KEY(dCode) REFERENCES
Department
```

The department code, dCode, which identifies the department to which the course belongs must exist in the table, Department. The foreign key constraint connects the two tables, Department and Course, and enforces the rule that a course cannot be in a department that does not exist. There are two consequences of this: (a) a course cannot be inserted if it is given by a department that does not exist, and (b) a department cannot be deleted if one or more courses are given by the department.

Since there is only one tuple for each course, because the course number, cNumber, is the primary key, a course can belong to only one department. Since the foreign key by default is declared NULL, a course can belong to no department, which could occur, for example, if a new course is proposed but not yet approved. So the participation on the many side of the relationship is optional (or partial). The foreign key constraint applied to dCode establishes a many-to-one relationship between Department and Course. Department is the parent table and Course is the child table.

UNIQUE

CONSTRAINT CourseNameUK UNIQUE (cName, dCode))

The course name, cName, must be different for every course in the same department, that is, every tuple has a different combination of values of cName and dCode.

NOT NULL

```
cName     VARCHAR(30) NOT NULL
```

The name of the course, cName, must be given for every course, that is, every tuple has a value for cName.

NOT NULL

```
credits  DECIMAL(1) DEFAULT 3 NOT NULL CHECK (credits
IN (3,4))
```

The value of credits, credits, must be given for every course, that is, every tuple has a value for credits.

DEFAULT

The value of credits, credits, will be assigned the value of 3 if the value of credits is not given when the course is inserted into the database.

CHECK

The value of credits, credits, can only be 3 or 4.

5.8.1.8 Test Many-to-one Foreign Key Constraint

First, insert another course: COMP248, with name, Introduction to Programming, with 3 credits but without a department to which the course belongs. This is permitted because the foreign key can be null, that is, dCode, is by default declared NULL (not declared, NOT NULL).

```
MariaDB [webdb]>  INSERT INTO Course
      VALUES('COMP248', null, 'Introduction to Programming',
      3);
MariaDB [webdb]> SELECT * FROM Course;
MariaDB [webdb]> SELECT * FROM Department;
```

Course

(1) a course must be given by a department that exists

Allocate the course, COMP248 to a department which does not exist. This could happen, for example, when dCode for Computer Science is typed incorrectly, 12 instead of 10. This is rejected.

```
MariaDB [webdb]> UPDATE Course
      SET dCode = 12
```

```
        WHERE cNumber = 'COMP248';
MariaDB [webdb]> SELECT * FROM Course;
```

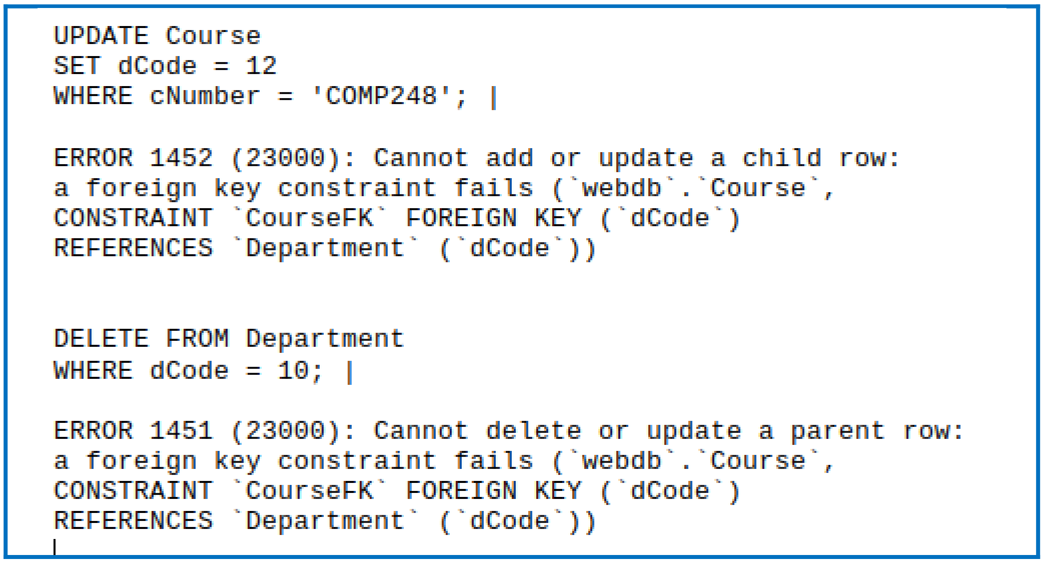
(2) Assign the course, COMP248, to the Computer Science department which exists. This is allowed.

```
MariaDB [webdb]> UPDATE Course
        SET dCode = 10
        WHERE cNumber = 'COMP248';
MariaDB [webdb]> SELECT * FROM Course;
```

Department
(3) a department cannot be deleted if the department gives one or more courses
Delete the department, Computer Science, (dCode = 10) which gives courses. This is rejected

.

```
MariaDB [webdb]> DELETE FROM Department
        WHERE dCode = 10;
MariaDB [webdb]> SELECT * FROM Department;
```



```
UPDATE Course
SET dCode = 12
WHERE cNumber = 'COMP248'; |

ERROR 1452 (23000): Cannot add or update a child row:
a foreign key constraint fails (`webdb`.`Course`,
CONSTRAINT `CourseFK` FOREIGN KEY (`dCode`)
REFERENCES `Department` (`dCode`))


DELETE FROM Department
WHERE dCode = 10; |

ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails (`webdb`.`Course`,
CONSTRAINT `CourseFK` FOREIGN KEY (`dCode`)
REFERENCES `Department` (`dCode`))
|
```

Figure 5.10 Screen - Many-to-one, foreign key error messages

Figure 5.10 shows the error messages from the UPDATE and DELETE FROM commands.

5.8.2 Many-to-many: Student-Course

5.8.2.1 Semantics

There are two entity sets, Student and Course, and one relationship set, EnrolledIn. The ER diagram is given in Figure 5.11. A student is enrolled in many courses and a course enrolls many students. Therefore, this relationship is many-to-many. The multiplicity, many-to-many, is indicated on the ER diagram by the absence of an arrow on both sides of the relationship.

The multiplicity, the upper limit of many courses and many students, is specified. Assume that a full-time student cannot take more than 6 courses at one time. A course can only enroll a number of students that does not exceed the seating capacity of the classroom, which is assumed to be 50. Also the maximum enrollment (size) in a course may be set by a department policy based on teaching methods and so could be less than the seating capacity of the classroom. Furthermore, different departments may have different policies for the size of a course and the size may also depend on the level of the course (junior or senior level). These rules cannot be enforced by schema-based constraints and so programming (application-based constraints) must be used.

The participation, the lower limit of many courses and many students, is also specified. A student may be allowed to take no courses in one academic term and still remain a student. A course can be given, for example, in only one of three academic terms each year and so will have an enrollment of 0 when it is not given. However, when a course is given, there is likely some minimum enrollment which is required, say 10 students. This rule would be enforced by programming.

In addition to multiplicity and participation, some of the other rules are the following. The constraints needed to enforce the rules are given in brackets.

- The rules for Course are given in subsection 5.8.1.1.
- Every student has an identification number different from all other students (primary key).
- The name of every student and the gender of every student in the database must be recorded (not null).
- The gender of a student is either male, m, or female, f, (check).
- Each student is enrolled in each course one time at most (primary key of EnrolledIn).

5.8.2.2 ER Diagram

The ER diagram is shown in Figure 5.11.

5.8.2.3 Schema

There is only one choice for the schema, from subsection 5.6.2.1.

Student(sID, sName, major, gender)	parent
EnrolledIn(sID (fk ⇌ Student), cNumber (fk ⇌ Course), grade)	child
Course(cNumber, cName, credits)	parent

The primary keys in each table are underlined.

A many-to-many relationship (EnrolledIn) is always converted to a table with attributes which are the primary keys of the two tables related by the relationship and any attributes which belong to the relationship (grade in this case). The primary key of the relationship (EnrolledIn) consists of the primary keys of the two tables which it relates (sID, cNumber). In addition, the components of the primary key of the relationship table (EnrolledIn) are the foreign keys of the two tables which it relates.

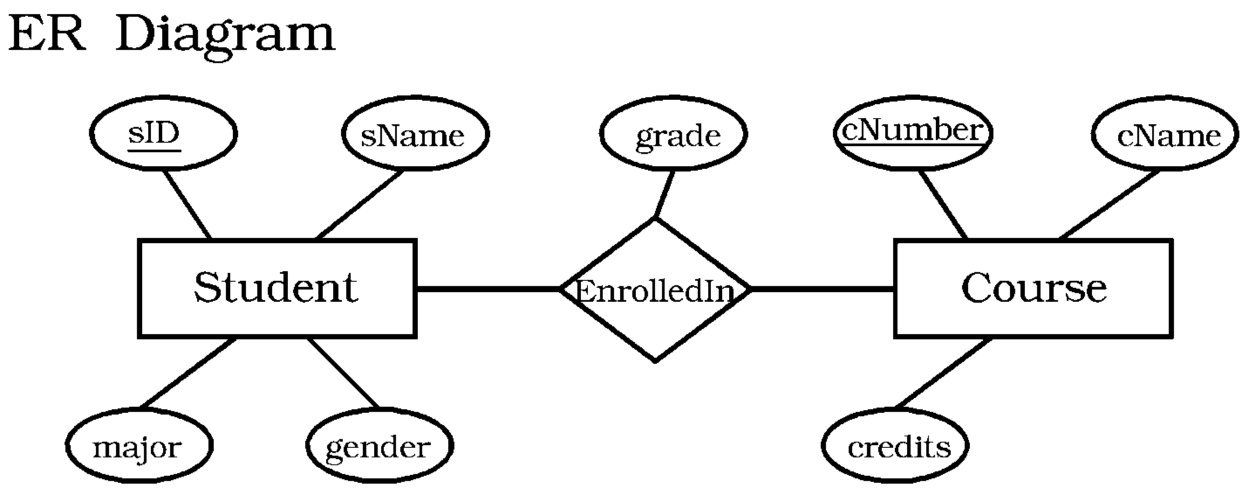


Figure 5.11 Many-to-many: Student-Course

EnrolledIn-Student is a many-to-one relationship where EnrolledIn is the child table, Student is the parent table and the foreign key is cID in EnrolledIn which references table, Student. Also EnrolledIn-Course is a many-to-one relationship where EnrolledIn is the child table, Course is the parent table and the foreign key is cNumber in EnrolledIn which references table, Course.

Table 5.5 Student Instance

Student

sID	sName	gender	major
1111222	Jerry	m	COMP
2222333	Elaine	f	PHYS
3333444	George	m	CHEM
4444555	Kramer	m	

The data could be stored in the two tables, Student and Course, but there would be redundancy in the tables, that is, the same data would be recorded in the tables more than once. For example, if enrollment data was stored in table, Course, the name of a course would be repeated many times. This is illustrated in a table below. Redundancy is avoided by converting the relationship, EnrolledIn, to a table.

5.8.2.4 Instance

Instances of Student, EnrolledIn and Course are given in Table 5.5, Table 5.6 and Table 5.7. The instance of course, Table 5.7, is the same as Table 5.3 but without the foreign key, dCode, which is in Table 5.3. The instance in Table 5.7 is called Course Instance2 to distinguish it from Course Instance in Table 5.3.

Table 5.6 EnrolledIn Instance

EnrolledIn

sID	cNumber	grade
1111222	COMP228	C+
1111222	COMP229	
4444555	COMP228	A+
2222333	PHYS245	B

Table 5.7 Course Instance2

Course

cNumber	cName	credits
COMP228	System Hardware	3
COMP229	System Software	3
PHYS245	Mechanics	3
PHYS253	Electricity & Magnetism	3

If the relationship, EnrolledIn, was not converted to a table, then all of the data would have to be stored in the two tables corresponding to the entity sets, Student and Course. This is possible but would produce redundancy (duplication of data) in one of the tables. The data in EnrolledIn could be included in either Student or Course. If the relationship, EnrolledIn, is combined with the table for the entity set, Course, the instance of Course becomes a table with five columns. A possible example of such a table with five rows of data is shown below. Storing data for a new course will also require putting null values for sID and grade. The redundancy in this table is the data about the course, COMP228, which is repeated in two rows, because two students are enrolled in COMP228. If instead, for example, 10 persons were enrolled in each of the four courses, there would be 40 rows in the combined table with data about each course repeated 10 times. It should be clear that it is better to create another table, EnrolledIn, to store enrollment data. The table below is not numbered because it is not a table in the database.

5.8.2.5 Create Tables

Use a script called, ct_mtom.sql (ct for CREATE TABLE, mtom for many-to-many), to create the three tables in the schema with the constraints needed to enforce the rules described in the subsection, Semantics.

Start the editor and type the script.

```
MariaDB [webdb]> emacs ct_mtom.sql
```

Course and EnrolledIn combined in one table

cNumber	cName	credits	sID	grade
COMP228	System Hardware	3	1111222	C+
COMP228	System Hardware	3	4444555	A+
COMP229	System Software	3	1111222	
PHYS245	Mechanics	3	2222333	B
PHYS253	Electricity & Magnetism	3		

Script 5.3 Create Student-EnrolledIn-Course: ct_mtom.sql
(Start of script)

```
-- file name: ct_mtom.sql
-- file name: ct_mtom.sql
-- create many-to-many relationship
-- there are three tables: Student, EnrolledIn, Course
-- There are two foreign keys in EnrolledIn:
--                  one references Student, one references Course
-- EnrolledIn is a child table, Student and Course are parent
tables
-- remove child tables first and then parent tables
DROP TABLE IF EXISTS EnrolledIn;     -- child table
DROP TABLE IF EXISTS Student;        -- parent table
DROP TABLE IF EXISTS Course;         -- parent table
-- create parent tables first, then the child tables.
CREATE TABLE Student (  -- parent table
sID    DECIMAL(7),
sName  VARCHAR(20) NOT NULL,
gender CHAR(1) NOT NULL CHECK (gender IN ('m','f')),
major  CHAR(4),
CONSTRAINT  StudentPK PRIMARY KEY(sID));
CREATE TABLE Course (   -- parent table
cNumber CHAR(8),cName     VARCHAR(30) NOT NULL,
credits DECIMAL(1) DEFAULT 3 NOT NULL CHECK (credits IN (3,4)),
CONSTRAINT  CoursePK PRIMARY KEY(cNumber));
CREATE TABLE EnrolledIn (  -- child table
sID     DECIMAL(7),
cNumber     CHAR(8),
grade  CHAR(2),
CONSTRAINT  EnrolledInPK PRIMARY KEY(sID, cNumber),
CONSTRAINT  EnrolledIn_StudFK FOREIGN KEY(sID) REFERENCES
Student(sID),
CONSTRAINT  EnrolledIn_CourFK FOREIGN KEY(cNumber) REFERENCES
Course(cNumber));
-- display description of attributes of tables
```

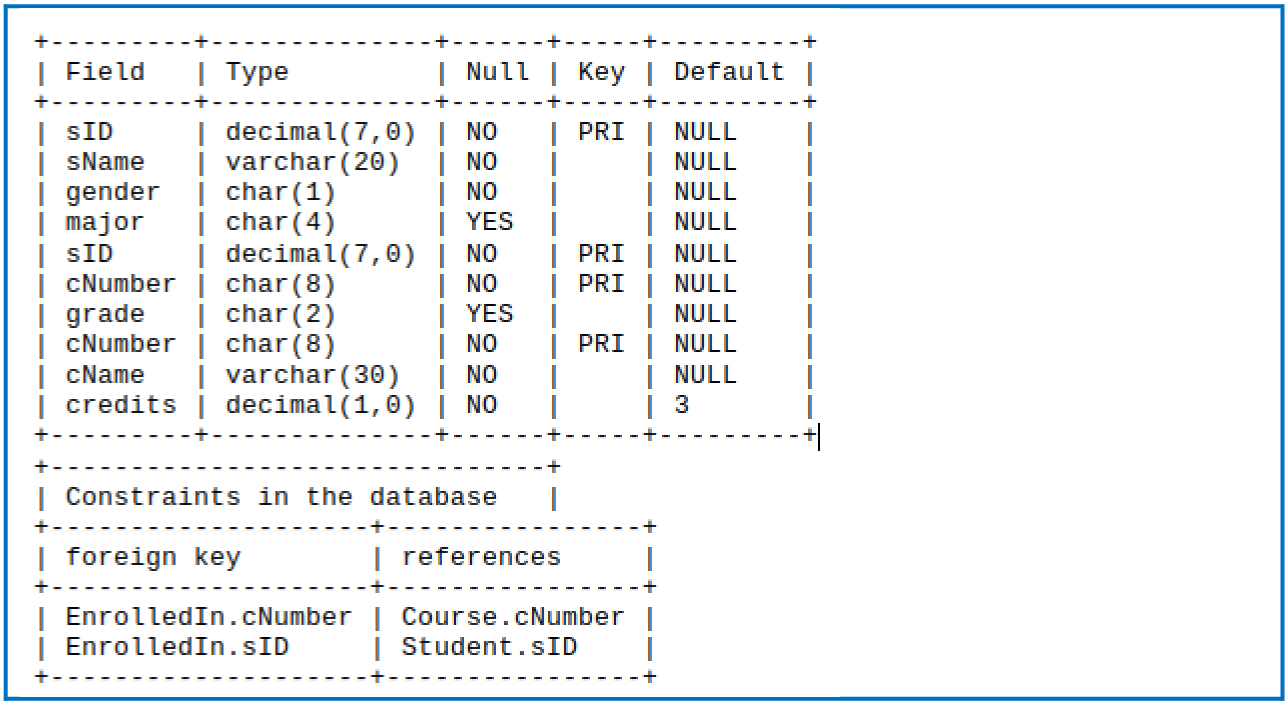
```
select "" AS "   Specification for  Student    ";
DESC Student;
select "" AS "   Specification for  EnrolledIn    ";
DESC EnrolledIn;
select "" AS "   Specification for  Course    ";
DESC Course;
-- display constraints
select "" AS "   Constraints in the database  ";
select
    concat(table_name, '.', column_name) as 'foreign key',
    concat(referenced_table_name, '.', referenced_column_name) as
'references'
from information_schema.key_column_usage
where  referenced_table_name is not NULL
    or referenced_column_name is not NULL)
    and table_schema = 'webdb'  and (table_name =   'Student' OR
```

(End of the script)

Save the script and terminate the editor.

Execute the script to create the tables.

MariaDB [webdb]> source ct_mtom.sql;



```
+---------+--------------+------+-----+---------+
| Field   | Type         | Null | Key | Default |
+---------+--------------+------+-----+---------+
| sID     | decimal(7,0) | NO   | PRI | NULL    |
| sName   | varchar(20)  | NO   |     | NULL    |
| gender  | char(1)      | NO   |     | NULL    |
| major   | char(4)      | YES  |     | NULL    |
| sID     | decimal(7,0) | NO   | PRI | NULL    |
| cNumber | char(8)      | NO   | PRI | NULL    |
| grade   | char(2)      | YES  |     | NULL    |
| cNumber | char(8)      | NO   | PRI | NULL    |
| cName   | varchar(30)  | NO   |     | NULL    |
| credits | decimal(1,0) | NO   |     | 3       |
+---------+--------------+------+-----+---------+
+-------------------------------+
| Constraints in the database   |
+--------------------+----------------+
| foreign key        | references     |
+--------------------+----------------+
| EnrolledIn.cNumber | Course.cNumber |
| EnrolledIn.sID     | Student.sID    |
+--------------------+----------------+
```

Figure 5.12   Screen - last part of output of script, ct_mtom.sql

Figure 5.12 shows the last part of the output of script, ct_mtom.sql, which gives the constraints.

5.8.2.6 Populate Tables

Use a script called, in_mtom.sql (in for INSERT INTO, mtom for many-to-many), to populate the tables with the instances in Tables 5.5, 5.6 and 5.7.

Start the editor and type the script.

MariaDB [webdb]> emacs in_mtom.sql

Script 5.4 Populate Student-EnrolledIn-Course: in_mtom.sql
(Start of script)

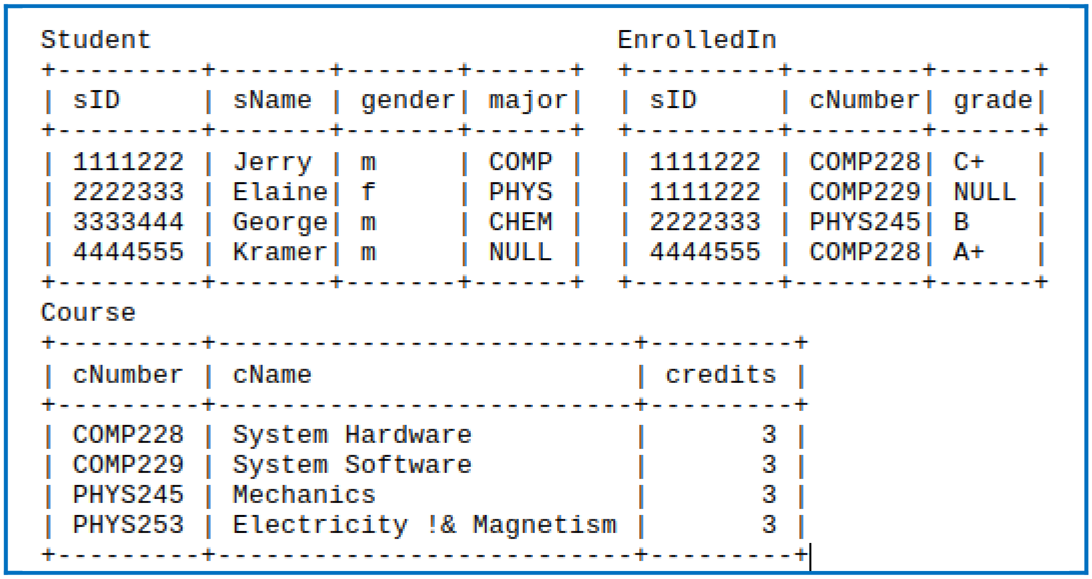
```
-- file name: in_mtom.sql
-- populate many-to-many relationship
-- there are three tables: Student-EnrolledIn-Course
-- two foreign keys in EnrolledIn: one references Student,
--                 one references Course
-- EnrolledIn is a child table, Student and Course are parent
tables
-- delete the contents of child tables first, then parent tables
DELETE FROM EnrolledIn;         -- child table
DELETE FROM Student;                  -- parent table
DELETE FROM Course;             -- parent table
-- populate parent tables first, then child tables
INSERT INTO Student      -- parent table
values(1111222, 'Jerry', 'm', 'COMP');
INSERT INTO Student
values(2222333, 'Elaine', 'f', 'PHYS');
INSERT INTO Student
values(3333444, 'George', 'm', 'CHEM');
INSERT INTO Student
values(4444555, 'Kramer', 'm', null);
INSERT INTO Course       -- parent table
VALUES('COMP228', 'System Hardware', 3);
INSERT INTO Course
VALUES('COMP229', 'System Software', 3);
INSERT INTO Course
VALUES('PHYS245', 'Mechanics', 3);
INSERT INTO Course
VALUES('PHYS253', 'Electricity & Magnetism', 3);
INSERT INTO EnrolledIn   -- child table
VALUES(1111222, 'COMP228', 'C+');
INSERT INTO EnrolledIn
VALUES(1111222, 'COMP229', null);
INSERT INTO EnrolledIn
VALUES(4444555, 'COMP228', 'A+');
INSERT INTO EnrolledIn
VALUES(2222333, 'PHYS245', 'B');
-- display the contents of the tables
select ""  AS "     Table: Student       ";
SELECT * FROM Student;
SELECT ""  AS "     Table: EnrolledIn    ";
SELECT * FROM EnrolledIn;
SELECT ""  "        Table: Course        ";
SELECT * FROM Course;
```

(End of the script)

Save the script and terminate the editor.
Execute the script to populate the tables.

MariaDB [webdb]> source in_mtom.sql

Figure 5.13 shows the last part of the output of script, in_mtom.sql.



```
Student                              EnrolledIn
+---------+-------+-------+------+  +---------+--------+------+
| sID     | sName | gender| major|  | sID     | cNumber| grade|
+---------+-------+-------+------+  +---------+--------+------+
| 1111222 | Jerry | m     | COMP |  | 1111222 | COMP228| C+   |
| 2222333 | Elaine| f     | PHYS |  | 1111222 | COMP229| NULL |
| 3333444 | George| m     | CHEM |  | 2222333 | PHYS245| B    |
| 4444555 | Kramer| m     | NULL |  | 4444555 | COMP228| A+   |
+---------+-------+-------+------+  +---------+--------+------+
Course
+---------+--------------------------+---------+
| cNumber | cName                    | credits |
+---------+--------------------------+---------+
| COMP228 | System Hardware          |       3 |
| COMP229 | System Software          |       3 |
| PHYS245 | Mechanics                |       3 |
| PHYS253 | Electricity !& Magnetism |       3 |
+---------+--------------------------+---------+
```

Figure 5.13 Screen - last part of output of script, in_mtom.sql

5.8.2.7 Integrity Constraints

The integrity constraints below are in the CREATE TABLE commands for Student, EnrolledIn and Course in the previous subsection.

(1) Student

PRIMARY KEY

```
CONSTRAINT  StudentPK PRIMARY KEY(sID)
```

The student ID, sID, must be different for every student, that is, every tuple has a different value of sID. Also, sID cannot be null; every tuple has a value for sID.

NOT NULL

```
sName  VARCHAR(20) NOT NULL
```

The name of the student, sName, must be given for every student, that is, every tuple has a value for sName.

```
gender  CHAR(1) NOT NULL CHECK (gender IN ('m','f'))
```

The gender of the student, gender, must be given for every student, that is, every tuple has a value for gender.

CHECK

The value of gender, gender, can only be m or f.

(2) EnrolledIn

PRIMARY KEY

```
CONSTRAINT  EnrolledInPK PRIMARY KEY(sID, cNumber)
```

The primary key is a composite primary key, meaning it consists of more than one attribute. The student ID, sID, and the course number, cNumber, together are the primary key. Every tuple must have a different pair of values of sID and cNumber. Also, sID and cNumber cannot be null; every tuple has a value for both sID and cNumber.

FOREIGN KEY

```
CONSTRAINT   EnrolledIn_StudFK  FOREIGN  KEY(sID)  REFERENCES
Student
CONSTRAINT  EnrolledIn_CourFK FOREIGN KEY(cNumber)
     REFERENCES Course
```

The first foreign key constraint connects the two tables, Student and EnrolledIn, and enforces the rule that a student with student, sID, who does not exist cannot enroll in a course. The second foreign key constraint connects the two tables, Course and EnrolledIn, and enforces the rule that a course with course number, cNumber, that does not exist cannot be taken by a student.

There are four consequences of the foreign key constraints: (a) a student who does not exist cannot be inserted in EnrolledIn, (b) a course that does not exist cannot be inserted in EnrolledIn, (c) A student who is enrolled in one or more courses cannot be deleted from table, Student, (d) a course cannot be deleted from table, Course, if one or more students are enrolled in the course.

The two foreign key constraints establish a many-to-many relationship between Student and Course. Student and Course are the parent tables and EnrolledIn is the child table of both parent tables.

Since both foreign keys by default are declared NULL (they are not declared, NOT NULL), a student can be enrolled in zero courses and a course can have an enrollment of zero. So the participation on both sides of EnrolledIn is optional (or partial).

(3) Course

PRIMARY KEY

```
CONSTRAINT  CoursePK PRIMARY KEY(cNumber)
```

The course number, cNumber, must be different for every course, that is, every tuple has a different value for cNumber. Also, cNumber cannot be null; every tuple has a value for cNumber.

NOT NULL

```
cName  VARCHAR(30) NOT NULL
```

The name of the course, cName, must be given for every course, that is, every tuple has a value for cName.

NOT NULL

```
credits DECIMAL(1) DEFAULT 3 NOT NULL CHECK (credits IN
(3,4))
```

The value of credits, credits, must be given for every course, that is, every tuple has a value for credits.

DEFAULT

The value of credits, credits, will be assigned the value of 3 if the value of credits is not given when the course is inserted into the database.

CHECK

The value of credits, credits, can only be 3 or 4.

5.8.2.8 Test Many-to-many Foreign Key Constraint

EnrolledIn

(1) insert a student who does not exist

Enroll the student with sID, 1212121, who does not exist in Student, into the course, COMP228, which does exist. This is rejected.

```
MariaDB [webdb]> INSERT INTO EnrolledIn
        VALUES(1212121, 'COMP228',null);
```

(2) insert a student who does exist into a course that does not exist

Enroll the student with sID, 1111222, who does exist, into the course, COMP238, which does not exist in Course. This is rejected.

```
MariaDB [webdb]> INSERT INTO EnrolledIn
        VALUES(1111222, 'COMP238',null);
```

(3) insert a student and course that does exist

Enroll the student with sID, 1111222, into the course, PHYS245. This is allowed.

```
MariaDB [webdb]> INSERT INTO EnrolledIn
        VALUES(1111222, 'PHYS245',null);
```

Student

(4) delete a student who is enrolled in a course

Delete the student with sID, 1111222, who is enrolled in a course. This is rejected.

```
MariaDB [webdb]> DELETE FROM Student
        WHERE sID = 1111222;
```

Course

(5) delete a course in which a student is enrolled

Delete the course, COMP228, in which students are enrolled. This is rejected.

```
MariaDB [webdb]> DELETE FROM Course
        WHERE cNumber = 'COMP228';
```

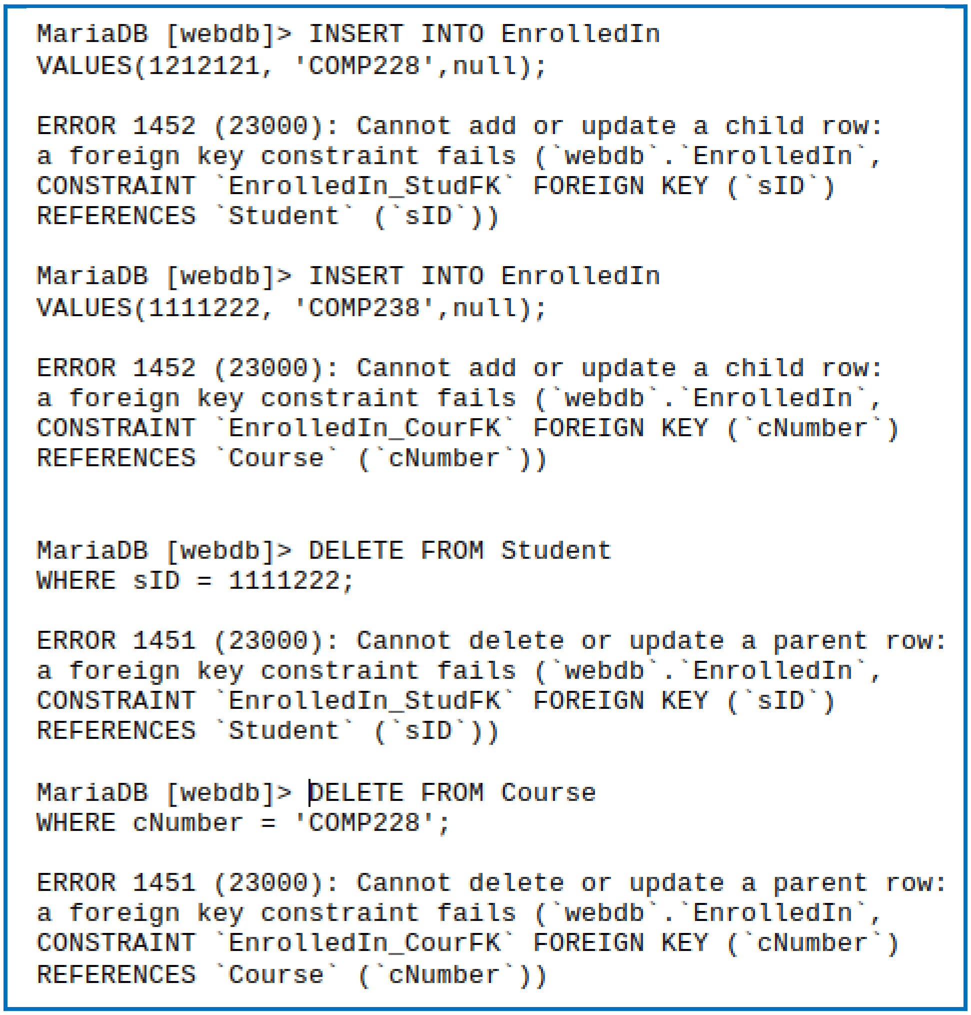
5.8.3 One-to-one: Department-Professor

5.8.3.1 Semantics

The entities, Department and Professor, are related by the relationship, ChairedBy. The ER diagram is given in Figure 5.15. A department is chaired (managed) by one professor. A department has one chairperson or no chairperson. A department has no chairperson, for example, if the chairperson resigns before another chairperson is chosen. A professor can be the chairperson of only one department or no department. This relationship is called one-to-one and the participation on both sides of the relationship is optional (or partial). This multiplicity, one-to-one, is indicated on the ER diagram by an arrow on both sides of the relationship.

Figure 5.14 shows the error messages from the INSERT INTO commands due to the two foreign key constraints.

Both the multiplicity and participation are a consequence of the semantics (meaning) of the entities and the relationship between them.



```
MariaDB [webdb]> INSERT INTO EnrolledIn
VALUES(1212121, 'COMP228',null);

ERROR 1452 (23000): Cannot add or update a child row:
a foreign key constraint fails (`webdb`.`EnrolledIn`,
CONSTRAINT `EnrolledIn_StudFK` FOREIGN KEY (`sID`)
REFERENCES `Student` (`sID`))

MariaDB [webdb]> INSERT INTO EnrolledIn
VALUES(1111222, 'COMP238',null);

ERROR 1452 (23000): Cannot add or update a child row:
a foreign key constraint fails (`webdb`.`EnrolledIn`,
CONSTRAINT `EnrolledIn_CourFK` FOREIGN KEY (`cNumber`)
REFERENCES `Course` (`cNumber`))


MariaDB [webdb]> DELETE FROM Student
WHERE sID = 1111222;

ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails (`webdb`.`EnrolledIn`,
CONSTRAINT `EnrolledIn_StudFK` FOREIGN KEY (`sID`)
REFERENCES `Student` (`sID`))

MariaDB [webdb]> DELETE FROM Course
WHERE cNumber = 'COMP228';

ERROR 1451 (23000): Cannot delete or update a parent row:
a foreign key constraint fails (`webdb`.`EnrolledIn`,
CONSTRAINT `EnrolledIn_CourFK` FOREIGN KEY (`cNumber`)
REFERENCES `Course` (`cNumber`))
```

Figure 5.14 Screen - Many-to-many, foreign key error messages

Multiplicity: One-to-one

each department is chaired by one professor (a department does not have joint chairpersons)

each professor chairs (manages) one department (a professor cannot chair more than one department)

Participation

each department is chaired by at most one professor (every department has one chairperson or no chairperson)

each professor chairs at most one department (not every professor is a chairperson)

If each department must have a chairperson at all times, then the participation of department for this relationship is total- every department is in one instance of this relationship. Whereas the participation of professor is partial: only some of the professors are chairs. Also each department is chaired by exactly one professor (every department has one chairperson). In this case, if a professor ceases to be chairperson unexpectedly, his or her name will remain in the database until another professor becomes chairperson.

In addition to multiplicity and participation, some of the other rules are the following. The constraints needed to enforce the rules are given in brackets.

- The rules for departments are given in subsection 5.8.1.1.
- Every professor has a numerical identification which is different from all other professors (primary key).
- The first and last name of every professor in the database must be recorded (not null).

5.8.3.2 ER Diagram

The ER diagram is shown in Figure 5.15.

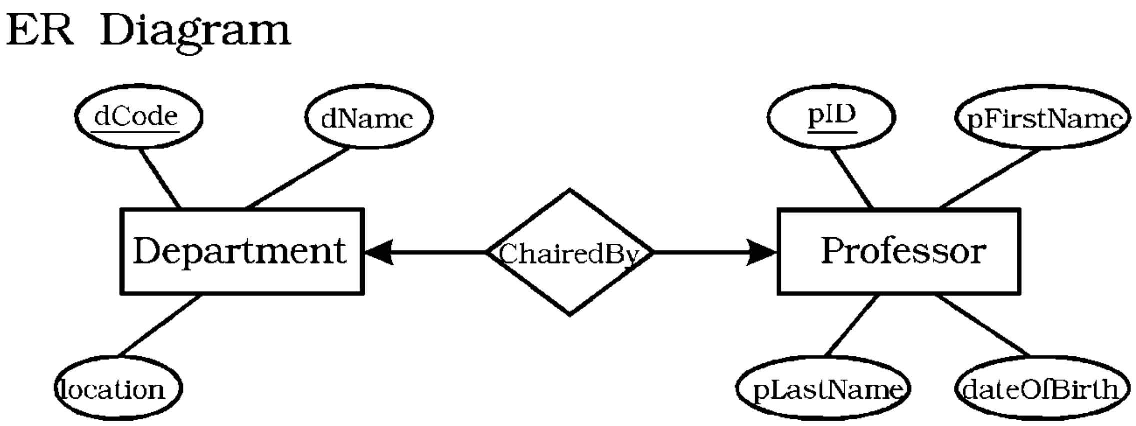


Figure 5.15 One-to-one: Department-Professor

5.8.3.3 Schema

There are four choices for the schema, from subsection 5.6.2.3.

The two schemas with two tables each are considered. They are named, Schema1 and Schema2, below. Schema1 is chosen as the schema for the relationship in Figure 5.15, as explained below.

The primary keys in each table are underlined.

(1) Schema1

Department(dCode, pID(fk → Professor, unique), dName, location) child
Professor(pID, pFirstName, pLastName, dateOfBirth) parent

The schema implies that Professor is the parent table and Department is the child table because the attribute, pID in Department is the primary key of Professor. This will be a one-to-one relationship if pID in Department is unique. A better name for pID in table, Department, would be dChair or dChairID.

(2) Schema2

Department(dCode, dName, location) parent
Professor(pID, dCode (fk → Department, unique), pFirstName, pLastName, dateOfBirth)

child

In this case, Department is the parent table and Professor is the child table because the attribute, dCode, in Professor is the primary key of Department. This will be a one-to-one relationship if dCode in Professor is unique. A better name for dCode in table, Professor, would be chairOfDept.

Schema1 is the better of the two choices because there are relatively few departments and relatively many professors, that is, there are few tuples in table, Department, compared to the number of tuples in table, Professor. In Schema1, there are relatively few values of foreign key, pID, and they all have a value (assuming all departments have a chair). In Schema2, there are many values of foreign key, dCode, and most are null (because most professors are not a chair).

Table 5.8 Department Instance2

dCode	pID	dName	location
10	123456	Computer Science	Library Building
11	345678	Physics	Science Building

Table 5.9 Professor Instance

pID	pFirstName	pLastName	dateOfBirth
123456	Jean	Charest	1958 June 24
234567	Bernard	Landry	1937 March 9
345678	Lucien	Bouchard	1938 December 22
456789	Jacques	Parizeau	1930 August 9
567890	Daniel	Johnson, Jr	

5.8.3.4 Instance

Instances of Department and Professor are given in Table 5.8 and Table 5.9.
The instance of Department, Table 5.8, is the same as Table 5.4 but with a foreign key. The instance in Table 5.8 is called Department Instance2 to distinguish it from Department Instance in Table 5.4.

5.8.3.5 Create Tables

Use a script called, ct_1to1.sql (ct for CREATE TABLE, 1to1 for one-to-one), to create the two tables in the schema with the constraints needed to enforce the rules described in the subsection, Semantics.
Start the editor and type the script.

```
MariaDB [webdb]> system emacs ct_1to1.sql;
```

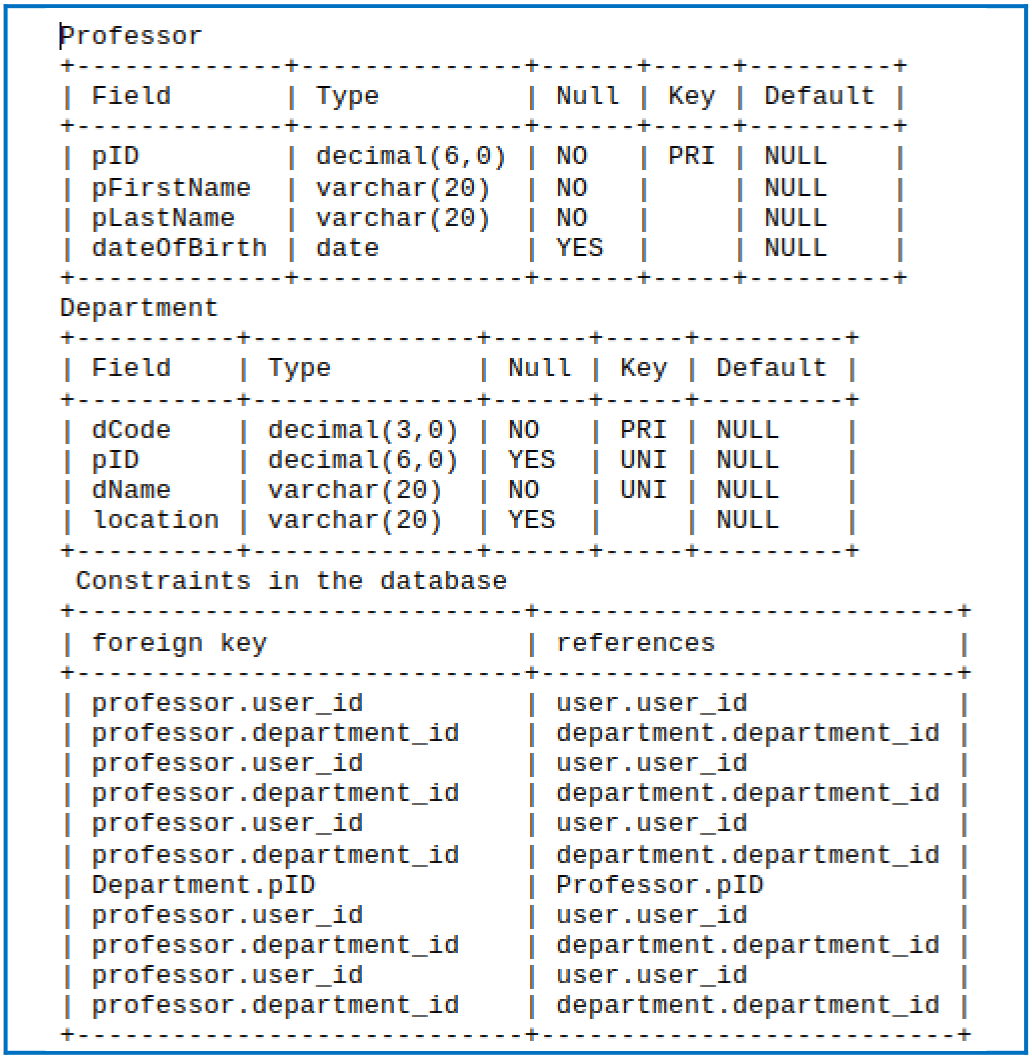
Script 5.5 Create Department-ChairedBy-Professor: ct_1to1.sql
(Start of script)

```
-- file name: ct_1to1.sql
-- create one-to-one relationship
-- there are two tables: Department-Professor
-- remove child tables first, then parent tables
DROP TABLE IF EXISTS Department;    -- child table
DROP TABLE IF EXISTS Professor;     -- parent table
-- create parent tables first and then child tables
CREATE TABLE Professor ( -- parent table
pID             DECIMAL(6),
pFirstName      VARCHAR(20) NOT NULL,
pLastName       VARCHAR(20) NOT NULL,
dateOfBirth     date,
CONSTRAINT      ProfPK PRIMARY KEY(pID));
CREATE TABLE Department (  -- child table
dCode           DECIMAL(3),
pID             DECIMAL(6),
dName           VARCHAR(20) NOT NULL,
location        VARCHAR(20),
CONSTRAINT      DeptPK PRIMARY KEY(dCode),
CONSTRAINT      DeptFK FOREIGN KEY (pID) REFERENCES
Professor(pID),
CONSTRAINT      DeptUK UNIQUE(pID),  -- UNIQUE establishes one-
to-one relationship
CONSTRAINT      DeptNameUK UNIQUE (dName));
-- display description of attributes of tables
select "" AS "   Specification for Professor      ";
DESC Professor;
select "" AS "   Specification for  Department    ";
DESC Department;
-- display constraints
select "" AS "   Constraints in the database  ";
select concat(table_name, '.', column_name) as 'foreign key',
    concat(referenced_table_name, '.', referenced_column_name) as
'references'
from information_schema.key_column_usage
where referenced_table_name is not null
    and table_name =  'Professor' OR table_name =  'Department
```

(End of the script)

Save the script and terminate the editor.
Execute the script to create the tables.

```
   MariaDB [webdb]> source ct_1to1.sql;
```



```
Professor
+-------------+--------------+------+-----+---------+
| Field       | Type         | Null | Key | Default |
+-------------+--------------+------+-----+---------+
| pID         | decimal(6,0) | NO   | PRI | NULL    |
| pFirstName  | varchar(20)  | NO   |     | NULL    |
| pLastName   | varchar(20)  | NO   |     | NULL    |
| dateOfBirth | date         | YES  |     | NULL    |
+-------------+--------------+------+-----+---------+
Department
+----------+--------------+------+-----+---------+
| Field    | Type         | Null | Key | Default |
+----------+--------------+------+-----+---------+
| dCode    | decimal(3,0) | NO   | PRI | NULL    |
| pID      | decimal(6,0) | YES  | UNI | NULL    |
| dName    | varchar(20)  | NO   | UNI | NULL    |
| location | varchar(20)  | YES  |     | NULL    |
+----------+--------------+------+-----+---------+
 Constraints in the database
+----------------------------+---------------------------+
| foreign key                | references                |
+----------------------------+---------------------------+
| professor.user_id          | user.user_id              |
| professor.department_id    | department.department_id  |
| professor.user_id          | user.user_id              |
| professor.department_id    | department.department_id  |
| professor.user_id          | user.user_id              |
| professor.department_id    | department.department_id  |
| Department.pID             | Professor.pID             |
| professor.user_id          | user.user_id              |
| professor.department_id    | department.department_id  |
| professor.user_id          | user.user_id              |
| professor.department_id    | department.department_id  |
+----------------------------+---------------------------+
```

Figure 5.16 Screen - last part of output of script, ct_1to1.sql

Figure 5.16 shows the last part of the output of script, ct_1to1.sql.

5.8.3.6 Populate Tables

Use a script called, in_1to1.sql (in for INSERT INTO, 1to1 for one-to-one), to populate the tables with the instances in Table 5.8 and Table 5.9.
Start the editor and type the script.

```
   MariaDB [webdb]> system emacs  in_1to1.sql;
```

Script 5.6 Populate Department-ChairedBy-Professor: in_1to1.sql
(Start of script)

```
-- file name: in_1to1.sql
-- populate one-to-one relationship
-- there are two tables: Department-Professor

-- delete the contents of child tables first, then parent tables
DELETE from Department;   -- child table
DELETE from Professor;    -- parent table

-- populate parent tables first, then child tables
INSERT INTO Professor
VALUES(123456, 'Jean', 'Charest', '1958-06-24');
INSERT INTO Professor
VALUES(234567, 'Bernard', 'Landry', '1937-03-09');
INSERT INTO Professor
VALUES(345678, 'Lucien', 'Bouchard', '1938-12-22');
INSERT INTO Professor
VALUES(456789, 'Jacques', 'Parizeau', '1930-08-09');
INSERT INTO Professor
VALUES(567890, 'Daniel', 'Johnson, Jr', null);
-- child table
INSERT INTO Department
values(10, 123456, 'Computer Science', 'Library Building');
INSERT INTO Department
VALUES(11, 345678, 'Physics', 'Science Building');

-- display the contents of the tables
select "" AS "   Contents of table: Professor     ";
SELECT * FROM Professor;
select "" AS "   Contents of table: Department     ";
SELECT * FROM Department;
```

(End of the script)

Save the script and terminate the editor.
Execute the script to populate the tables.

```
MariaDB [webdb]> source in_1to1.sql
```

Figure 5.17 shows the last part of the output of script, in_1to1.sql.

5.8.3.7 Integrity Constraints

The integrity constraints are in the CREATE TABLE commands for Professor and Department.

(1) Professor

PRIMARY KEY

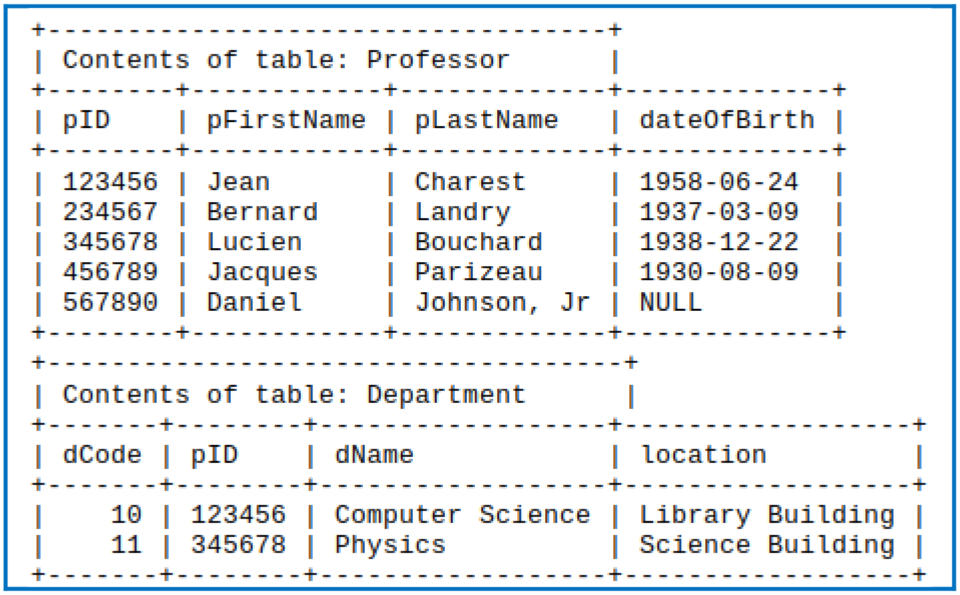
```
CONSTRAINT  ProfPK PRIMARY KEY(pID)
```

The professor identification, pID, must be different for every professor, that is, every tuple has a different value of pID. Also, pID cannot be null; every tuple has a value for pID.

NOT NULL

```
pFirstName     VARCHAR(20) NOT NULL
pLastName      VARCHAR (20) NOT NULL
```

Both the first name, pFirstName, and the last name, pLastName, must be given for every professor, that is, every tuple has values for pFirstName and pLastName.



```
+-------------------------------------+
| Contents of table: Professor        |
+--------+------------+-------------+-------------+
| pID    | pFirstName | pLastName   | dateOfBirth |
+--------+------------+-------------+-------------+
| 123456 | Jean       | Charest     | 1958-06-24  |
| 234567 | Bernard    | Landry      | 1937-03-09  |
| 345678 | Lucien     | Bouchard    | 1938-12-22  |
| 456789 | Jacques    | Parizeau    | 1930-08-09  |
| 567890 | Daniel     | Johnson, Jr | NULL        |
+--------+------------+-------------+-------------+
+--------------------------------------+
| Contents of table: Department        |
+-------+--------+------------------+------------------+
| dCode | pID    | dName            | location         |
+-------+--------+------------------+------------------+
|    10 | 123456 | Computer Science | Library Building |
|    11 | 345678 | Physics          | Science Building |
+-------+--------+------------------+------------------+
```

Figure 5.17 Screen - last part of output of script, in_1to1.sql

(2) Department

PRIMARY KEY

```
CONSTRAINT  DeptPK PRIMARY KEY(dCode)
```

The department code, dCode, must be different for every department, that is, every tuple has a different value of dCode. Also, dCode cannot be null; every tuple has a value for dCode.

NOT NULL

```
dName     VARCHAR(20) NOT NULL
```

The name of the department, dName, must be given for every department, that is, every tuple has a value for dName.

FOREIGN KEY

```
CONSTRAINT  DeptFK FOREIGN KEY(pID) REFERENCES Professor
```

The professor identification number, pID, which is the professor identification of the professor who is chair of the department, must exist in the table, Professor. The foreign key constraint connects the two tables, Department and Professor, and enforces the rule that a department cannot have a chairperson who is a professor that does not exist. There are two consequences of this: (a) a department cannot chaired by a professor that does not exist, and (b) a professor who is chairperson of department cannot be removed from the database.

Since there is only one tuple for each department, because the department code, dCode, is the primary key, there can only be one chair for each department. So this database does not allow a department to have joint chairs (two or more chairs). Since the foreign key by default is declared NULL (is not declared NOT NULL), a department can have no chairperson, which could occur, for example, if a chairperson resigns and is not yet replaced. So the participation on the Department side of the relationship is optional (or partial). Professor is the parent table and Department is the child table.

UNIQUE

```
CONSTRAINT  DeptProfUK UNIQUE(pID)
```

The professor identification number, pID, which is the professor identification of the professor who is chair of the department, must be different for every department. The unique constraint enforces the rule that a professor can be the chairperson of only one department. The foreign key constraint together with the unique constraint applied to pID establishes a one-to-one relationship between Department and Professor.

```
CONSTRAINT  DeptNameUK UNIQUE (dName)
```

The department name, dName, must be different for every department, that is, every tuple has a different value of dName.

5.8.3.8 Test One-to-one Foreign Key Constraint

First, create another department: Chemistry with department code, 12, located in the Science Building, but without a chairperson (pID is null). This is permitted because the foreign key by default is declared NULL (pID is not declared, NOT NULL).

```
MariaDB [webdb]> INSERT INTO Department
    VALUES(12, null, 'Chemistry', 'Science Building');
MariaDB [webdb]> SELECT * FROM Department;
```

Foreign key constraint

(1) chairperson of a department must be a professor who exists

Designate to Department, Chemistry, a chairperson who does not exist. This could happen, for example, when Bernard Landry becomes chairperson and by mistake his pID is typed incorrectly, 234568 instead of 234567. This is rejected.

```
MariaDB [webdb]> UPDATE Department
    SET pID = 234568
    WHERE dCode = 12;
```

(2) a professor who is chairperson of a department cannot be removed from the database

Delete from Professor the chairperson of Computer Science, who is Jean Charest with pID = 123456. This is rejected.

```
MariaDB [webdb]> DELETE FROM Professor
    WHERE pID = 123456;
```

Unique constraint

(3) a professor can be a chairperson of only one department

Designate to Chemistry a chairperson who is also chairperson of Physics. The chairperson of Physics is Lucien Bouchard who has pID = 345678. This is rejected.

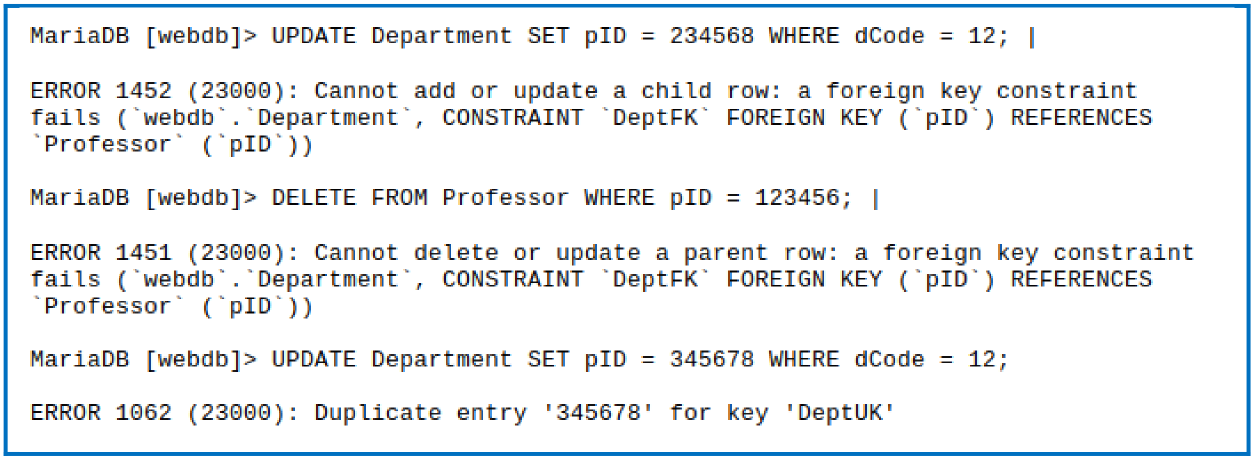
```
MariaDB [webdb]> UPDATE Department
    SET pID = 345678
    WHERE dCode = 12;
```

(4) a professor who exists can be chair of one department

Designate to Chemistry a chairperson who is professor that exists and who is not a chairperson of another department. The new chairperson of Chemistry is Bernard Landry who has pID = 234567. This is allowed.

```
MariaDB [webdb]> UPDATE Department
        SET pID = 234567
        WHERE dCode = 12;
MariaDB [webdb]> SELECT * FROM Department;
```

Figure 5.18 shows the foreign key error messages.



```
MariaDB [webdb]> UPDATE Department SET pID = 234568 WHERE dCode = 12; |

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
fails (`webdb`.`Department`, CONSTRAINT `DeptFK` FOREIGN KEY (`pID`) REFERENCES
`Professor` (`pID`))

MariaDB [webdb]> DELETE FROM Professor WHERE pID = 123456; |

ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint
fails (`webdb`.`Department`, CONSTRAINT `DeptFK` FOREIGN KEY (`pID`) REFERENCES
`Professor` (`pID`))

MariaDB [webdb]> UPDATE Department SET pID = 345678 WHERE dCode = 12;

ERROR 1062 (23000): Duplicate entry '345678' for key 'DeptUK'
```

Figure 5.18 Screen - One-to-one, foreign key error messages

5.8.4 Many-to-many: Course-Course

5.8.4.1 Semantics

The entity, Course, is related to itself by the relationship, IsPrerequisite. Prerequisite means something required or needed before something else. If course A must be taken before course B, then course A is a prerequisite for course B. Each course can have many other courses as prerequisites. Also each course can be a prerequisite for many other courses. Therefore, this relationship is many-to-many. The ER diagram is given in Figure 5.19. The multiplicity, many-to-many, is indicated on the ER diagram by no arrow on either side of the relationship.

The multiplicity, the upper limit of how many prerequisites each course can have and how many courses can have the same prerequisite, is not specified. A course in the final year of a program is likely to have more than one prerequisite, but is less likely to be a prerequisite for another course. A course in the first year of a program is likely to be a prerequisite for more than one other course, but is less likely to have a prerequisite.

The participation, the lower limit of the number of prerequisites, is specified. A course in the first term of the first year of a program is likely to have no prerequisite. So the lower limit of the number of prerequisites a course can have is 0; a course can have zero or more prerequisites. A course in the last term of the final year of a program is likely not to be a prerequisite for another course. So the lower limit of the number of courses for which a particular course is a prerequisite is also 0; a course can be a prerequisite for zero or more other courses. Therefore, the participation is optional (or partial) on both sides of the many-to-many relationship.

In addition to multiplicity and participation, some of the other rules are the following.
- The rules for courses are given in subsection 5.8.1.1.
- If course A is a prerequisite for course B then course B cannot be a prerequisite for course A.
- Also, a course cannot be a prerequisite for itself, that is, course A cannot be a prerequisite for course A. These are examples of two rules that cannot be enforced by the schema constraints (constraints in the CREATE TABLE command). These rules can be enforced only by application programming.

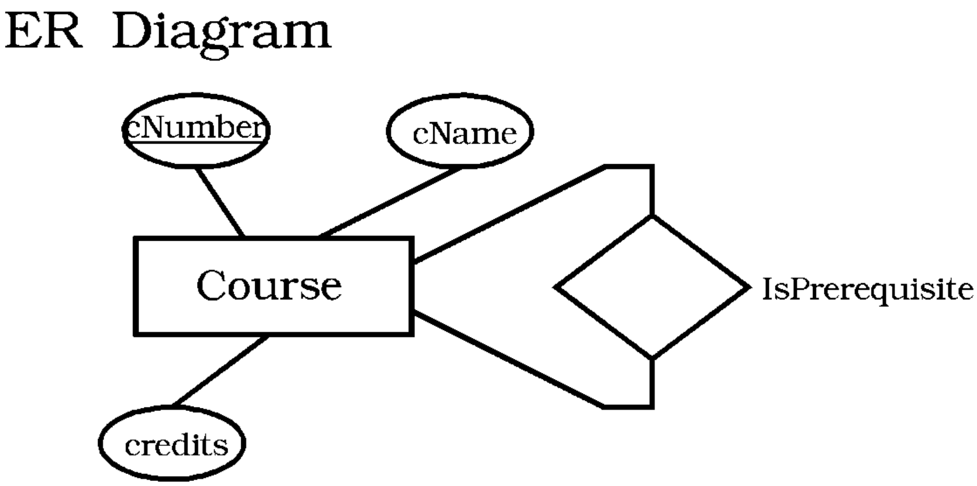


Figure 5.19 Many-to-many: Course-Course

5.8.4.2 ER Diagram

The ER diagram is shown in Figure 5.19.

5.8.4.3 Schema

There is only one choice for the schema, from subsection 5.6.2.1.

Course(<u>cNumber</u>, cName, credits)	parent
IsPrerequisite(<u>course</u>(fk ⇀ Course), <u>prerequisite</u>(fk ⇀ Course))	child

The primary keys in each table are underlined.

Since both attributes of IsPrerequisite, course and prerequisite, are foreign keys that reference the primary key of Course, cNumber, both of them cannot have the same name, cNumber. The two different names, course and prerequisite, are chosen for the foreign keys in IsPrerequisite so as to convey the meaning of the two attributes.

The data which gives the prerequisites of courses could be stored in the single table, Course, but there would be redundancy in the table, that is, the same data would be recorded in the table more than once, for example, the name of the course. To avoid redundancy, a second table is created which corresponds to the relationship, IsPrerequisite. Now there are two relationships, IsPrerequisite-Course, one for course has-prerequisite and one for course is-prerequisite. Both are many-to-one relationships where IsPrerequisite has the role of many in both relationships. Both many-to-one relationships are enforced by two foreign keys in the table, IsPrerequisite, where each foreign key references the same primary key of table, Course. The table with the foreign keys, IsPrerequisite, is called the child table and the table which is referenced by the foreign keys, Course, is called the parent table.

The two foreign keys in table, IsPrerequisite, are single attributes: a course and its prerequisite (also a course). The primary key of table, IsPrerequisite, is the two attributes together. The primary key is the combination of course and it's prerequisite simply to prevent listing the same prerequisite for a course more than once.

5.8.4.4 Instance

An instance of Course is the same as the instance in Table 5.7. An instance of IsPrerequisite is given in Table 5.10. Note that it is not realistic that a computer science course, COMP228, is a prerequisite for a physics course, PHYS245, second line of the table. However, this is included in Table 5.10 so as to have rows in the table with the same course, PHYS245, in both the course and prerequisite column, second and third rows of the table. So PHYS245 is a course which both has a prerequisite (second row) and is a prerequisite (third row).

5.8.4.5 Create Tables

Use a script called, ct_mtom2.sql (ct for CREATE TABLE, mtom2 for many-to-many2), to create the two tables in the schema with the constraints needed to enforce the rules described in the subsection, Semantics.

Start the editor and type the script.

```
MariaDB [webdb]> emacs ct_mtom2.sql
```

Table 5.10 IsPrerequisite Instance

course	prerequisite
COMP229	COMP228
PHYS245	COMP228
PHYS253	PHYS245

Script 5.7 Create Course-IsPrerequisite-Course: ct_mtom2.sql

(Start of script)

```
-- file name: ct_mtom2.sql
-- create many-to-many relationship
-- there are two tables: Course-IsPrerequisite
-- There are two foreign keys in IsPrerequisite: both reference
Course
-- IsPrerequisite is a child table, Course is a parent table
-- remove child tables first and then parent tables
DROP TABLE IF EXISTS IsPrerequisite;
DROP TABLE IF EXISTS EnrolledIN;
-- child table of Course from previous relationship
DROP TABLE IF EXISTS Course;    -- parent table
-- create parent tables first, then the child tables.
CREATE TABLE Course (
cNumber            CHAR(8),
cName       VARCHAR(30) NOT NULL,
credits            DECIMAL(1) DEFAULT 3 NOT NULL CHECK (credits
IN (3,4)),
CONSTRAINT  CoursePK PRIMARY KEY(cNumber));
CREATE TABLE IsPrerequisite (
course      CHAR(8),
prerequisite       CHAR(8),
CONSTRAINT  IsPrerequisitePK PRIMARY KEY(course, prerequisite),
CONSTRAINT  IsPrerequisite_courseFK FOREIGN KEY(course)
REFERENCES Course(cNumber),
CONSTRAINT  IsPrerequisite_prereqFK FOREIGN KEY(prerequisite)
REFERENCES Course(cNumber));
-- display description of attributes of tables
select "" AS "   Specification for Course      ";
DESC Course;
select "" AS "   Specification for IsPrerequisite     ";
DESC IsPrerequisite;
-- display constraints
select "" AS "   Constraints in the database  ";
select
    concat(table_name, '.', column_name) as 'foreign key',
    concat(referenced_table_name, '.', referenced_column_name) as
'references'
from   information_schema.key_column_usage
where  (referenced_table_name is not NULL
    or referenced_column_name is not NULL)
    and table_schema = 'webdb'
    and (table_name =   'COURSE' OR table_name =
'ISPREREQUISITE');
```

(End of the script)

Save the script and terminate the editor.
Execute the script to create the tables.

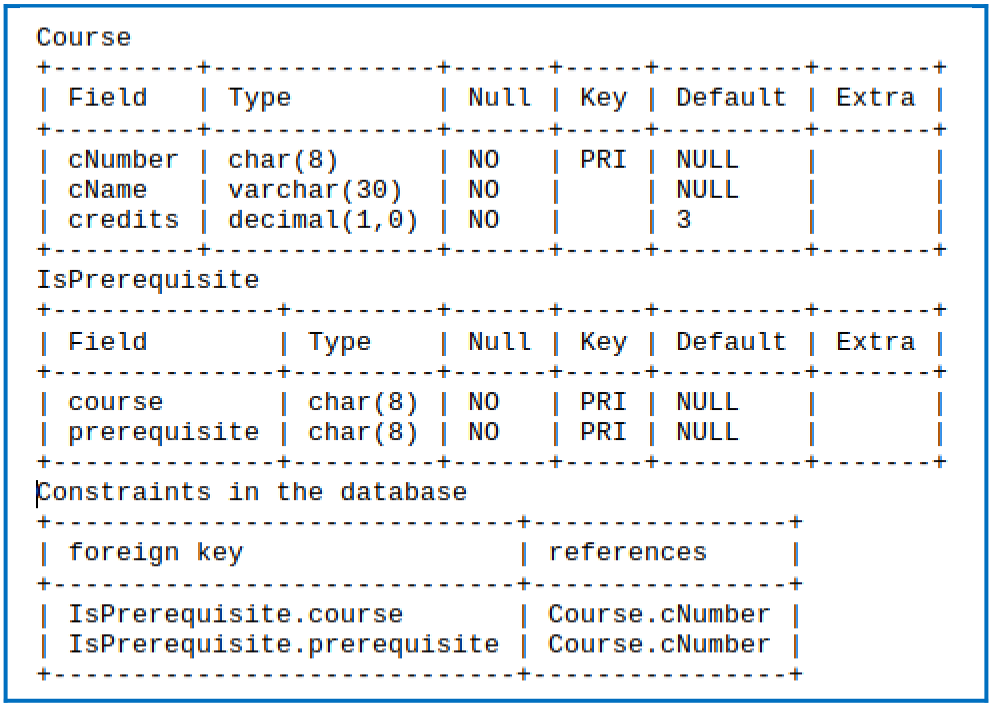
```
MariaDB [webdb]> source ct_mtom2.sql
```

Figure 5.20 shows the last part of the output of script, ct_mtom2.sql.

5.8.4.6 Populate Tables

Use a script called, in_mtom2.sql (in for INSERT INTO, mtom for many-to-many2), to populate the tables with the instances in Table 5.7 and Table 5.10.
Start the editor and type the script.

MariaDB [webdb]> emacs in_mtom2.sql



```
Course
+---------+--------------+------+-----+---------+-------+
| Field   | Type         | Null | Key | Default | Extra |
+---------+--------------+------+-----+---------+-------+
| cNumber | char(8)      | NO   | PRI | NULL    |       |
| cName   | varchar(30)  | NO   |     | NULL    |       |
| credits | decimal(1,0) | NO   |     | 3       |       |
+---------+--------------+------+-----+---------+-------+
IsPrerequisite
+--------------+---------+------+-----+---------+-------+
| Field        | Type    | Null | Key | Default | Extra |
+--------------+---------+------+-----+---------+-------+
| course       | char(8) | NO   | PRI | NULL    |       |
| prerequisite | char(8) | NO   | PRI | NULL    |       |
+--------------+---------+------+-----+---------+-------+
Constraints in the database
+-----------------------------+----------------+
| foreign key                 | references     |
+-----------------------------+----------------+
| IsPrerequisite.course       | Course.cNumber |
| IsPrerequisite.prerequisite | Course.cNumber |
+-----------------------------+----------------+
```

Figure 5.20 Screen - last part of output of script, ct_mtom2.sql

Script 5.8 Populate Course-IsPrerequisite-Course: in_mtom2.sql
(Start of script)

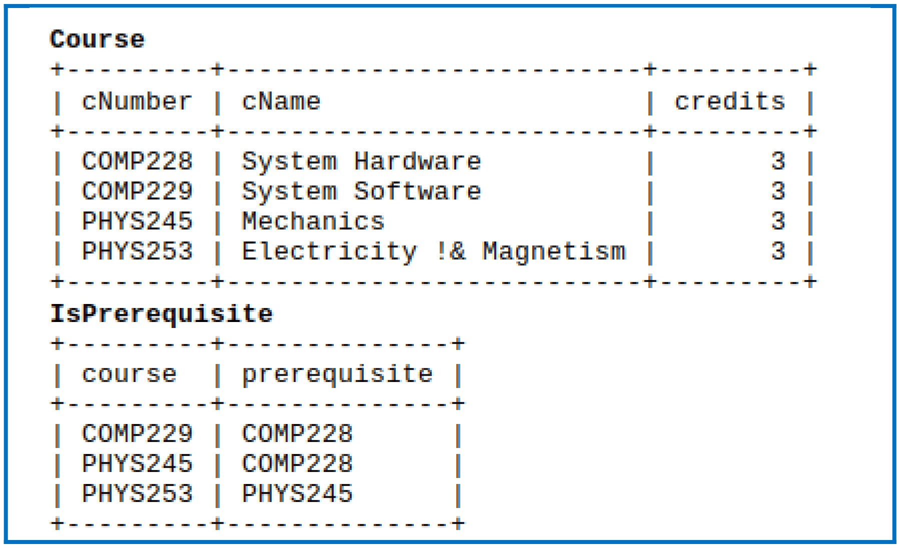
```
-- file name: in_mtom2.sql
-- populate many-to-many relationship
-- there are two tables: Course-IsPrerequisite
-- two foreign keys in IsPrerequisite: both reference Course
-- IsPrerequisite is a child table, Course is a parent table
-- delete the contents of child tables first, then parent tables
DELETE FROM IsPrerequisite;  -- child table
DELETE FROM Course;          -- parent table
-- populate parent tables first, then child tables
INSERT INTO Course  -- parent table
VALUES('COMP228', 'System Hardware', 3);
INSERT INTO Course
VALUES('COMP229', 'System Software', 3);
INSERT INTO Course
VALUES('PHYS245', 'Mechanics', 3);
INSERT INTO Course
VALUES('PHYS253', 'Electricity !& Magnetism', 3);
INSERT INTO IsPrerequisite   -- child table
VALUES('COMP229', 'COMP228');
INSERT INTO IsPrerequisite
VALUES('PHYS245', 'COMP228');
INSERT INTO IsPrerequisite
```

```
VALUES('PHYS253', 'PHYS245');
--display the contents of the tables
select "" AS "   Contents of table:   Course   ";
SELECT * FROM Course;
select "" AS "   Contents of table:   IsPrerequisite   ";
SELECT * FROM IsPrerequisite;
```

(End of the script)

Save the script and terminate the editor.
Execute the script to populate the tables.
MariaDB [webdb]> source in_mtom2.sql

Figure 5.21 shows the last part of the output of script, in_mtom2.sql.



```
Course
+---------+------------------------+---------+
| cNumber | cName                  | credits |
+---------+------------------------+---------+
| COMP228 | System Hardware        |       3 |
| COMP229 | System Software        |       3 |
| PHYS245 | Mechanics              |       3 |
| PHYS253 | Electricity !& Magnetism |       3 |
+---------+------------------------+---------+
IsPrerequisite
+---------+--------------+
| course  | prerequisite |
+---------+--------------+
| COMP229 | COMP228      |
| PHYS245 | COMP228      |
| PHYS253 | PHYS245      |
+---------+--------------+
```

Figure 5.21 Screen - last part of output of script, in_mtom2.sql

5.8.4.7 Integrity Constraints

The integrity constraints are in the CREATE TABLE commands for Course and IsPrerequisite.

(1) Course

PRIMARY KEY

```
CONSTRAINT  CoursePK PRIMARY KEY(cNumber)
```

The course number, cNumber, must be different for every course, that is, every tuple has a different value for cNumber. Also, cNumber cannot be null; every tuple has a value for cNumber.

NOT NULL

```
cName     VARCHAR(30) NOT NULL
```

The name of the course, cName, must be given for every course, that is, every tuple has a value for cName.

NOT NULL

```
Credits DECIMAL(1) DEFAULT 3 NOT NULL CHECK (credits IN
(3,4))
```

The value of credits, credits, must be given for every course, that is, every tuple has a value for credits.

DEFAULT

The value of credits, credits, is assigned the value of 3 if the value of credits is not given when the course is inserted into the database.

CHECK

The value of credits, credits, is only 3 or 4.

(2) IsPrerequisite

PRIMARY KEY

```
CONSTRAINT  IsPrerequisitePK PRIMARY KEY(course,
prerequisite)
```

The course number, course, and prerequisite number, prerequisite, must be different for every prerequisite of a course, that is, every tuple has a different combination of values of course and prerequisite. Also, course and prerequisite cannot be null; every tuple has a value for both course and prerequisite.

FOREIGN KEY

```
CONSTRAINT  IsPrerequisite_courseFK FOREIGN KEY(course)
     REFERENCES Course(cNumber)
CONSTRAINT  IsPrerequisite_prereqFK FOREIGN
     KEY(prerequisite) REFERENCES Course(cNumber)
```

Both foreign keys connect the tables, Course and IsPrerequisite. The first foreign key constraint enforces the rule that a course with number, cNumber that does not exist cannot have a prerequisite. The second foreign key constraint enforces the rule that a course with course number, cNumber that does not exist cannot be a prerequisite.

There are two consequences of the foreign key constraints: (a) a course that does not exist cannot be inserted in IsPrerequisite and (b) a course cannot be deleted from table, Course, which has a prerequisite or is a prerequisite, that is, when the course is also in table, IsPrerequisite.

The two foreign key constraints establish a many-to-many relationship between Course and itself. Course is the parent table and IsPrerequisite is the child table, as given in Figure 5.18.

Since both foreign keys by default are NULL (they are not declared, NOT NULL), a course can have zero prerequisites and a course can be zero prerequisites for other courses. So the participation on both sides of the IsPrerequisite relation is optional (or partial).

5.8.4.8 Test Many-to-many Foreign Key Constraint

IsPrerequisite

(1) insert a course which does not exist

Add course, COMP238, which does not exist and which has the prerequisite, COMP228. This is rejected.

```
MariaDB [webdb]> INSERT INTO IsPrerequisite
     VALUES('COMP238', 'COMP228');
```

(2) insert a prerequisite which does not exist
Add course, COMP238, which does not exist and which is a prerequisite for COMP228. This is rejected.

```
MariaDB [webdb]> INSERT INTO IsPrerequisite
        VALUES('COMP228', 'COMP238');
```

Course
(3) delete a course which has a prerequisite
Delete the course, PHYS253, which has the prerequisite, PHYS245. This is rejected.

```
MariaDB [webdb]> DELETE FROM Course
        WHERE cNumber = 'PHYS253';
```

(4) delete a course which is a prerequisite
Delete the course, COMP228, which is a prerequisite for, COMP229. This is rejected.

```
MariaDB [webdb]> DELETE FROM Course
        WHERE cNumber = 'COMP228';
```

Figure 5.22 shows the foreign key error messages.

5.8.5 Sample Database: Four Relationships

5.8.5.1 Semantics

The four relationships have been given in the four previous sections. The four relationships consist of six tables (relations). The semantics of the four relationships are described in the first subsection of the last four sections.

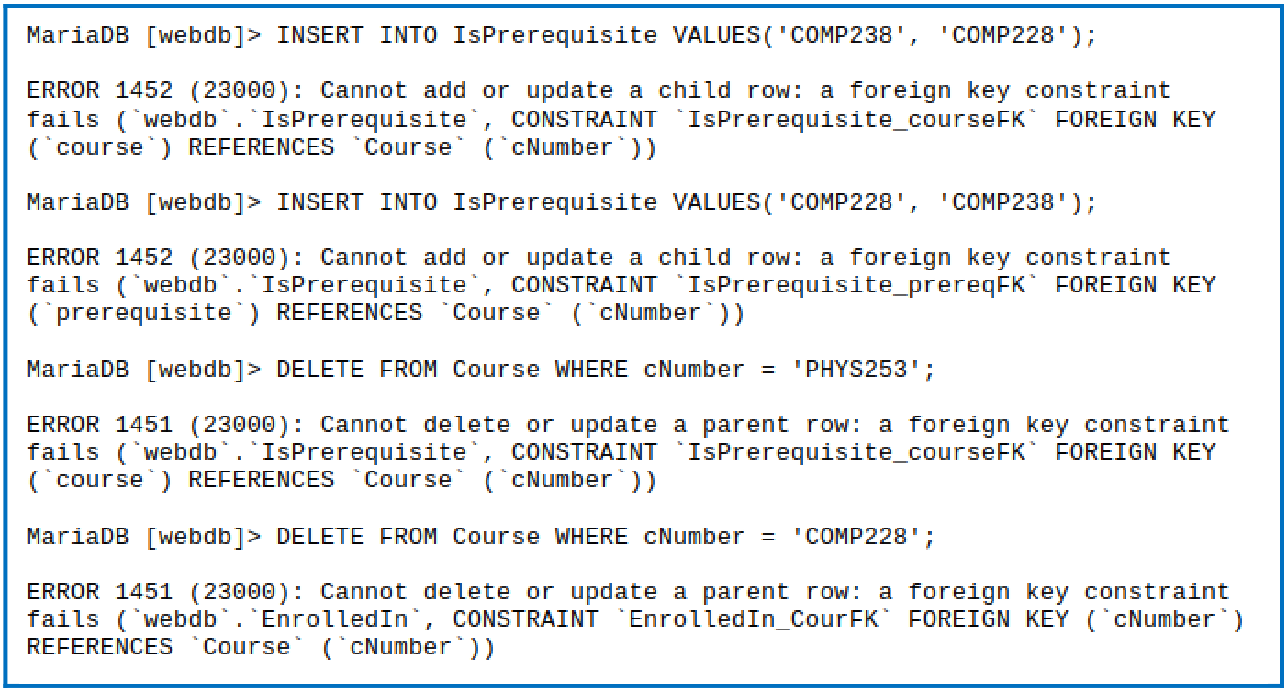
The entire database consisting of these relations is produced by two scripts, just as each relationship was produced. The first script creates the tables from the schema of the tables; this is the container of the database. The other script populates the tables from the instance of the tables; this is the contents of the database. The first script also removes the tables before it creates the tables in case the tables already exist. The second script also deletes the contents of the tables before it populates the tables in case the tables already contain data.

The removal of tables and the deletion of the contents of tables, as well as the creation and population of tables, must be done in a particular order, which depends on whether each table is a parent table, a child table or both. A table is a parent table if its primary key is referenced by foreign key in another table. A table is a child table if it has a foreign key.
Order of removal and deletion - child tables first, then parent tables.
Order of creation and population - parent tables first, then child tables.

The order of removal/deletion and creation/population of tables in the sample database is given in Table 5.11. Tables that are only children are removed/deleted first and created/populated last. Tables that are only parents are removed/deleted last and created/populated first. The tables that are both children and parents must be removed/deleted and created/populated in the middle of the sequence in an order that depends on how these tables are related to each other.

5.8.5.2 ER Diagram

The ER diagram is shown in Figure 5.6.



```
MariaDB [webdb]> INSERT INTO IsPrerequisite VALUES('COMP238', 'COMP228');

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
fails (`webdb`.`IsPrerequisite`, CONSTRAINT `IsPrerequisite_courseFK` FOREIGN KEY
(`course`) REFERENCES `Course` (`cNumber`))

MariaDB [webdb]> INSERT INTO IsPrerequisite VALUES('COMP228', 'COMP238');

ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
fails (`webdb`.`IsPrerequisite`, CONSTRAINT `IsPrerequisite_prereqFK` FOREIGN KEY
(`prerequisite`) REFERENCES `Course` (`cNumber`))

MariaDB [webdb]> DELETE FROM Course WHERE cNumber = 'PHYS253';

ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint
fails (`webdb`.`IsPrerequisite`, CONSTRAINT `IsPrerequisite_courseFK` FOREIGN KEY
(`course`) REFERENCES `Course` (`cNumber`))

MariaDB [webdb]> DELETE FROM Course WHERE cNumber = 'COMP228';

ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint
fails (`webdb`.`EnrolledIn`, CONSTRAINT `EnrolledIn_CourFK` FOREIGN KEY (`cNumber`)
REFERENCES `Course` (`cNumber`))
```

Figure 5.22 Screen - many-to-many Course-Course, foreign key error messages

5.8.5.3 Schema

The schema of the database is the schemas for the individual relationships in the last four sections.

5.8.5.4 Instance

The instances of the database are given in Tables 5.3, 5.5, 5.6, 5.8, 5.9, 5.10.

5.8.5.5 Create Tables

Use a script called, ct_db.sql (ct for CREATE TABLE, db for database), to create the database in Figure 5.6 with the constraints needed to enforce the rules described in the previous four sections.

Start the editor and type the script.

```
MariaDB [webdb]> system emacs  ct_db.sql;
```

Script 5.9 Create Entire Database: ct_db.sql
(Start of script)

```
-- file name: ct_db.sql
-- create database with six tables
-- remove child tables first, then parent tables
DROP TABLE IF EXISTS EnrolledIn;       -- child table
DROP TABLE IF EXISTS IsPrerequisite;   -- child table
DROP TABLE IF EXISTS Course;          -- child table, parent table
DROP TABLE IF EXISTS Department;      -- child table, parent table
DROP TABLE IF EXISTS Student;          -- parent table
DROP TABLE IF EXISTS Professor;        -- parent table
```

Table 5.11 Order of Tables

Table	Parent of	Child of	Order of removal and deletion	Order of creation and population
Student	EnrolledIn		5	1 (first)
EnrolledIn		Student, Course	1 (first)	5
Course	EnrolledIn, IsPrerequisite	Department	3	4
Department	Course	Professor	4	3
Professor	Department		6	2
IsPrerequisite		Course	2	6

```
-- create parent tables first, then child tables
CREATE TABLE Student (   -- parent table
sID          DECIMAL(7),
sName        VARCHAR(20) NOT NULL,
gender       CHAR(1) NOT NULL,
major        CHAR(4),
CONSTRAINT  StudentPK PRIMARY KEY(sID));

CREATE TABLE Professor (    -- parent table
pID          DECIMAL(6),
pFirstName   VARCHAR(20) NOT NULL,
pLastName    VARCHAR(20) NOT NULL,
dateOfBirth  date,
CONSTRAINT  ProfPK PRIMARY KEY(pID));

CREATE TABLE Department (  -- parent table, child table
dCode        DECIMAL(3),
pID          DECIMAL(6),
dName        VARCHAR(30) NOT NULL,
location     VARCHAR(30),
CONSTRAINT  DepartmentPK PRIMARY KEY(dCode),
CONSTRAINT  DeptFK FOREIGN KEY(pID) REFERENCES Professor(pID),
CONSTRAINT   DeptUK UNIQUE(pID),    -- UNIQUE establishes one-to-
one relationship
CONSTRAINT DepartmentNameUK UNIQUE (dName));

CREATE TABLE Course (  -- parent table, child table
cNumber      CHAR(8),
dCode        DECIMAL(3),
cName        VARCHAR(30) NOT NULL,
credits      DECIMAL(1) DEFAULT 3 NOT NULL CHECK (credits IN
(3,4)),
CONSTRAINT  CoursePK PRIMARY KEY(cNumber),
CONSTRAINT  CourseFK FOREIGN KEY(dCode) REFERENCES
Department(dCode) ,
CONSTRAINT  CourseNameUK UNIQUE (cName, dCode));

CREATE TABLE EnrolledIn (  -- child table
sID          DECIMAL(7),
cNumber      CHAR(8),
grade        CHAR(2),
CONSTRAINT  EnrolledInPK PRIMARY KEY(sID, cNumber),
CONSTRAINT  EnrolledIn_StudFK FOREIGN KEY(sID) REFERENCES
Student(sID),
CONSTRAINT  EnrolledIn_CourFK FOREIGN KEY(cNumber) REFERENCES
Course(cNumber));

CREATE TABLE IsPrerequisite (  -- child table
course       CHAR(8),
prerequisite CHAR(8),
CONSTRAINT  IsPrerequisitePK PRIMARY KEY(course, prerequisite),
CONSTRAINT  IsPrerequisite_courseFK FOREIGN KEY(course)
REFERENCES Course(cNumber),
CONSTRAINT  IsPrerequisite_prereqFK FOREIGN KEY(prerequisite)
REFERENCES Course(cNumber));

-- display description of attributes of tables
select "" AS "   Contents of table:   Course   ";
DESC Course;
select "" AS "   Contents of table:   Department  ";
DESC Department;
select "" AS "   Contents of table:   Student  ";
DESC Student;
select "" AS "   Contents of table:   Professor  ";
DESC Professor;
select "" AS "   Contents of table:   EnrolledIn  ";
DESC EnrolledIn;
select "" AS "   Contents of table:   IsPrerequisite  ";
DESC IsPrerequisite;
```

(End of the script)

Save the script and terminate the editor.
Execute the script to create the tables.

```
MariaDB [webdb]> source ct_db.sql;
```

Display a list of all tables in the database.

```
MariaDB [webdb]> SHOW tables;
```

5.8.5.6 Populate Tables

Use a script called, in_db.sql (in for INSERT INTO, db for database), to populate the six tables of the database.

Start the editor and type the script.

```
MariaDB [webdb]> emacs in_db.sql
```

Script 5.10 Populate Entire Database: in_db.sql
(Start of script)

```
-- file name: in_db.sql
-- populate the six tables of the database
-- delete the contents of child tables first, then parent tables
DELETE from EnrolledIn;         -- child table
DELETE from IsPrerequisite;     -- child table
DELETE from Course;             -- child table, parent table
DELETE from Department;         -- child table, parent table
DELETE from Student;            -- parent table
DELETE from Professor;          -- parent table
-- populate parent tables first, then child tables

-- Student - parent table
INSERT INTO Student
VALUES(1111222, 'Jerry', 'm', 'COMP');
INSERT INTO Student
VALUES(2222333, 'Elaine', 'f', 'PHYS');
INSERT INTO Student
VALUES(3333444, 'George', 'm', 'CHEM');
INSERT INTO Student
VALUES(4444555, 'Kramer', 'm', null);

-- Professor - parent table
INSERT INTO Professor
VALUES(123456, 'Jean', 'Charest', '1958-06-22');
INSERT INTO Professor
VALUES(234567, 'Bernard', 'Landry', '1937-03-13');
INSERT INTO Professor
VALUES(345678, 'Lucien', 'Bouchard', '1938-12-22');
INSERT INTO Professor
VALUES(456789, 'Jacques', 'Parizeau', '1930-08-09');
INSERT INTO Professor
VALUES(567890, 'Daniel', 'Johnson, Jr', null);

-- Department - parent table and child table
INSERT INTO Department
VALUES(10, 123456, 'Computer Science', 'Library Building');
```

```
INSERT INTO Department
VALUES(11, 345678, 'Physics', 'Science Building');

-- Course - parent table and child table
INSERT INTO Course
VALUES('COMP228', 10, 'System Hardware', 3);
INSERT INTO Course
VALUES('COMP229', 10, 'System Software', 3);
INSERT INTO Course
VALUES('PHYS245', 11, 'Mechanics', 3);
INSERT INTO Course
VALUES('PHYS253', 11, 'Electricity !& Magnetism', 3);

-- EnrolledIn - child table
INSERT INTO EnrolledIn
VALUES(1111222, 'COMP228', 'C+');
INSERT INTO EnrolledIn
VALUES(1111222, 'COMP229', null);
INSERT INTO EnrolledIn
VALUES(4444555, 'COMP228', 'A+');
INSERT INTO EnrolledIn
VALUES(2222333, 'PHYS245', 'B');

-- IsPrerequisite - child table
INSERT INTO IsPrerequisite
VALUES('COMP229', 'COMP228');
INSERT INTO IsPrerequisite
VALUES('PHYS245', 'COMP228');
INSERT INTO IsPrerequisite
VALUES('PHYS253', 'PHYS245');

--display the contents of the tables
select "" AS "   Contents of table: Course      ";
SELECT * FROM Course;
select "" AS "   Contents of table: Department     ";
SELECT * FROM Department;
select "" AS "   Contents of table: Student     ";
SELECT * FROM Student;
select "" AS "   Contents of table: Professor     ";
SELECT * FROM Professor;
select "" AS "   Contents of table: EnrolledIn     ";
SELECT * FROM EnrolledIn;
select "" AS "   Contents of table: IsPrerequisite     ";
SELECT * FROM IsPrerequisite;
```

(End of the script)

Save the script and terminate the editor.
Execute the script to populate the tables.

```
MariaDB [webdb]> source in_db.sql;
```

5.8.5.7 Integrity Constraints

The integrity constraints are in the CREATE TABLE commands for the six tables. They have been described in the previous four subsections.

5.8.5.8 Test Foreign Key Constraint

The tests of all foreign key constraints have been demonstrated in the previous four subsections.

6 SQL/PSM

6.1 Introduction

SQL is a database access and manipulation language but it is nonprocedural and does not contain control structures. SQL/PSM (Structured Query Language/Persistent Stored Modules) is an ISO standard which extends SQL with a procedural programming language to be used for storing procedures in the relational database system. It includes all of the database access statements of SQL and all of the constructs of a modern programming language. In particular, SQL/PSM provides the control structures for conditional control (if) and iterative control (loop, while, for). SQL/PSM is completely integrated with SQL. SQL commands can be executed in a SQL/PSM function and SQL/PSM functions can be called within a SQL command.

SQL/PSM is a formal name used in the SQL standard maintained by ANSI (America National Standards Institute). In MySQL, SQL/PSM is called Stored Programs. The genesis of SQL/PSM was Oracle's PL/SQL, the procedural programming language by Oracle. PL/SQL includes all of the constructs of a modern programming language but does not confirm to the ANSI standards. The SQL_MODE was introduced in MariaDB 10.3. Setting the sql_mode system variable to `Oracle` in a block allows the database server to understand a subset of Oracle's PL/SQL language.

In this chapter features in SQL/PSM are discussed as well as some features in PL/SQL. The implementation of SQL/PSM in MySQL/MariaDB is closest to the ANSI standards.

6.2 Block

A block is a unit of a SQL/PSM program which contains one or more instructions. A program in PL/SQL consists of one or more blocks. There are three kinds of blocks:

(1) anonymous (meaning unnamed by the programmer) this is not implemented in MySQL/ MariaDB but could be used with the command: SET sql_mode='oracle';
(2) procedure
(3) function

In addition there are blocks called Triggers and Scheduled Events which are discussed later in this chapter. Triggers automate operation when a table in the database is modified. Scheduled Events is a block that is executed at a pre-determined time.

The procedure and function blocks must have a name. Data can be passed to both procedure and function blocks by a parameter list enclosed in parentheses, ( ), after the block name. Data is returned from a function; it is mandatory that a function returns a value. On the other hand a stored procedure could have three types of parameters and can pass multiple values to and from the procedure. The parameter can be an IN parameter for input and is the default type. An INOUT parameter is used to pass a value to and from the procedure. An OUT parameter is used to pass a value from the procedure. Data cannot be either passed to or returned from an anonymous block, because it does not have a name. Information about the three types of blocks is summarized in the Table 6.1.

6.2.1 Structure of Blocks

All three kinds of blocks have the same structure. There are four parts to a block.

(1) header part - mandatory for procedure and function, absent in anonymous block
(2) declaration part - optional
(3) execution part - mandatory
(4) exception part - optional

Table 6.1 Blocks in SQL/PSM

Property	Type of block		
	Anonymous	Procedure	Function
named		x	x
precompiled		x	x
passed values		x	x
return value		x	x

Note that in Oracle documentation the header part is called the specification and the rest of the block, parts (2), (3) and (4), is called the body.

The header part is used only for named blocks (procedure and function) and gives the name of the block and arguments, if any, passed to the block. The header for a function also gives the datatype of the return value. The declaration part defines variables, cursors and sub-blocks. The variables are local, that is, they do not exist outside the block. The execution part contains program instructions. The exception part contains error instructions, instructions that are executed if an error occurs.

The syntax for the three kinds of blocks follows. The keywords are in uppercase. The items in square brackets are optional. The name of the part of the block is in parentheses, ( ), which is not part of the syntax.

Syntax consists of the rules for the formation of the grammatical constructs in a language and must be learned by using the rules (by practicing) in a computer language just as is one learns the rules in a spoken language. So the syntax which follows is for reference purposes only and should be consulted when writing blocks until the syntax is memorized.

6.2.2 Syntax for anonymous blocks

```
[DECLARE                              (declaration)
    local declarations]
  BEGIN                               (execution)
    executable statements
  [EXCEPTION                          (exception)
    exception handlers]
  END;
```

Only one or more executable statements are required.

6.2.3 Syntax for procedures

```
PROCEDURE name [(parameter[, parameter, ...])]             (header)
    IS                                                     (declaration)
      [local declarations]
    BEGIN                                                  (execution)
      executable statements
    [EXCEPTION                                             (exception)
      exception handlers]
    END [name];
```

Only the name of the procedure and one or more executable statements are required.

6.2.4 Syntax for functions

```
FUNCTION name [(parameter[, parameter, ...])] RETURN datatype    (header)
    IS                                                     (declaration)
      [local declarations]
    BEGIN                                                  (execution)
      executable statements
    [EXCEPTION                                             exception)
      exception handlers]
    END [name];
```

Only the name of the function, datatype of the return value and one or more executable statements are required.

6.3 Use of SQL

SQL/PSM blocks can be executed from the command line of SQL.

A SQL/PSM block is input by typing in the statement making up the block at the SQL prompt. But they are not executed in the same way. To execute a SQL/PSM block, a special delimiter must be typed on the next line by itself after the last line of the block, and then Enter is pressed. In contrast, an SQL command is executed by a semicolon (;) at the end of the command or on the next line after the command, and then Enter is pressed. Since the semicolon is used to end each statement, redefine the SQL statement terminator with the delimiter command and at the end of the block execute the delimiter command to restore the semicolon(;) as the SQL command terminator**.**

Both the last-entered SQL/PSM block and last-entered SQL command are stored in the buffer and could be edited using the edit command with the editor declared in the user's environment.

6.4 Execution of Blocks using SQL

All three kinds of SQL/PSM blocks can also be created with a text editor and saved with a filename that has the extension, .sql, just like a SQL script. All blocks can be executed using SQL, where the commands are typed at the command line of SQL. Also, procedures and functions can be executed by being called in other procedures or functions, and by being referenced in a SQL command. The prompt of SQL used here is, MariaDB [webdb]>.

6.4.1 Anonymous block

An anonymous block does not have a block name and is executed just like a script using SQL. Consider the anonymous block :

```
DELIMITER $$
begin
SELECT  count(* ) AS Number_of_Course FROM  Course;
end$$
DELIMITER ;
```

Suppose this block is stored in a file named, mdb-anonblock.sql. It is executed by the command:

```
   MariaDB [webdb]> source mdb-anonblock.sql;
```

Loading an anonymous block causes its immediate execution; for the above source the result is:

```
MariaDB [webdb]> begin
    -> SELECT  count(* ) AS Number_of_Course FROM  Course;
    -> end$$
```

```
+------------------+
| Number_of_Course |
+------------------+
|                4 |
+------------------+
```

The following is a file containing an anonymous block, mdb-anonblock1.sql. Here the Oracle sql_mode is used for this anonymous block. Each time such a block is 'executed' it is automatically compiled first. The SQL delimiter character semicolon (;) is changed in the first statement to $$ so the semicolon could be used in the block to terminate a statement.

```
DELIMITER $$
SET sql_mode='oracle';
declare notuples int(4);
begin
SELECT  count(* ) AS notuples FROM  Course;
end$$
DELIMITER ;
```

The output of the block, is followed by resetting the SQL delimiter character to a semicolon.

```
MariaDB [webdb]> begin
    -> SELECT  count(* ) AS notuples FROM  Course;
    -> end$$
DELIMITER ;
```

```
+----------+
| notuples |
+----------+
|        4 |
+----------+
```

6.4.2 Procedure

A block which is a procedure is not executed like a script using SQL. Consider the following example of a procedure block:

```
DELIMITER $$
create or replace Procedure Procblock(OUT numtuples INT)
begin
select count(*)
into numtuples
from Person;
end$$
DELIMITER ;
```

If the procedure block is stored in the file, procblock.sql (note the name of the file need not be the same as the name of the procedure), it is first compiled and stored in the database by the command:

```
        MariaDB [webdb]> source procblock.sql
```

This is the same command that would execute a SQL script stored in a file or an anonymous block. For a procedure block the source command just compiles and stores the code in the database to be used later. The procedures stored in the database can be displayed with the following command:

```
        MariaDB [webdb]> SHOW PROCEDURE STATUS;
```

Subsequently. the procedure block can be executed by the command:

```
        MariaDB [webdb]> call procblock(@numtuples)
```

The result can be displayed using the code:

```
        MariaDB [webdb]> select @numtuples;
                             +------------+
                             | @numtuples |
                             +------------+
                             |          4 |
                             +------------+
```

Instead of writing a procedure to output the number of rows of each table in the database, a procedure that uses a prepared statement could be used as follows.

```
        DELIMITER $$
        create or replace FUNCTION dynprocblock(in tname 
varchar(20))
        begin
        SET @t1 =CONCAT('SELECT count(*)  FROM ',tname );
        PREPARE stmt FROM @t1;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
        end$$
        DELIMITER ;
```

Store this code in the file, dyn-proc-block.sql, and load it in the database with the source statement. Subsequently, the number of tuples in any table can be retrieved using the comand as shown below.

```
   MariaDB [webdb]> call dynprocblock('Person');
   MariaDB [webdb]> call dynprocblock('Course');
```

6.4.3 Function

A block which is a function is also compiled and executed like a block which is a procedure using SQL.

A function block with no parameters follows

```
        DELIMITER $$
        CREATE OR REPLACE FUNCTION funcblock()
        RETURN DEC(5,0)  AS numtuples DEC(5,0);
        -- note the specification of the return value DUPLICATED
                BEGIN
                SELECT count(*) INTO numtuples FROM Course;
                RETURN numtuples;
                END$$
        DELIMITER ;
```

If the function block is stored in the file, funcblock.sql. It is compiled by the command:

```
     MariaDB [webdb]> source funcblock.sql
```

The functions stored in the database can be displayed with the following command:

```
        MariaDB [webdb]> SHOW FUNCTION STATUS;
```

Subsequently, the function block can be executed by the command:

```
     MariaDB [webdb]> select  funcblock();
```

The compiled procedure and function blocks are stored in the database separately from the source files, procblock.sql and funcblock.sql. The source files can be deleted which is not advisable, since they are a record and the blocks can still be executed. To delete the compiled procedure and function blocks from the database use the commands:

```
     MariaDB [webdb]> DROP PROCEDURE procblock;
     MariaDB [webdb]> DROP FUNCTION funcblock;
```

To find the number of courses with a certain number of credits, write the function NoCourseWithCredits.sql as follows:

```
DELIMITER $$
create Function NoCourseWithCredits(crscredits decimal(1))
RETURNS dec(5)
DETERMINISTIC
BEGIN
```

```
DECLARE numtuples dec(5);
select count(*) into numtuples
        from Course
        where credits = crscredits;
return(numtuples);
end$$
DELIMITER ;
```

Before loading this function in the database, verify its status by using the show function status command as follows.

```
        MariaDB [webdb]> show function status;
        Empty set (0.001 sec)
```

Store it in the database by storing the function code in a file and loading it in the database with the following command. If there are no errors it will be accepted.

```
        MariaDB [webdb]> source NoCourseWithCredits.sql;
        Query OK, 0 rows affected (0.009 sec)
```

Verify the status again as follows:

```
        MariaDB [webdb]> show function status;
```

The code of the function and the status of the course table can be retrieved using the commands

```
  MariaDB [webdb]>  show  CREATE FUNCTION NoCourseWithCredits;
  MariaDB [webdb]> select * from Course;
              +---------+-------------------------+---------+
              | cNumber | cName                   | credits |
              +---------+-------------------------+---------+
              | COMP228 | System Hardware         |       3 |
              | COMP229 | System Software         |       3 |
              | PHYS245 | Mechanics               |       3 |
              | PHYS253 | Electricity & Magnetism |       3 |
              +---------+-------------------------+---------+
                     4 rows in set (0.000 sec)
```

Use the function by a select statement as follows.

```
MariaDB [webdb]> select NoCourseWithCredits(2);
                        +------------------------+
                        | NoCourseWithCredits(2) |
                        +------------------------+
                        |                      0 |
                        +------------------------+
MariaDB [webdb]> select NOCourseWithCredits(3);
                        +------------------------+
                        | NOCourseWithCredits(3) |
                        +------------------------+
                        |                      4 |
                        +------------------------+
```

6.5 Package

A package is a module in which one or more named blocks (procedures and functions) are stored. A package is intended for the storage of logically related procedures and functions, but any named block can be stored in any package. There are two parts of a package which must be created, each in its own file:

(a) specification
(b) body

The specification contains the declaration of the procedures and functions, and the body contains their definitions.

6.5.1 **Names**

The package specification and package body names are the names of the files in which the specification and body are stored. Typically, the file name of a particular package specification is, pack1, and the file name of the corresponding package body is, pack1b. However, the more verbose file names, pack1_spec and pack1_body, are used below to emphasize the contents of the two parts of the package.

6.5.2 **Syntax**

In the syntax below, optional keywords are in square brackets. If the keywords, OR REPLACE, are included, the contents of an existing file with the same name will be replaced when the specification or body is saved.

Syntax of package specification
CREATE [OR REPLACE] PACKAGE *package_specification_name*
AS
procedure declarations
function declarations
END *package_specification_name*;

Syntax of package body
CREATE [OR REPLACE] PACKAGE BODY *package_specification_name*
AS
procedure definitions
function definitions
END *package_specification_name*;

6.5.3 **Compilation**

Both the specification file and body file must be compiled. If the specification of package1 is stored in file, pack1_spec.sql and the body of package1 is stored in file, pack1_body.sql, the specification and body are compiled by commands,

```
MariaDB [webdb]> source pack1_spec
MariaDB [webdb]> source pack1_body
```

6.5.4 **Execution**

A procedure or function in a package is executed by adding the name of the package specification followed by a dot (.) as a prefix to the name of the procedure or function.

Syntax: EXEC *package_specification_name.block_name*

Assume that the package, pack1_spec, contains the procedure, procblock, and the function, funcblock.

The procedure is executed by SQL with the command

```
   MariaDB [webdb]> call pack1_spec.procblock
```

The function is executed by SQL using SELECT with the command

```
   MariaDB [webdb]>  select pack1_spec.funcblock (parameters);
```

6.5.5 Deletion

The compiled package specification and package body are stored separately from the source files for the specification and body. The compiled files are stored with the same name, but without the extension, .sql, and so, pack1_spec and pack1_body, are the compiled files of source files, pack1_spec.sql and pack1_body.sql, respectively.

The package body stored in file, pack1_body, is deleted by the command,

```
     MariaDB [webdb]> DROP PACKAGE BODY pack1_body;
```

Both the package body, pack1_body, and package specification, pack1_spec, are deleted by the command

```
     MariaDB [webdb]> DROP PACKAGE pack1_spec;
```

6.5.6 Package Example

Consider the file, package-spec-only.sql, contents of which are shown below. It is loaded into the database with the command

```
        MariaDB [webdb]> source  package-spec-only.sql;
```

```
SET sql_mode=ORACLE;
DELIMITER $$
CREATE OR REPLACE PACKAGE Course_tools AS
  FUNCTION NumbCourses() RETURN DEC(5,0);
  FUNCTION CourseName(cNumber char(8)) RETURN Varchar(30);
  PROCEDURE ChangeCredits(Nmbr char(8), Crdts decimal(1,0));
      -- NOTE: no IN, OUT INOUT
  PROCEDURE ChangeName(Nmbr char(8), Nom Varchar(30));
      -- NOTE: no IN, OUT INOUT
END;
$$
DELIMITER ;
```

We can see creation of this package by using the following command:

```
MariaDB [webdb]> SHOW CREATE PACKAGE Course_tools\G
*************************** 1. row ***************************
             Package: Course_tools
            sql_mode:
PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ORACLE,NO_KEY_OPTIONS,NO
```

```
_TABLE_OPTIONS,NO_FIELD_OPTIONS,NO_AUTO_CREATE_USER,SIMULTANEOUS_
ASSIGNMENT
      Create Package: CREATE DEFINER="webdba"@"localhost" PACKAGE
"Course_tools" AS
  FUNCTION NumbCourses() RETURN DEC(5,0);
  FUNCTION CourseName(cNumber char(8)) RETURN Varchar(30);
  PROCEDURE ChangeCredits(Nmbr char(8), Crdts decimal(1,0));
  PROCEDURE ChangeName(Nmbr char(8), Nom Varchar(30));
END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
1 row in set (0.001 sec)
```

The body of the package is loaded and after a procedure can be executed with a call statement as follows.

```
MariaDB [webdb]> call Course_tools.ChangeCredits('COMP399', 1);
```

Each of the blocks in a package can always be loaded separately and called directly as follows

```
        MariaDB [webdb]> call ChangeCredits('COMP399', 1);
        MariaDB [webdb]> drop package  Course_tools;
        Query OK, 0 rows affected (0.021 sec)
        MariaDB [webdb]> SHOW CREATE PACKAGE Course_tools\G
        ERROR 1305 (42000): PACKAGE Course_tools does not exist
```

6.6 Trigger

SQL provides features to express integrity constraints as part of the database schema. Constraints, in essence, provide database designers with more control over the database content.

An active element is a statement that is written once, stored in the database, and executed when an event occurs. This event is called a trigger. An event may be an insertion of a tuple into a predefined table or a specified change in the database that causes a specified (boolean-valued) condition to become true. It is also possible to implement many of the constraints and triggers in scripts such as PHP, and JSP. However, this is left to the application programmer and has to be included in each application!

A trigger is a procedure that is implicitly executed when an INSERT, UPDATE, or DELETE statement is issued against an associated table. A simple procedure is explicitly executed by a user, application, or a trigger. A trigger can restrict DML operations against a table and hence could provide enforcement of constraints in addition to that supported by the database model.

A statement in a trigger body could cause another trigger to be fired. Such triggers are said to be cascading triggers. Thus, a trigger is a named program unit that executes an anonymous block to perform a task when a specified event changes the data in the database. Triggers are used to enforce constraints that cannot be enforced by the schema constraints of the tables. For example, triggers can enforce referential integrity constraints that cannot be enforced by foreign keys. Commonly, DBMS's support the options, ON DELETE CASCADE and ON DELETE SET NULL, in the declaration of a foreign key, but not support ON UPDATE CASCADE and ON UPDATE SET NULL. So triggers must be used to control the updates of tuples in parent tables which are referenced by tuples in child tables.

6.6.1 Types of triggers

A trigger can be used if a required constraint cannot be enforced using the built in DBMS integrity constraints Also triggers enforce referential integrity when child and parent tables are on different nodes of a distributed database. Finally triggers enforce complex business rules not definable using the integrity constraints.

There are five types of triggers in terms of the database events which can cause triggers to be executed.

(1) DML commands: INSERT, UPDATE, DELETE (but not SELECT)
DML commands (excluding SELECT) change data in a table. A DML triggers is the most common type of trigger and is usually the only type used by program developers.

(2) DDL commands: CREATE TABLE, ALTER INDEX, DROP TRIGGER
DDL commands create or modify tables and indexes. The DDL triggers are used primarily by database administrators.

(3) Database events: database starts or stops, user logs on or logs off, error occurs
The purpose of these triggers is to record activity in the database.

(4) INSTEAD OF: replace DML events on views
The INSTEAD OF trigger does not depend on the type of event because it replaces the event (the operation insert, update or delete which caused the trigger to execute).

(5) Suspended statements: enter a suspend mode due to a problem
If there is a space problem, this trigger will suspend a statement until the problem is fixed.

Only the DML triggers will be covered in this chapter.

6.6.2 Syntax of DML triggers

```
CREATE [OR REPLACE] trigger_name
{BEFORE | AFTER}
{INSERT | DELETE | UPDATE | UPDATE OF column_list} ON table_name
[FOR EACH ROW]
[WHEN (. . .) ]
[DECLARE . . . ]
BEGIN
        executable statements
[EXCEPTION . . . ]
END [trigger_name];
```

6.6.3 Categories of DML triggers

Triggers can be classified into Row and Statement triggers. Each can be of type before or after an event.

Row Triggers (before and after)
A row trigger is fired once for each row affected by, say, an UPDATE statement. Row triggers are used when the trigger action depends on data provided by the triggering statement or rows that are affected.

Statement Triggers (before and after)
A statement trigger is fired once, regardless of the number of rows in the table that the triggering statement affects (even if no rows are affected). If a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once, regardless of the number of rows deleted from the table. This is useful when the trigger action does not depend on the data provided by the triggering statement or the rows that are affected.

The general protocol for executing triggers is as follows.

- SQL statement is issued
- Execute any BEFORE statement-level triggers
- For each row affected by the triggering SQL statement
    - Execute any BEFORE row-level triggers
    - Lock and change row, and perform integrity constraint checking
    - The lock is not released until the transaction is committed
- Execute any AFTER row-level triggers
- Execute any AFTER statement-level triggers

From the syntax of a DML trigger, there are 12 categories of DML triggers.
(a) There are three events (actions caused by statements) which modify a table: insert, update and delete.
Keywords: INSERT or DELETE or UPDATE or UPDATE OF
(b) The trigger is executed either before or after the event occurs.
Keywords: BEFORE or AFTER
(c) The trigger is executed either for each row which is affected (row-level) by the statement or only once for the whole statement (statement-level). For example, if an insert statement affects all the rows of a table, a row-level trigger executes once for each row of the table, but the statement-level trigger executes only one time.
Keywords: FOR EACH ROW for row-level, or no keyword for statement-level

A trigger responds to an event which is one of three statements (3), either before or after the event (2) and at one of two levels (2). Therefore there are 3 x 2 x 2 = 12 categories of DML triggers.

6.6.4 Parts of a trigger

There are three parts to the definition of a trigger: (i) event, (ii) condition and (iii) action.
(i) The event specifies one of the 12 categories of triggers. The event fires (starts the execution of) the trigger.
(ii) The condition starts with the keyword, WHEN. The condition determines whether the action should be executed or not. The condition evaluates to true (execute action) or false (do not

execute action). The condition is optional. If the condition is absent, the action is always executed.
(iii) The action is the operation which is executed. The action is a PL/SQL anonymous block, which usually contains SQL statements and can contain calls to procedures. When the action is executed for the case of before the event (BEFORE INSERT, BEFORE UPDATE, BEFORE DELETE), the action does not replace the event. For both before and after the event, the action can call the function, raise_application_error(), which aborts (prevents the occurrence of) the event and displays a message.

6.6.5 **Compile a trigger**

The source code of trigger is stored in a file with a name that has the extension, .sql, just as for the file names of procedures, functions and packages. A trigger must be compiled before it is used. It is compiled with the same command as for the compilation of procedures, functions and packages.

The compiled trigger is stored separately from the source code of the trigger. The compiled trigger has the same name as the file name of the source code of the trigger except without the extension, .sql. So if the source code of a trigger is stored in a file with name, trigger1.sql, the compiled trigger has name, trigger1.

The trigger with file name, trigger1.sql, is compiled by the command

```
   MariaDB [webdb]> source trigger1.sql
```

Note that it is not necessary to type the extension, .sql, of the trigger file name to compile the trigger.
The compiled trigger, trigger1, is deleted by the command

```
   MariaDB [webdb]> DROP TRIGGER trigger1;
```

6.6.6 **Enable and disable a trigger**

Triggers can be enabled and disabled. When a trigger is created it is enabled by default. When an enabled trigger is not needed it can be disabled instead of deleted in case it is needed later.
A trigger is disabled by the command

```
   MariaDB [webdb]> ALTER TRIGGER triggername DISABLE;
```

A trigger is enabled by the command

```
   MariaDB [webdb]>ALTER TRIGGER triggername ENABLE;
```

All triggers for a table are disabled by the command

```
   MariaDB [webdb]> ALTER TABLE tablename DISABLE ALL TRIGGERS;
```

All triggers for a table are enabled by the command

```
   MariaDB [webdb]> ALTER TABLE tablename ENABLE ALL TRIGGERS;
```

The column, status, in the system table, user_triggers, has the value, enabled, for triggers that are enabled and has the value, disabled, for triggers that are disabled. Display the values of status of triggers as follows.

```
MariaDB [webdb]> SELECT trigger_name, status FROM user_triggers;
```

Examples

Trigger 1: In this example of a before-insert trigger, the user inserting a tuple and the date of the insert is also recorded in the user table. The code of the trigger, stored in a file before-insert-users-trg.sql, is shown.

```
     DELIMITER |
     CREATE TRIGGER users_before_insert
     BEFORE INSERT
        ON users FOR EACH ROW
     BEGIN
        DECLARE whoinserted varchar(50);
        -- Find username of person performing inserting a new
user
        SELECT USER() INTO whoinserted ;
        -- Update create_date field to current system date
        SET NEW.insert_date = SYSDATE();
        -- Update created_by field to the username of the person
        --      performing the INSERT
        SET NEW.inserted_by = whoinserted;
     END; |
     DELIMITER ;

        MariaDB [webdb]>  source before-insert-users-trg.sql;
        Query OK, 0 rows affected (0.033 sec)
```

This is followed by an insert statement:

```
MariaDB [webdb]> insert into users(last_name, first_name, email)
                values('Parker', 'Arlo', 'ap@localhost');
        Query OK, 1 row affected (0.024 sec)

MariaDB [webdb]> select * from users where email like 'ap%';
+------+-------_-+----------+------------+-----------+----------------+
|userid|last_name|first_name|email       |insert_date|inserted_by     |
+------+---------+----------+------------+-----------+----------------+
| 34838|Parker   |Arlo      |ap@localhost|2020-03-08 |webdba@localhost|
+------+---------+----------+------------+-----------+----------------+
```

Trigger 2: This example illustrates a database constraint based on a business rule. This falls outside the model of the database and hence needs to be enforced by a trigger, in this case a before-insert trigger.

Consider the convention that no database engineer is allowed to supervise more than 4 projects in an organization. Suppose there are two tables as follows.

```
     create table DBEngineer(
      EID DECIMAL(4),
      SIN DECIMAL(9),
      Name char(20),
      HireAge DECIMAL(2) CHECK (HireAge BETWEEN 25 AND 65),
      CONSTRAINT Engineer_PK PRIMARY KEY(eid),
      CONSTRAINT Engineer_CK UNIQUE(SIN) );
     create table DBProjLeader(
      ProjNo DECIMAL (4) primary key,
      EID  DECIMAL (4),
      FOREIGN KEY (EID) REFERENCES Engineer(EID));

     MariaDB [webdb]> insert into  DBEngineer values(11,
     123456789, 'Smith', 35);
     Query OK, 1 row affected (0.014 sec)
     MariaDB [webdb]> insert into  DBEngineer values(12,
     234567891, 'Ellison', 73);
     ERROR 4025 (23000): CONSTRAINT `DBEngineer.HireAge` failed
     for `webdb`.`DBEngineer`
     MariaDB [webdb]> insert into  DBEngineer values(12,
     234567891, 'Ellison', 33);
     Query OK, 1 row affected (0.013 sec)

     DELIMITER $$
     Create Trigger NumberOfProjs BEFORE Insert on DBProjLeader
     FOR EACH ROW
     BEGIN
     Declare Howmany DECIMAL(2);
     Select count(ProjNo) into Howmany
     from DBProjLeader P
     where EID = NEW.EID;
     if ( Howmany > 3) then
     SIGNAL SQLSTATE '52000' SET MYSQL_ERRNO = 5001, MESSAGE_TEXT
     = 'Too many project for this DB engineer!';
     END IF;
     end$$
     DELIMITER ;

     MariaDB [webdb]> insert into  DBProjLeader values(1,11),
     (2,11),(3,11),(4,11) ;
     Query OK, 4 rows affected (0.064 sec)
     MariaDB [webdb]>  insert into  DBProjLeader values(5, 11);
     ERROR 5001 (52000): Too many project for this DB engineer!
     MariaDB [webdb]> select * from DBProjLeader;
                           +--------+------+
                           | ProjNo | EID  |
                           +--------+------+
                           |      1 |   11 |
                           |      2 |   11 |
                           |      3 |   11 |
                           |      4 |   11 |
                           +--------+------+
```

The trigger did not allow the insertion of the fifth project to the same database engineer!

Trigger 3: This example illustrates a database constraint based on setting a status of a person based on income and at the same time illustrate what is called a mutating trigger. The constraint falls outside the model of the database and hence needs to be enforced by a trigger, in this case a before-row insert trigger.
A mutating trigger when fired by a DML statement on a table, tries to use the same table. Hence the table is mutating.

Create a new table, person, with the person status based on income. The value of status is as shown in the trigger person_st given under and assumed to be stored in a file, tr_person_st.sql

```
create table person (name char(25) primary key,
dob date,
income decimal(9,2),
status char (6));

delimiter $$
CREATE  TRIGGER person_st
BEFORE INSERT ON person
FOR EACH ROW
BEGIN
DECLARE STATUS CHAR(4);
SET STATUS = 'Poor';
IF (new.income > 50000) THEN
SET STATUS = 'Med'; END IF;
IF (new.income > 100000)THEN
SET STATUS = 'Rich'; END IF;
INSERT INTO person VALUES(new.name, new.dob, new.income,
STATUS);
END$$
delimiter ;


MariaDB [webdb]> source tr_person_st.sql;
Query OK, 0 rows affected (0.035 sec)
```

Now insert a tuple in person which fires the trigger. The database finds that it is a mutating trigger.

```
insert into person (name, dob,income)
             values('Jones', '1968-06-08', 61000.00);
ERROR 1442 (HY000): Can't update table 'person' in stored
function/trigger because it is already used by statement
which invoked this stored function/trigger
```

Trigger 4: To avoid the problem of a mutating trigger, use a different temporary table (person1) for insertion. Insert the data in this temporary table which fires a trigger that inserts the data with the correct status in the main table (person).

```
create table person (name char(25) primary key,
dob date,
income decimal(9,2),
status char (6));
create table person1 (name char(25) primary key,
dob date,
income decimal(9,2));
delimiter $$
CREATE  TRIGGER person1_st
BEFORE INSERT ON person1
FOR EACH ROW
BEGIN
DECLARE STATUS CHAR(4);
SET STATUS = 'Poor';
IF (new.income > 50000) THEN
SET STATUS = 'Med'; END IF;
IF (new.income > 100000)THEN
SET STATUS = 'Rich'; END IF;
INSERT INTO person VALUES(new.name, new.dob,
           new.income, STATUS);
END$$
delimiter ;

MariaDB [webdb]>insert into person1 (name, dob,income)
              values('Jones', '1968-06-08', 61000.00);

MariaDB [webdb]> select * from person1;
             +-------+------------+----------+
             | name  | dob        | income   |
             +-------+------------+----------+
             | Jones | 1968-06-08 | 61000.00 |
             +-------+------------+----------+

MariaDB [webdb]> select * from person;
         +-------+------------+----------+--------+
         | name  | dob        | income   | status |
         +-------+------------+----------+--------+
         | Jones | 1968-06-08 | 61000.00 | Med    |
         +---------+-------------+-----------+---------+
```

Trigger 5: This example illustrates checking of data being entered for validity before inserting in the database table, PERSON_CONTACT. The trigger and its applications, both valid and invalid are illustrated below.

```
CREATE TABLE PERSON_CONTACT(
Pid   INT NOT NULL AUTO_INCREMENT,
Name varchar(25),
email VARCHAR(50),
PRIMARY KEY (Pid));
DELIMITER $$
```

```
CREATE OR REPLACE TRIGGER PERSON_CONTACT_CHECK BEFORE INSERT
ON PERSON_CONTACT
FOR EACH ROW
BEGIN
IF INSTR(NEW.email,'@') = 0 THEN
SIGNAL SQLSTATE '50002'
SET  MYSQL_ERRNO  =  5002,  MESSAGE_TEXT  =  'invalid  email
address';
END IF;
END$$
Query OK, 0 rows affected (0.042 sec)
DELIMITER ;
  MariaDB [webdb]> Query OK, 0 rows affected (0.042 sec)

MariaDB [webdb]> insert into PERSON_CONTACT(Name, email)
values('Jones', 'jones@some.where');
Query OK, 1 row affected (0.012 sec)

MariaDB  [webdb]>  insert  into  PERSON_CONTACT(Name,  email)
values('Smith', 'smith.some.where');
ERROR 5002 (50002): invalid email address
```

The content of the database is verified to ascertain that the trigger worked.

```
MariaDB [webdb]> select * from PERSON_CONTACT;
+-----+-------+------------------+
| Pid | Name  | email            |
+-----+-------+------------------+
|   1 | Jones | jones@some.where |
+-----+-------+------------------+
```

The above trigger is an after-row trigger. Since the database is not updated until the end of the trigger, even if an after-row trigger is used, the changes would not be propagated to the database. This is illustrated next.

Trigger 6: This example illustrates the checking of data being entered for validity after inserting in the database table, PERSON_CONTACT. The trigger and its applications, both valid and invalid are illustrated below.

```
CREATE TABLE PERSON_CONTACT1(
Pid   INT NOT NULL AUTO_INCREMENT,
Name varchar(25),
email VARCHAR(50),
PRIMARY KEY (Pid));

DELIMITER $$
CREATE OR REPLACE TRIGGER PERSON_CONTACT1_CHECK AFTER INSERT
ON PERSON_CONTACT1
```

```
FOR EACH ROW
BEGIN
IF INSTR(NEW.email,'@') = 0 THEN
SIGNAL SQLSTATE '50002'
SET MYSQL_ERRNO = 5002, MESSAGE_TEXT = 'invalid email
address';
END IF;
END$$
Query OK, 0 rows affected (0.042 sec)
DELIMITER ;
  MariaDB [webdb]> Query OK, 0 rows affected (0.042 sec)

MariaDB [webdb]> insert into PERSON_CONTACT1(Name, email)
values('Jones', 'jones@some.where');
Query OK, 1 row affected (0.012 sec)

MariaDB [webdb]> insert into PERSON_CONTACT1(Name, email)
values('Smith', 'smith.some.where');
ERROR 5002 (50002): invalid email address
```

The content of the database is verified to ascertain that the trigger worked. The database did not write data to storage until the statement and all triggers are finished.

```
MariaDB [webdb]> select * from PERSON_CONTACT1;
+-----+-------+------------------+
| Pid | Name  | email            |
+-----+-------+------------------+
|   1 | Jones | jones@some.where |
+-----+-------+------------------+
```

Trigger 7: This example illustrates the checking of data being updated and the recording of a time stamp for all inserts and updates to the database table, PERSON_CONTACT2 .

```
CREATE TABLE PERSON_CONTACT2(
Pid   INT NOT NULL AUTO_INCREMENT,
Name varchar(25),
email VARCHAR(50),
logday DATE, logtime TIME,
PRIMARY KEY (Pid));

DELIMITER $$
CREATE OR REPLACE TRIGGER PERSON_CONTACT2_CHECK BEFORE
INSERT ON PERSON_CONTACT2
FOR EACH ROW
BEGIN
SET NEW.logday = CURDATE(), NEW.logtime = CURTIME();
IF INSTR(NEW.email,'@') = 0 THEN
SIGNAL SQLSTATE '50002'
```

```
SET MYSQL_ERRNO = 5002, MESSAGE_TEXT = 'invalid email
address';
END IF;
END$$
DELIMITER ;
DELIMITER $$
CREATE OR REPLACE TRIGGER PERSON_CONTACT2_UPDATE_EMAIL
BEFORE UPDATE ON PERSON_CONTACT2
FOR EACH ROW
BEGIN
IF INSTR(NEW.email,'@') = 0 THEN
SIGNAL SQLSTATE '50002'
SET MYSQL_ERRNO = 5002, MESSAGE_TEXT = 'invalid email
address';
ELSE
IF NEW.email <> OLD.email THEN
SET NEW.logday = CURDATE(), NEW.logtime = CURTIME();
END IF;
END IF;
END$$
DELIMITER ;

MariaDB [webdb]> insert into PERSON_CONTACT2(Name, email)
values('Jones', 'jones@some.where');
Query OK, 1 row affected (0.012 sec)
MariaDB [webdb]> insert into PERSON_CONTACT2(Name, email)
values('Smith', 'smith.some.where');
ERROR 5002 (50002): invalid email address
MariaDB [webdb]> insert into PERSON_CONTACT2(Name, email)
values('Smith', 'smith@some.where');

MariaDB [webdb]> select * from  PERSON_CONTACT2;
 +-----+-------+------------------+------------+----------+
 | Pid | Name  | email            | logday     | logtime  |
 +-----+-------+------------------+------------+----------+
 |   3 | Jones | jones@some.where | 2020-03-09 | 14:00:51 |
 |   4 | Smith | smith@some.where | 2020-03-09 | 14:01:06 |
 +-----+-------+------------------+------------+----------+

MariaDB [webdb]> update PERSON_CONTACT2 set
email='jones@else.where' where Name='Jones';
Query OK, 1 row affected (0.000 sec)
Rows matched: 1  Changed: 1  Warnings: 0
Query OK, 1 row affected (0.000 sec)
MariaDB [webdb]> select * from  PERSON_CONTACT2;
```

```
+-----+-------+-------------------+------------+----------+
| Pid | Name  | email             | logday     | logtime  |
+-----+-------+-------------------+------------+----------+
|   3 | Jones | jones@else.where  | 2020-03-09 | 14:03:11 |
|   4 | Smith | smith@some.where  | 2020-03-09 | 14:01:06 |
+-----+-------+-------------------+------------+----------+

MariaDB [webdb]> update PERSON_CONTACT2 set
email='smith.else.where' where Name='Smith';
ERROR 5002 (50002): invalid email address
MariaDB [webdb]>  select * from  PERSON_CONTACT2;

+-----+-------+-------------------+------------+----------+
| Pid | Name  | email             | logday     | logtime  |
+-----+-------+-------------------+------------+----------+
|   3 | Jones | jones@else.where  | 2020-03-09 | 14:03:11 |
|   4 | Smith | smith@some.where  | 2020-03-09 | 14:01:06 |
+-----+-------+-------------------+------------+----------+
```

6.7 Events

Most DBMS including MySQL/MariaDB have an event scheduler that allows the setting up of regular database operations to run at a specified frequency or time. In order for these operations to be executed the event scheduler status should be ON. Check the current status by using the following command.

```
MariaDB [webdb]> show variables like 'event_scheduler'
              +-----------------+-------+
              | Variable_name   | Value |
              +-----------------+-------+
              | event_scheduler | OFF   |
              +-----------------+-------+
```

Enabling of the event_scheduler requires superuser privilege to run the command:

```
MariaDB [webdb]> set global event_scheduler =1;
Query OK, 0 rows affected (0.000 sec)
MariaDB [webdb]> show variables like 'event_scheduler'
              +-----------------+-------+
              | Variable_name   | Value |
              +-----------------+-------+
              | event_scheduler | On    |
              +-----------------+-------+
```

To log events, create the table, logofevents. For the attribute, quand, MariaDB automatically assigns the following additional properties to the column, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

```
MariaDB [webdb]> CREATE TABLE LOGOFEVENTS(
   quand TIMESTAMP,
MESSAGE VARCHAR(31));
Query OK, 0 rows affected (0.020 sec)
MariaDB [webdb]> desc LOGOFEVENTS;

+-------+-----------+----+---+-------------+-----------------+
|Field  |Type       |Null|Key|Default      |Extra            |
+-------+-----------+----+---+-------------+-----------------+
|quand  |timestamp  |NO  |   |current_time)|on update current|
|MESSAGE|varchar(31)|YES |   |stamp(NULL   |_timestamp()     |
+-------+-----------+----+---+-------------+-----------------+

MariaDB [webdb]> insert into LOGOFEVENTS (MESSAGE)
value ('Start of event log')
Query OK, 1 row affected (0.001 sec)

MariaDB [webdb]> select * from LOGOFEVENTS;
        +---------------------+--------------------+
        | quand               | MESSAGE            |
        +---------------------+--------------------+
        | 2020-03-09 14:48:02 | Start of event log |
        +---------------------+--------------------+

MariaDB [webdb]>CREATE EVENT logging_start
ON SCHEDULE EVERY 5 MINUTE
Do INSERT INTO LOGOFEVENTS(MESSAGE) values('Another');

MariaDB [webdb]> select * from  LOGOFEVENTS;
        +---------------------+--------------------+
        | quand               | MESSAGE            |
        +---------------------+--------------------+
        | 2020-03-09 15:23:36 | Start of event log |
        | 2020-03-09 15:23:50 | Another            |
        | 2020-03-09 15:28:50 | Another            |
        | 2020-03-09 15:33:50 | Another            |
        +---------------------+--------------------+

MariaDB [webdb]> drop event logging_start;
Query OK, 0 rows affected (0.002 sec)
```

6.8 Cursor

A cursor is a symbolic name for a pointer to a row of data returned by a SQL SELECT statement. Most DBMS's use a work area, to store the results of a SQL SELECT statement and a cursor identifies the location of a particular row of data in this work area.

A cursor is declared in a procedure by the following syntax.

Syntax: CURSOR *cursorname* FOR *selectstatement*;

There are optional parts to the syntax which are not given in this declaration.
The cursor name in a program, *cursorname*, acts as a pointer to (address of) a row in the set of rows returned by the SELECT statement, *selectstatement*.

The cursor name, *cursorname*, is used in four steps, which have the following syntax.
(1) CURSOR - declare cursor
Syntax: given above
(2) OPEN - execute the query and assign the value of *cursorname* to point to the first row of data
Syntax: OPEN *cursorname*;
(3) FETCH - retrieve current row of data and assign the value of *cursorname* to point to the next value of data
Syntax: FETCH *cursorname* INTO *var_name [, var_name] ...*
(4) CLOSE - deallocate memory used by cursor
Syntax: CLOSE *cursorname*;

A cursor must be declared and then opened before it is used. An example of the use of a cursor follows. At the DB prompt, create the SQL script given below and save it using the name indicated and exit emacs.

```
MariaDB [webdb]> system emacs faux-merge-two-lists.sql;

DROP TABLE IF EXISTS list1;
CREATE TABLE list1 (fname char(15));
DROP TABLE IF EXISTS list2;
CREATE TABLE list2 (fname char(15));
DROP TABLE IF EXISTS list;
CREATE TABLE list  (fname char(15));
DROP PROCEDURE IF EXISTS faux_merge;
DELIMITER //
CREATE PROCEDURE faux_merge()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE n1, n2 char(15);
  DECLARE ptr1 CURSOR FOR SELECT fname FROM list1;
  DECLARE ptr2 CURSOR FOR SELECT fname FROM list2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
  OPEN ptr1;
  OPEN ptr2;
  read_loop: LOOP
    FETCH ptr1 INTO n1;
    FETCH ptr2 INTO n2;
   IF done THEN
      LEAVE read_loop;
    END IF;
    IF n1 < n2 THEN
      INSERT INTO list VALUES (n1);
    ELSE
```

```
        INSERT INTO list VALUES (n2);
      END IF;
    END LOOP;
    CLOSE ptr1;
    CLOSE ptr2;
  END; //
  DELIMITER ;
  INSERT INTO list1 VALUES ('Albert'),('Mike'),('Roy');
  INSERT INTO list2 VALUES ('Elsbeth'), ('Jane'), ('Mary');
  CALL faux_merge;
  SELECT * FROM list;

  MariaDB [webdb]>  source faux-merge-two-lists.sql;

                          +--------+
                          | fname  |
                          +--------+
                          | Albert |
                          | Jane   |
                          | Mary   |
                          +--------+
```

6.9 Conclusions

With the influence of the new internet based applications, the relational database is changing to provide support for these new applications. Along with this, SQL is evolving. One of the areas is to support what is called NoSQL and column type data. For example, the recent versions of MariaDB have support for more features of Oracle's PL/SQL as well as column data. These topics are not covered in this text.

7 HTML

7.1 Introduction

HTML stands for HyperText Markup Language. It is the language in which web pages are written. So a web page is a program written in HTML. A browser is the general name for a program that interprets (executes) the web pages and displays them on the screen. A browser is also called web browser or web client but will always be called a browser here.

In this chapter HTML will be presented. CSS (Cascading Style Sheets) are not used in this chapter. CSS is covered in the next chapter. HTML 5 is the latest version. HTML is evolving with a promise of backward compatibility. How this would be achieved is an open question. Also there is a move to migrate to XHTML (eXtensible HyperText Markup Language). XHTML is similar to HTML but includes additional features and rules. The main differences between HTML and XHTML are given in a section of the next chapter where XHTML is used together with CSS.

Web pages can include HTML combined with code written in various other languages, generally called scripting languages. Examples of scripting languages are PHP and JavaScript. The scripting languages, PHP and JavaScript, are covered in later chapters.

The HTML language consists of content which is marked up by constructs called tags. There are three kinds of content: text, images (multimedia) and links. In most of the demonstrations in this chapter only text and links are used, and both are referred to as text. The markup tags specify the nature of the contents and/or the format in which the content is to be presented (displayed) by the browser.

The two most widely used browsers at present for Windows operating systems are Microsoft Internet Explorer and Firefox. Firefox is open source (free) software. For Unix/Linux operating systems, the most popular browser at present is FireFox. Internet Explorer is not available on Unix/Linux. Other browsers are Opera, Cliq, Seamonkey, and Chrome.

7.2 History

Research into hypertext was underway ever since the 1945 article in The Atlantic by Verner Bush. Hypertext was discussed in science fiction literature even earlier. HTML created by Tim Berners-Lee at CERN (Centre Européan de Recherche Nucléaire) in Geneva, Switzerland and first used at CERN in 1991 was accepted by the internet community. At the same time CERN developed HTTP (hypertext transfer protocol) to transmit web pages between computers on a network and a browser program to receive and display web pages. The browser program was a text-based (also called line-mode) user interface which was given the name, world wide web, at CERN. Also the software for the first web server was written at CERN and was initially on only one computer at CERN which stored all of the web pages.

The first graphical user interface (gui) browser was developed at the University of Illinois for the National Center for Supercomputing Applications (NCSA) in 1993 and was called Mosaic. The leader of the team that developed Mosaic was Marc Andreesen who in 1994 left NCSA to start the company, Mosaic Communications Corporation, in partnership with Jim Clark. The company was renamed, Netscape Communications Corporation, in the same year after legal action was taken by the University of Illinois about the name, Mosaic. This company in 1994 released the first version of the browser now called Netscape.

The world wide web consortium (W3C) was established in 1994 which maintains the standards of all web technologies, including HTML, XHTML and CSS. A short account of the history of the web (web is short for world wide web) is on a web site given in the last section, References.

7.3 Web Page

The file with the contents to be displayed by a web browser is called a web page or, more formally, an HTML document. The filename of the web page usually has the extension, .htm or .html, so that it can be identified as a web page. For example, a web page file can be named file1.htm or file1.html. It is convenient to use lowercase letters for filenames so that the names are easy to remember.

A group of related web pages is called a web site. When there are several web pages in the same directory, the web page with the name, index.html, should be used for the home page (the first page which has links to the other pages). When the home page has the name, index.html, and the browser requests a web page from a web site but does not give the name of the web page, the page, index.html, is sent to the browser.

A web page (a file) is stored on the hard disk of the server and displayed on the screen of the client. This is discussed next.

7.4 Client and Server

The concepts of client and server are introduced in Chapter 1. The terms, client and server, are the names of computers which have two different functions as determined by the programs which are running on the computers. The client runs the program called a browser and the server runs the program called an http server. The http server is also called a web server.

Note the overloading in the terminology that relates to the server; both the computer and the program that runs on the computer are called a server. So it is necessary to qualify the word, server, if there is any possibility of ambiguity. The server program is a daemon (program that runs in the background) while the server computer is hardware on which the program runs.

A web page is displayed by a browser which is running on a client computer that is accessed by a person (always called a user) using a keyboard or mouse. The web page is stored on the persistent storage (usually a hard disk) of a server computer, and the program, http server, running on the server computer sends the web page to the client computer, when the browser requests the page. The two computers, client and server, are connected to each other by the Internet, much like two telephones are connected to each other by a telephone network.

The web page to be displayed by the browser is specified either by typing the address of the web page in the address bar of the browser or by clicking a link to a web page which is inside another web page that is being displayed by the browser. The browser sends a request to the server computer indicated in the address. The server computer receives the request and responds by sending the web page to the client computer and then the browser displays the new web page on the screen.

The address of a web page is more formally called a URL, which is discussed next.

7.5 **URL**

URL is the acronym for Uniform Resource Locator. The URL gives the address or location of a file on a server on the Internet. Every information resource in the world which is accessible on the Internet must have a unique URL. Often the word, address, is used instead of URL. In particular, the text field in some browsers where the current URL is displayed and where a new URL is to be typed is labeled, Address or not labeled at all.

7.5.1 Parts of a URL

The URL for a web page (file with HTML instructions) has four parts with the syntax as follows:
protocol://server:port/file

7.5.1.1 protocol

The protocol for the world wide web is http (hypertext transfer protocol), the set of rules for sending files and other data between a browser and http server over the Internet.

For the protocol, http, the characters, **://**, follow the protocol. For some other protocols, only the character, **:**, follows the protocol. The web also supports gateways for other protocols such as FTP (file transfer protocol). For the gateway to a mail agent the mailto protocol is used, which is an example of a protocol which is followed by only, **:**.

7.5.1.2 server

Server is the name of the computer which stores the web page. The server name is also called the domain name, which is discussed in the next section. The server name represents a numerical address of the server computer. The numerical address is called the IP (internet protocol) address. Either the server name or IP address can be used in the URL. The IP address of a computer connected to the Internet is currently a 32-bit number divided into four one-quarter parts (four 8-bit numbers) separated by dots. Each part is written as a decimal number. Since the minimum and maximum value of an 8-bit number is 0 and 2^8-1, each of the four decimal numbers has values between 0 and 255.

A colon, :, separates the server and port. If the port is not included in the URL the default port of 80 is used, and a slash, /, separates the server and file.

Every computer connected to the internet, including both server computers and client computers, has a unique IP address. A server computer has a static IP address, meaning an IP address that does not change with time. A client computer may have a dynamic IP address, an IP address that changes from time to time due to the limited number of IP addresses available to the internet service provider (ISP) of the client.

7.5.1.3 port

Port is a 16-bit number, a number between 0 and 65535 in decimal. The default port on a server for the http protocol is 80 and if the default port for http is used, the port can be omitted in the URL. A slash, /, separates the port and file.

7.5.1.4 file

File is the path which is zero or more directories followed by the name of a file on the hard disk of the server, all separated by slashes, /. So file has the format, *dirname1*/*dirname2*/ . . . /*filename*. Here *dirname1*, *dirname2 ...* are the names of directories in the path to the file, *filename*. If the web site belongs to a user with name, *username*, sometimes the first directory is preceded by, *~username,* to indicate that it is the web site for that user.

If *filename* is not given after the last directory in the path, a default file in that directory may be displayed by the browser. Usually the default file has the name, index.html.

7.5.2 Example of a URL

The URL of a web page in Computer Science at Concordia University is

http://www.cse.concordia.ca:80/currentstudents/courseschedules/Undergraduate_Fall_2019-2020.php

The four parts of the URL are:

(1) protocol - http
(2) server - www.cse.concordia.ca
(3) port - 80
(4) file -currentstudents/courseschedules/Undergraduate_Fall_2019-2020.php
currentstudents is the name of a directory in the web document root directory of the server,
courseschedules is a name of subdirectory in directory, currentstudents and
Undergraduate_Fall_2011-2012.php is the name of a file in directory, courseschedules

The server name, www.cse.concordia.ca, represents the IP address, 132.205.96.30.
The URL can be written using the IP address instead of the server name:

http://132.205.96.30:80/currentstudents/courseschedules/Undergraduate_Fall_2019-2020.php

Since the default port is 80 for the http protocol, it is unnecessary to include the default port in the URL. So the URL can be written without the port by either using the server name or the IP address:

http://www.cse.concordia.ca/currentstudents/courseschedules/Undergraduate_Fall_2019-2020.php
http://132.205.96.30/currentstudents/courseschedules/Undergraduate_Fall_2019-2020.php

The best analogy to a URL is a mailing address. The protocol is not shown in a mailing address. The protocol corresponds to the method of delivery, for example, delivery by the post office or by a private courier.

Example of a mailing address:

Prof. Bill Bates - line 1

Room 3.121 - line 2
Department of Computer Science - line 3
Concordia University - line 4
1455 de Maisonneuve Blvd. - line 5
Montreal, QC H3G 1M8 - line 6
Canada - line 7

The server in a URL corresponds to lines 4 - 7. The file in a URL corresponds to lines 1 - 3, where line 1 corresponds to the filename and lines 2 and 3, directories. There is some redundancy in a mailing address, unlike in a URL, since the postal code in a Canadian address identifies a small region of Canada (H3G 1M8 is a small part of Montreal).

7.6 **Domain Name**

All computers connected to the internet, both clients and servers, can be given a name, called a domain name, which represents the 32-bit IP address of the computer, introduced in the last section. The main reason for using a domain name is that a name is easier to remember than a number. Furthermore, the number is arbitrary but the name can describe the web site. In the example in the last section, the domain name, www.cse.concordia.ca, includes cse which stands for computer science engineering, concordia which is the name of a university and ca which stands for Canada. This is easier to remember than the IP address, 132.205.96.30. If an IP address of a web site changes for some reason, the domain name does not change; rather the IP address is changed in the database of the domain name server, described next.

A domain name server is a computer that converts domain names into the corresponding IP addresses, a 32-bit number as described above. The IP address is needed to establish a connection between computers, just as a telephone number is needed to connect telephones. The browser of a client computer sends a domain name to one of many computers (the domain name servers) which access a database called the Domain Name System (DNS). This database contains the domain name and the IP address. If the IP address is found for the domain name, the browser uses the IP address received from the domain name server to connect to the web server.

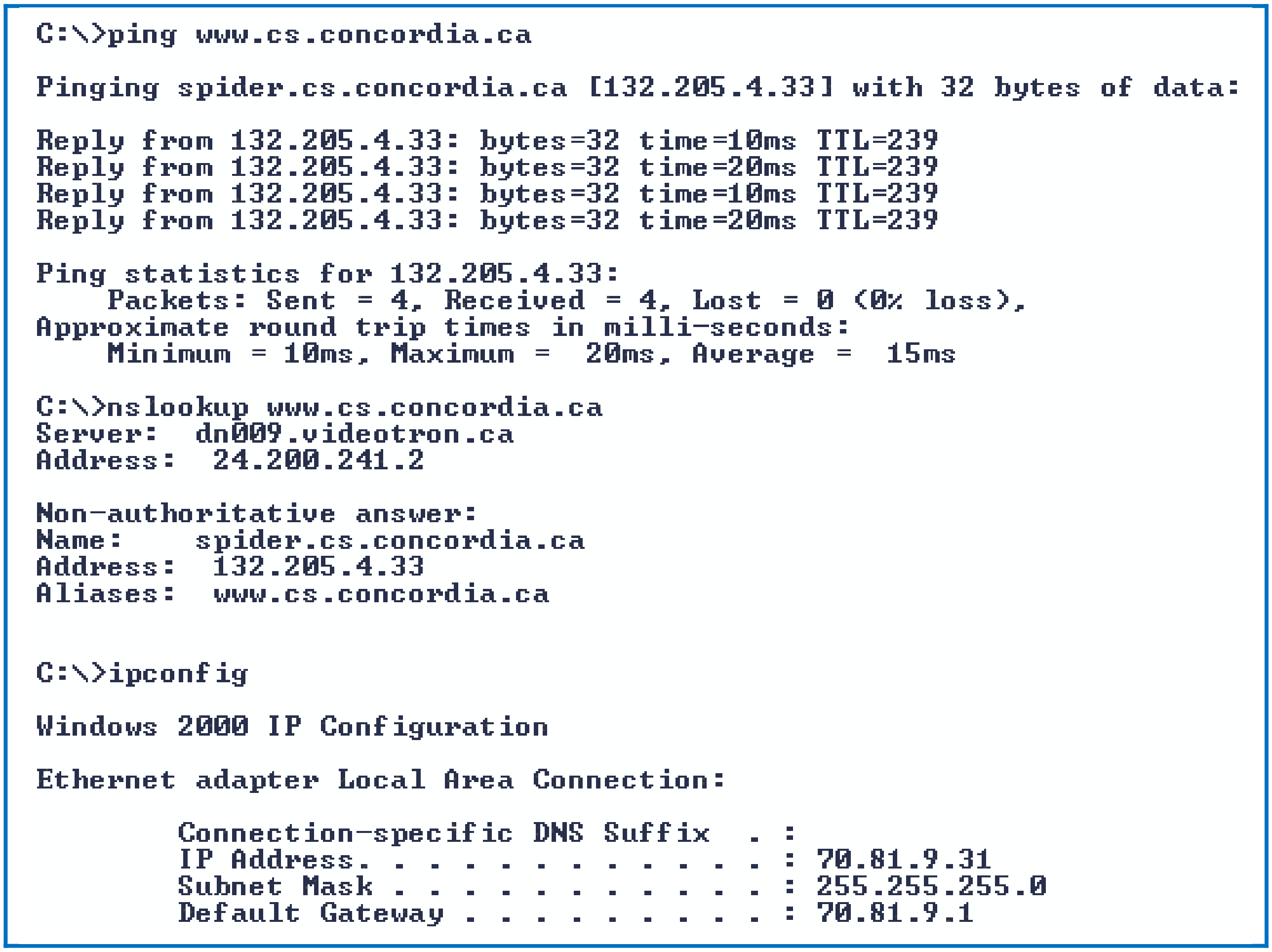
The IP address that corresponds to a domain name is found by using the command, ping or nslookup, at the DOS prompt of a Windows operating system. On a Unix system, ping and nslookup are also used, but nslookup is being replaced with htdig. Client computers (the computers with the browser) do not have domain names but do have IP addresses since they are connected to the internet. The IP address of a client computer is found by issuing the command, ipconfig, at the DOS prompt of a Windows operating system.

To start a DOS shell in recent version of Winx, use the search box; for older version of Winx click: Start/Programs/Accessories/Command Prompt.

To stop the DOS shell, type, exit, at the prompt. Figure 7.1 shows commands, ping and nslookup, to display the IP address for the server, www.cse.concordia.ca, in Windows 2000. Figure 7.1 also shows the command, ipconfig, to display the IP address of the host computer

(computer on which the command was given). The commands given in Figure 7.1 are the following.

ping www.cse.concordia.ca
nslookup www.cse.concordia.ca
ipconfig



```
C:\>ping www.cs.concordia.ca

Pinging spider.cs.concordia.ca [132.205.4.33] with 32 bytes of data:

Reply from 132.205.4.33: bytes=32 time=10ms TTL=239
Reply from 132.205.4.33: bytes=32 time=20ms TTL=239
Reply from 132.205.4.33: bytes=32 time=10ms TTL=239
Reply from 132.205.4.33: bytes=32 time=20ms TTL=239

Ping statistics for 132.205.4.33:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 10ms, Maximum =  20ms, Average =  15ms

C:\>nslookup www.cs.concordia.ca
Server:  dn009.videotron.ca
Address:  24.200.241.2

Non-authoritative answer:
Name:    spider.cs.concordia.ca
Address:  132.205.4.33
Aliases:  www.cs.concordia.ca


C:\>ipconfig

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 70.81.9.31
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 70.81.9.1
```

Figure 7.1 IP addresses

7.7 Components of a Web Page

The language, HTML (HyperText Markup Language), is used to create HTML documents, which are always called web pages. Therefore, web pages are programs written in HTML. The components of a web page are basically content and tags.

The content in a web page is the information contained by the web page. Content includes text, images (more generally, multimedia) and links. The tag is an instruction that formats the content. Tags usually come in pairs: opening tag and closing tag. The browser executes the instructions (tags) by displaying the content in the format specified by the tags. The formatting of the content by tags is called marking up the content.

The term, mark up, originally was used in the printing industry where a manuscript was in a printed form on paper (instead of electronic form in a computer). An editor (a person, not a program) marked up (meaning, wrote on) the manuscript with a variety of instructions to indicate the style of text to be used, the layout of the text and so on. The document was prepared for printing using these instructions. The mark up (instructions written on pages of a manuscript) corresponds to the tags in a web page.

HTML tags can be distinguished by the purpose of the tags. Loosely, the tags can be classified on this basis as follows.

$ structural tags: define how text is organized
$ appearance tags: define how text is presented
$ location tags: define where text and other information appear
$ hypertext links tags: define links to other places
$ list tags: define ordered and unordered lists

A formal description of the composition of a web page is expressed in terms three components: attributes, tags and elements. A definition of the components and examples follow.

7.7.1 Attribute

An HTML attribute is a property applied to the content. The HTML attribute has a name and value. It is enclosed in the opening tag. Most attributes are optional.

Syntax:

attributename=attributevalue

Examples:

face=helvitica
size=5
color=green

In the examples there are three attributes. The first attribute has name, face, and value, helvitica. The second attribute has name, size, and value, 5. The third attribute has name, color, and value, green.

In XHTML all values of attributes must be enclosed in quotes. So the first example in XHTML is

face=”helvitica”

In HTML it is not required to enclose attribute values in quotes unless the value includes white space or special characters. Attribute values are not enclosed in quotes here.

7.7.2 Tag

An HTML tag is an instruction to format (markup) content. Most tags, but not all, come in pairs called opening tags and closing tags. The opening tag starts with the name of the tag followed by optional attributes (indicated by square brackets in the syntax below). The closing tag consists only of the name of the tag preceded by a slash (/). The name of the tag can be lowercase, uppercase or a combination of both, that is, the HTML language is not case sensitive. Note, however, that XHTML is case sensitive and lowercase must be used.

Syntax:

(a) syntax of opening tag: <*tagname* [*attribute1 attribute2* . . .]>
where square brackets mean optional
(b) syntax of closing tag: </*tagname*>

Examples:
Example of opening tag: `<font color=green>`
where font is the tagname and color=green is an attribute
Example of corresponding closing tag: `</font>`

7.7.3 Element

An HTML element is the smallest unit of a web page which produces output that is displayed on the screen. In general, an HTML element consists of an opening tag, content, and a closing tag.
Syntax:
<*tagname* [*attribute1 attribute2* . . .]>*content*</*tagname*>

Examples:

```
<p>First paragraph</p>
```

In the example, p is a tagname and First paragraph is the content.

```
<h1 align=center>Main heading</h1>
```

In the example, h1 is the tagname, align=center is an attribute and Main heading is the content.

```
<font face=helvitica size=5 color=green>Hello</font>
```

In the example, font is the tagname, face=helvitica size=5 color=green are three attributes and Hello is the content.

7.8 Editors

There are two kinds of editors that can be used to create a web page: text editor and HTML editor. A text editor is a word processor that is used to type both the content of the web page and all of the HTML tags in the web page. A text editor displays the HTML code. An HTML editor is a program that simplifies the creation of a web page by using menus and various tools to make the web page instead of typing the HTML tags. An HTML editor shows the output of the HTML code as displayed by a browser, and this is edited by the programmer.

A web page can be created on any computer. If the web page is created by an editor on the server, then the file can be saved directly on the hard disk of the server. However, if the web page is made using an editor of a client computer (a computer with a browser) and saved on the hard disk of that computer, it must be transferred to the server computer through the internet, using a file transfer program (ftp). Of course, the programmer who makes the web pages must have an account on the server computer in either case.

7.8.1 Text editor

A text editor allows complete control over the creation of the web page, but requires considerable knowledge of and experience with HTML, since all tags must be typed by the programmer.

Any text editor can be used to create a web page (file with HTML instructions). The simplest editor in Windows is Notepad which can only save files as text. Notepad adds the extension, .txt, to filenames. For example, if the name, file1.htm, is used to save a file, Notepad may name the file, file1.html.txt. To avoid this, enclose the name in double quotes, “file1.htm”, when saving the file in Notepad. More sophisticated editors (word processors), like Microsoft Word, can also be used to create a web page as long as the file is saved as type, text, which is called, for example, MS_DOS text by Word and ASCII DOS text by WordPerfect.

A popular text editor in Linux is emacs. Emacs can be set up to show different parts of the HTML code in different colors.

7.8.2 HTML editor

A HTML editor requires only a brief knowledge of HTML because the editor inserts the tags. But usually a HTML editor generates a lot more HTML source code than is necessary and generally will not produce exactly the layout or appearance that is desired. So a programmer may need to change the source code, that is, change some tags, after using a HTML editor.

HTML editors are sometimes referred to as WYSIWYG (what you see is what you get) editors, since the output of the HTML code is displayed by the editor. Like most programs, there are three types of HTML editors in terms of how they are marketed: commercial, shareware and freeware. Commercial programs must be bought before they can be used. Shareware are programs that can be used without cost for a period of time and then paid for if the program will continue to be used. Freeware are programs that can be used for free and are always available for download on the internet.

The most popular HTML editor used currently by professional web programmers is Dreamweaver, a commercial program which is relatively expensive. HTML editors which are shareware and freeware can be found by searching the internet.

7.9 Structure of a Web Page

The structure of a web page consists of two basic parts: (1) the head or the preamble and (2) the body. The head, except for the title, is not displayed on the screen by the browser. The body contains the content that is displayed on the screen.

The tags which define the basic parts of a web page are shown below and briefly described. The tags below, like most tags, consist of pairs of opening and closing tags. The tags are not case sensitive and are often written in uppercase so that they are easy to see in a document. However, it is easier to type the tags in lowercase in an actual web page. Note that in XHTML tags must be typed in lowercase.

```
<HTML>    <!-- indicates the start of a web page (at the top of a
               document)-->
   <HEAD> <!-- indicates the start of the head of a web page.-->
                   <!-- The title of the web page is here
                        and also other information.    -->
```

```
    <TITLE>     <!-- indicates the start of the title of a web
              page -->
    </TITLE>    <!-- marks the end of the title -->
  </HEAD>     <!-- marks the end of the head -->
  <BODY>      <!-- indicates the start of the body of the web
              page -->
                  <!-- The content of the web page, that is,
                       the text which is displayed by the
                       browser, is here. -->
  </BODY>     <!-- marks the end of the body of the web page-->
</HTML>       <!-- marks the end of a web page -->
```

The web page starts and ends with html tags to identify the contents of the file as html instructions. The head tags and body tags are between the html tags. Comments (text ignored by the browser) are enclosed in the tag which starts with, <!--, and ends with, -->. This is multiline comment which corresponds to the comment style in the C language, /* *comment* */.

7.10 **Web Page Demonstrations**

The web page demonstrations illustrate by examples the use of the common HTML tags in Figure 7.2. The tags in Figure 7.2 are also listed in Table 7.1 which also gives the name and brief description of the tag. The tags are divided into seven categories for the purpose of demonstrating the tags. Each category is covered by a subsection below. The categories themselves are somewhat arbitrary in terms of classifying the tags.

There are two kinds of web page demonstrations given in this section. One group of demonstrations is represented by the source code of the web pages and the source code is given in the text of this chapter. These web pages are stored on a server and can be viewed by typing the URL of the web page in a browser.

Alternatively, the web pages can be stored on a client (computer with a browser) and viewed by clicking, File/Open, on the menu bar of the browser and then selecting an HTML file. On the client make a directory, C:\htmldemos, for example if Windows is used, and then store all web pages in that directory.

For convenience a web page, called a menu, has links to all of the web page demonstrations and each web page has a link back to the menu web page. The links in the web pages assume that all web pages are stored in the same directory as the menu. These web pages are numbered and are listed at the beginning of this chapter after the Contents. The display of these web pages by the browser are shown in figures in this chapter, which give the name of the web page (file name) in the caption.

The other group of demonstrations does not have source code in this chapter. The demonstrations are only shown in figures which include both the source code and the display of the web page by a browser. Of course, the source code can be typed, stored in a web page and displayed by a browser, if desired, but the display by the browser is already shown in the figure.

7.10.1 Menu

The web page, menuhtml.htm, is a menu which gives the links to demonstration programs in this section.

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

In the web home directory on the server, create the directory, c07html, and then set the permissions of the directory to 707 (write permission is required because the interactive web page writes files on the server). After, change the working directory to c07html. This is done by the following series of Linux commands.

```
pwd                       (display working directory)
mkdir c07html             (create directory, c07html)
chmod 707 c07html         (change permissions of directory, c07html)
cd c07html                (change working directory to c07html)
```

Web Page 7.1 menuhtml.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

Start a text editor in Linux, for example, emacs. Type the following file, or copy and paste it, and save it using the filename, menuhtml.htm, in directory, c07html. In Linux it is necessary to set the permissions of all web pages to 604. Set the permissions for the menu web page after typing and saving the file. View the permissions before and after changing the permissions by the command, ls -l.

```
emacs menuhtml.htm        (type and save the file, then exit emacs)
ls -l m*                  (display permissions of all files starting with letter, m)
chmod 604 menuhtml.htm    (change permissions of file, menuhtml.htm)
ls -l m*                  (display permissions of all files starting with letter, m)
```

(Type the following web page)

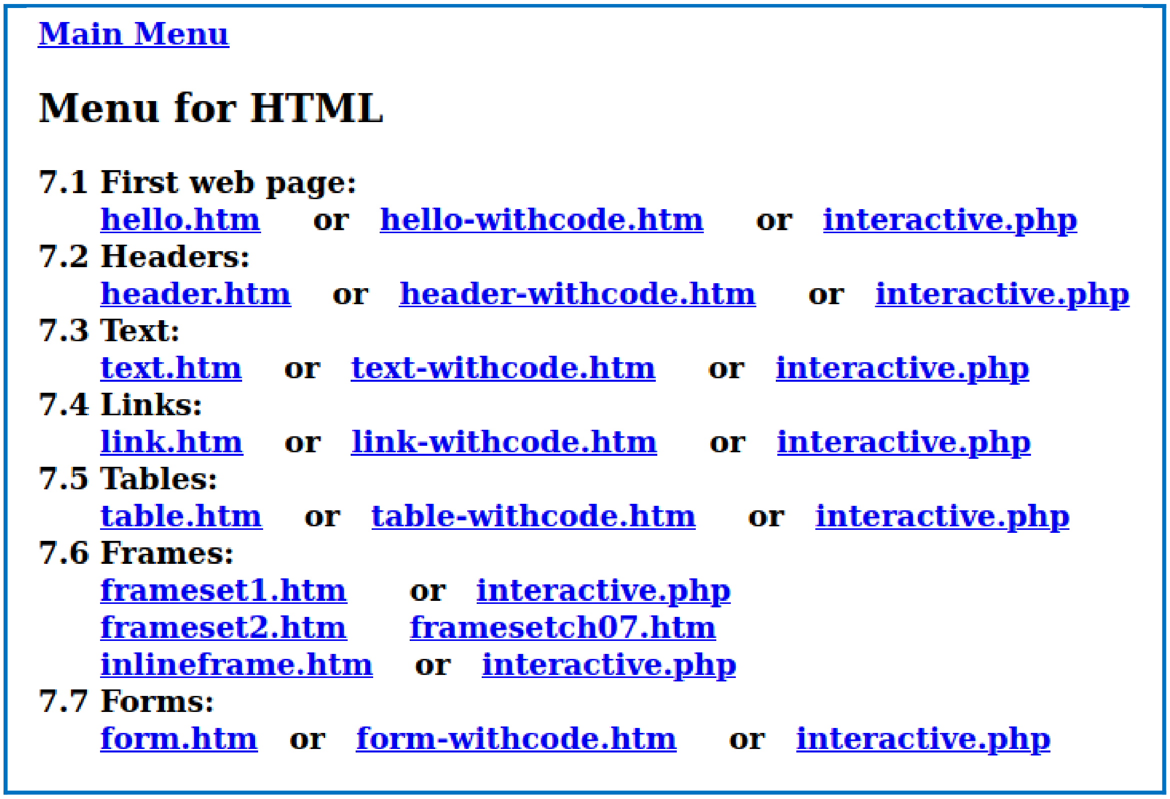
```
<html>
<head>
    <title>Menu for HTML</title>
<head>
<body>
<table width=100%><tr><td width=35%>
<h2><a href="../index.html">Main Menu</a></h2>
<h1>Menu for HTML</h1>
<h3>
7.1 First web page:<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="hello.htm">hello.htm</a>
or <a href="hello-withcode.htm">hello-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;or <a href="interactive.php?
fn=hello.htm">interactive.php</a><br><br>
7.2 Headers:<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="header.htm">header.htm</a>
or <a href="header-withcode.htm">header-withcode.htm</a><br>
```

```
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;or <a href="interactive.php?
fn=header.htm">interactive.php</a><br><br>
7.3 Text:<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="text.htm">text.htm</
a>
or <a href="text-withcode.htm">text-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;or <a href="interactive.php?
fn=text.htm">interactive.php</a><br><br>
7.4 Links:<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="link.htm">link.htm</
a>
or <a href="link-withcode.htm">link-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;or <a href="interactive.php?
fn=link.htm">interactive.php</a><br><br>
7.5 Tables:<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="table.htm">table.htm</a>
or <a href="table-withcode.htm">table-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;or <a href="interactive.php?
fn=table.htm">interactive.php</a><br><br>
7.6 Frames:<br>
<blockquote>
<a href="frameset1.htm">frameset1.htm</a><br>
&nbsp;&nbsp;or <a href="interactive.php?
fn=frameset1.htm">interactive.php</a><br>
<a href="frameset2.htm">frameset2.htm</a><br>
<a href="framesetch07.htm">framesetch07.htm</a><br>
<a href="inlineframe.htm">inlineframe.htm</a><br>
&nbsp;&nbsp;or <a href="interactive.php?
fn=inlineframe.htm">interactive.php</a>
</blockquote>
7.7 Forms:<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="form.htm">form.htm</
a>
or <a href="form-withcode.htm">form-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;or <a href="interactive.php?
fn=forminteractive.htm">interactive.php</a><br>
</td><td width=65% height=100% valign=top>
<iframe src=texttags.htm width=100% height=100%></iframe>
</td></tr></table></h3></body></html>
```

(End of the web page)

Start a browser, for example, Internet Explorer or Firefox in Windows, or Firefox in Linux. Type the address (address is used in this context to mean URL) of the file, menuhtml.htm, in the address bar of the browser and then press Enter. The address has the form, http://*servername*/*directoryname(s)*/c07html/menuhtml.htm. Note that the title appears in the title bar at the top of the screen.

Figure 7.2 shows the menu as displayed by the browser. Table 7.1 Text Formatting Tags, is also displayed by the browser to the right of the menu but is not shown in Figure 7.3.



Main Menu

Menu for HTML

7.1 First web page:
hello.htm **or** hello-withcode.htm **or** interactive.php
7.2 Headers:
header.htm **or** header-withcode.htm **or** interactive.php
7.3 Text:
text.htm **or** text-withcode.htm **or** interactive.php
7.4 Links:
link.htm **or** link-withcode.htm **or** interactive.php
7.5 Tables:
table.htm **or** table-withcode.htm **or** interactive.php
7.6 Frames:
frameset1.htm **or** interactive.php
frameset2.htm framesetch07.htm
inlineframe.htm **or** interactive.php
7.7 Forms:
form.htm **or** form-withcode.htm **or** interactive.php

Figure 7.2 Menu for web pages: menuhtml.htm

There are three web pages for each demonstration. For example, the demonstration of headers, has the three web pages, headers.htm, headers-withcode.htm and interactive.php.
(a) The web page, headers.htm, shows only the output of the HTML source code.
(b) The web page, headers-withcode.htm, shows both the source code itself followed by the output of the source code. This web page can be used to associate the lines of source code with the output of the corresponding source code, to learn HTML.
(c) The web page, interactive.php, shows the source code on the left side of the page and the output of the source code on the right side. The source code can be changed by typing on the left side and then, Submit, can be clicked to see the output of the changed source code on the right side. To try this, change one of the words, hello, to, hi, for example and then click Submit. The original source code and its output can be displayed again by clicking, original page.

The source code itself is inserted into the web page, (b), by simply replacing the two characters which are the tag delimiters, < and > , by the escape sequences for the characters, &lt; for < and &gt; for >, and enclosing the entire code in the tags, <pre> and </pre>. Escape sequences and tags are discussed later.

The code for the web pages that display only the output of the code, such as header.htm, is given. The code for the web pages that display both the code and the output of the code, such as header-withcode.htm, is not given. Also the code for the web page, interactive.php, is not given.

Common HTML Tags

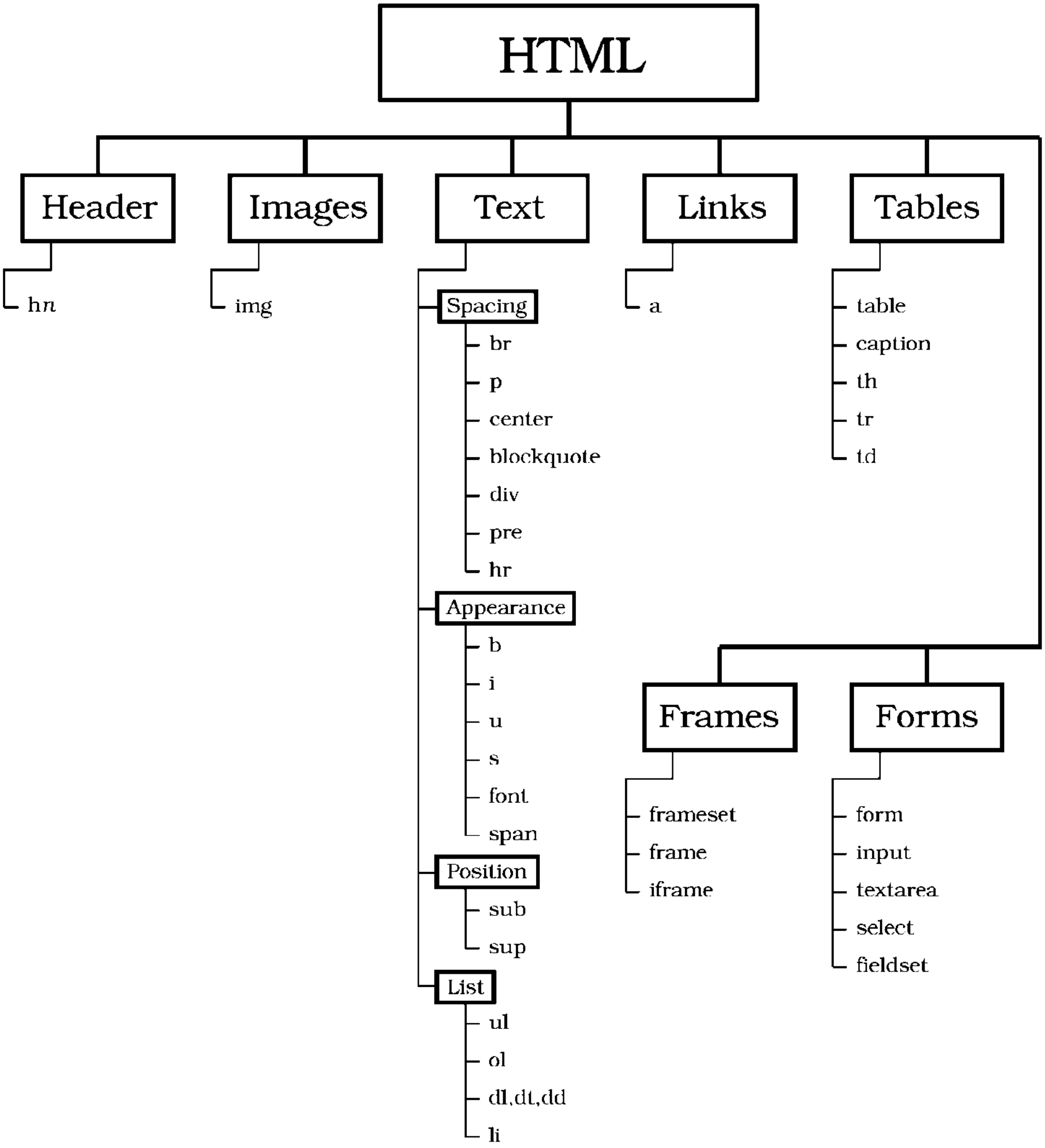


Figure 7.3 Common HTML Tags

Table 7.1 Common HTML Tags

Category	Tag	Name	Description
Spacing	p	paragraph	start new paragraph
	br	line break	start new line
	pre	preformat	display text using the spacing in the html
	center	center	display text or other elements in the center
	div	division	divide text into sections and align sections
	blockquote	blockquote	indent one or more lines of text
	hr	horizontal rule	draw horizontal line the width of the
Appearance	h*n*	header	change size of characters and start new
	b	bold	display text in bold
	i	italics	display text in italics
	u	underline	display text with line under it
	s	strike	display text with line through it
	font	font	change face, size and color of characters
	tt	typewriter text	display text in monospaced (fixed-width)
	span	span	change appearance using styles
Position	sub	subscript	display text slightly raised
	sup	superscript	display text slightly lowered
List	ul	unordered list	display unnumbered list with bullets
	ol	ordered list	display numbered list
	dl, dt, dd	definition list	display list of term-definition pairs
	li	list item	display an item in an unordered or ordered
Link	a	anchor	create a link or anchor
Table	table	table beginning	start a table
	caption	table caption	display title for a table
	th	table header	display row with headings
	tr	table row	display row with data
	td	table data	display data in each column of a row
Frame	frameset	frameset	define a set of frames to be displayed
	frame	frame	define a web page to be a frame in a
	iframe	inline frame	define a web page to be a frame inside a

Table 7.1 Common HTML Tags(continued)

Form	form	form beginning	start a form
	input	form input	type text in elements or click on elements
	textarea	form textarea	type text in the element, textarea
	select	form select	click on an item in a drop-down list
	fieldset	form fieldset	group together related elements

First web page

By tradition the first program in any programming language displays the message, hello, world. This is the output of the first program in the classic book, The C Programming Language, by Kernighan and Ritchie, first published in 1978.

Web Page 7.2 hello.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system. Start a text editor and type the following file, or copy and paste it, and save it in directory, c07html, using the filename, hello.htm. Then change the permissions of the file to 604.

(Type the following web page)

```
<html>
<head>
  <title> First web page </title>
</head>
<body>
<h1>First web page</h1>
  hello, world
<h2>
<a href="menuhtml.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

(End of the web page)

Start the browser, display the menu for web pages in Figure 7.3 and then click, hello.htm. Alternatively, type the address (address is used in this context to mean URL) of the file, hello.htm, in the address bar of the browser and then press Enter. Note that the title appears in the title bar at the top of the screen. To return to the menu for web pages, click, Menu.

Figure 7.4 shows the display of the web page, hello.htm. Each web page in this section starts with a heading and ends with a link to the menu for the chapter. The heading, First web page, at the top of the page before the message, hello, world, and the link after the message can be commented out so as to display only the message, hello, world.

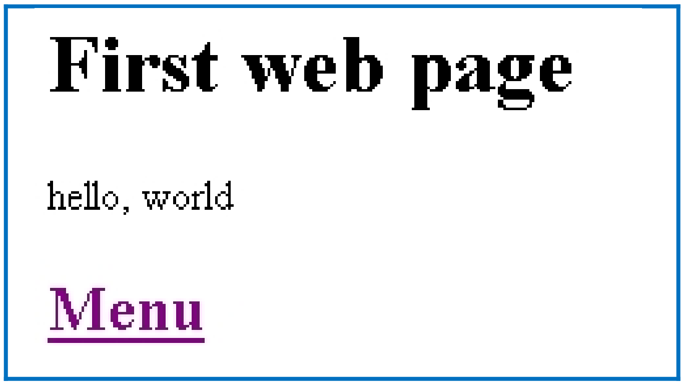
7.10.2 Headers

There are six levels of headers defined in HTML, which are used to organize a web page. The headers are specified by the tags, <h*n*> and </h*n*>, where *n* is a number from 1 to 6. The headers include an automatic new line. Tags are not case sensitive in HTML and so <H*n*> and </H*n*> can also be used. However, tags are case sensitive in XHTML and lowercase must be used.

Web Page 7.3 header.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

Type the following file, or copy and paste it, in an editor and save it in directory, c07html, using the filename, header.htm. Then change the permissions of the file to 604.



First web page

hello, world

Menu

Figure 7.4 First web page: hello.htm

(Type the following web page)

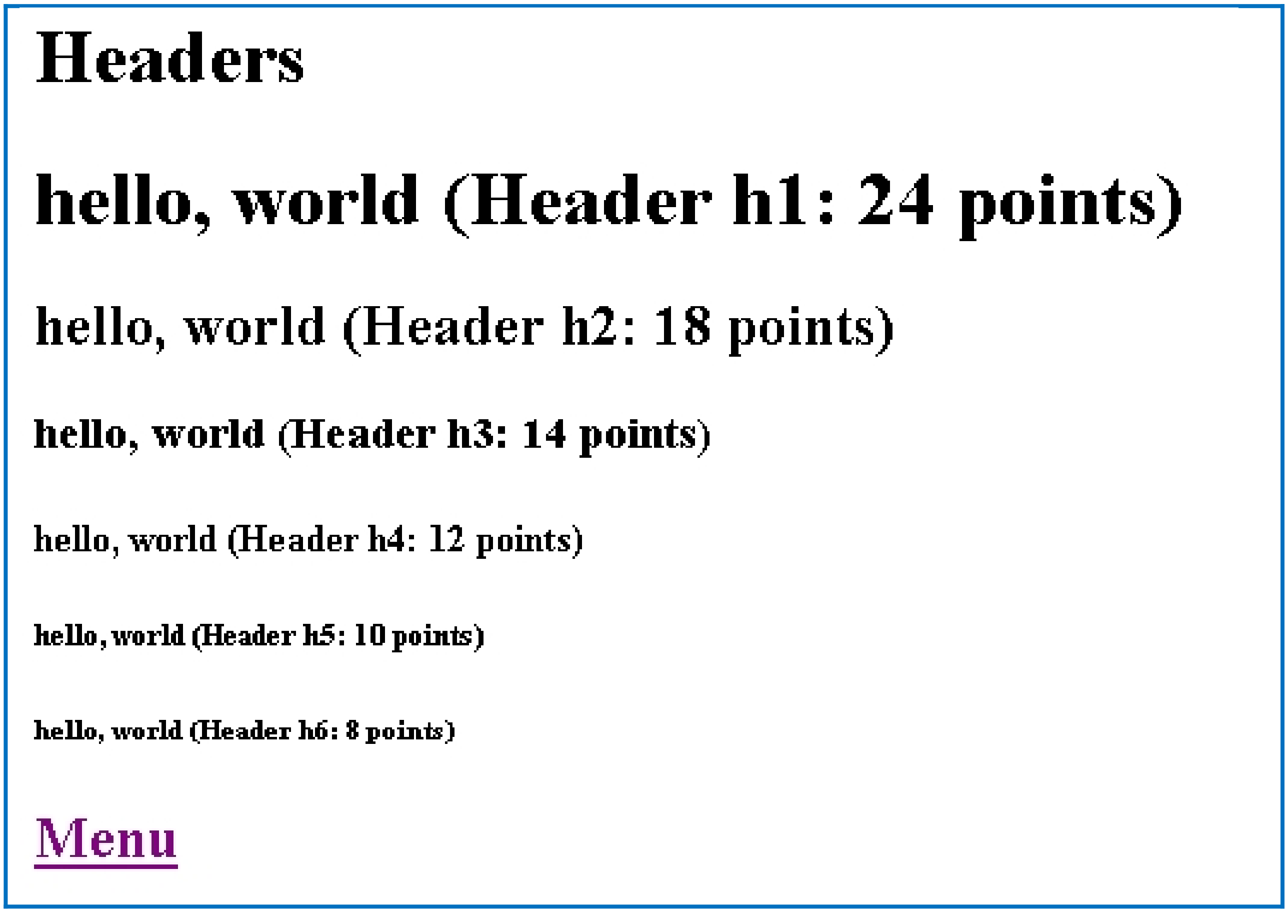
```
<html>
<head>
    <title>Headers</title>
</head>
<body>
<h1>Headers</h1>
<h1>hello, world (Header h1: 24 points) </h1>
<h2>hello, world (Header h2: 18 points) </h2>
<h3>hello, world (Header h3: 14 points) </h3>
<h4>hello, world (Header h4: 12 points) </h4>
<h5>hello, world (Header h5: 10 points) </h5>
<h6>hello, world (Header h6: 8 points) </h6>
<h2>
<a href="menuhtml.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

(End of the web page)

Start the browser, display the menu for web pages in Figure 7.3 and then click, header.htm. Alternatively, type the address of the file, header.htm, in the address bar of the browser and then press Enter.

Note that the smaller the header number, the larger the characters. The size, in points, of each header is given in the web page. In the printing industry the size of type is measured in points. A point is defined as 1/72 inch and so 72 points = 1 inch. Since one inch is defined as 25.4 mm, 3 points = 1 mm, to a good approximation. Obviously, the actual size of characters on a screen depends on the size of the screen and the screen resolution that is used.

Figure 7.5 shows the display of the web page, header.htm.



Headers

hello, world (Header h1: 24 points)

hello, world (Header h2: 18 points)

hello, world (Header h3: 14 points)

hello, world (Header h4: 12 points)

hello, world (Header h5: 10 points)

hello, world (Header h6: 8 points)

Menu

Figure 7.5 Headers web page: header.htm

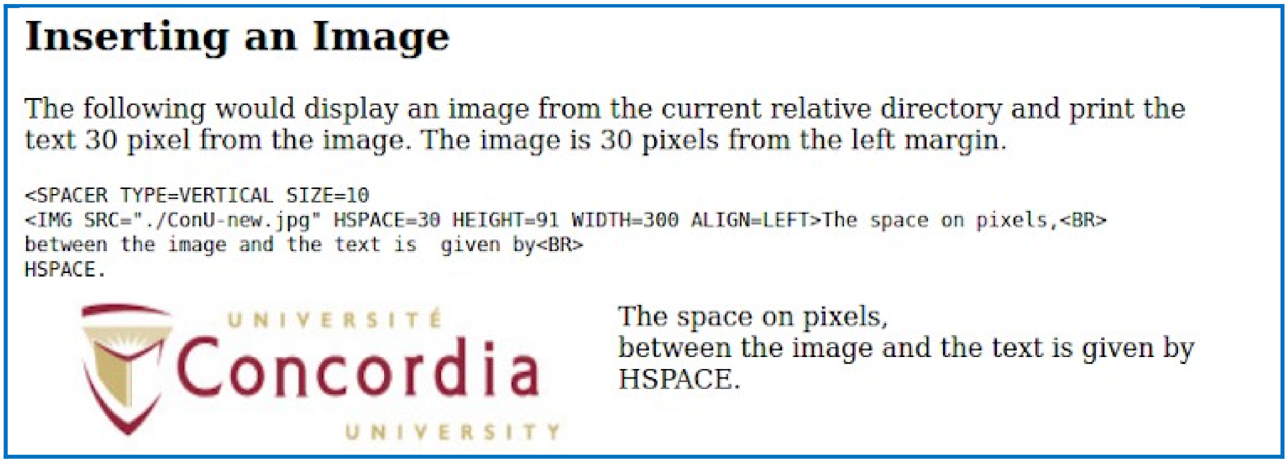
7.10.3 Images

The tag to put an image in a web page is IMG and its use is illustrated in Figure 7.6. A source for the image is specified by the SRC attribute inside the tag. The size, in pixels, of the image to be displayed is given by the HEIGHT and WIDTH attributes. Thus in Figure 7.6 the height of the image is 100 pixels and the width is 400 pixels. The alignment of the image is given by the ALIGN attribute. The alignment can be left, right or center.

Both the code for the web page and, below it, the output of the code is given in Figure 7.6. The file name of the image is, ConUlogo.gif, and the file is in the same directory as the web page itself, as indicated by ./ in front of the file name.

7.10.4 Text

The primary purpose of a markup language is to format the text which is contained in the document. The common text formatting tags are demonstrated here. They are listed in Table 7.1, by category. The categories, spacing, appearance, position and list, are covered in this section. The last four categories, link, table, frame and form, are covered in individual sections after this section.



Inserting an Image

The following would display an image from the current relative directory and print the text 30 pixel from the image. The image is 30 pixels from the left margin.

```
<SPACER TYPE=VERTICAL SIZE=10
<IMG SRC="./ConU-new.jpg" HSPACE=30 HEIGHT=91 WIDTH=300 ALIGN=LEFT>The space on pixels,<BR>
between the image and the text is  given by<BR>
HSPACE.
```

The space on pixels,
between the image and the text is given by
HSPACE.

Figure 7.6 Images

7.10.4.1 Spacing

The main unexpected difference (to beginners) between web pages displayed by a browser and text documents displayed by an editor is about spacing. This difference concerns whitespace. Whitespace in a document means two or more spaces, one or more tabs and one or more newlines. The whitespace in the source file of a web page is not displayed by a browser. Instead, two or more spaces and tabs are converted to one space by the browser. A newline is converted to a space if the newline is typed after a character which is not a space. Otherwise newlines are ignored by the browser.

The whitespace in text can be preserved by enclosing the text in the tags, <pre> and </pre>, but this method is not usually used to arrange the layout of a web page. Instead, the whitespace is created with the spacing tags in Table 7.1 or by using the escape sequence for space in Table 7.3.

The most commonly used spacing tag is <br> (br for break) which gives one newline. This corresponds to pressing, Enter, in a word processor. The tag, <p>, gives two newlines after the text that follows the tag, corresponding to pressing, Enter, twice in a word processor.

The tag, <center>, centers one or more lines on a page. This capability is also furnished by the attribute, align, of other tags, for example, <p> and <div>.

The tag, <div>, divides text into sections (also called blocks) which can be aligned by the attribute, align. The alignment can be left (default), center, right and justify, where justify aligns text to both the left and right margins. A newline is added before and after the text.

The tag, <blockquote>, indents one or more lines of text (a block of text), which is the format for a quotation. Two newlines are added both before and after the text.

7.10.4.2 Appearance and Position

The appearance of text is determined primarily by the font which has three characteristics: style (or face), size and color of text. For example, the style and size of the text in this paragraph is Times New Roman, 12 point. There are two kinds of fonts: regular (proportionally-spaced) and monospaced (fixed-width). The default regular font is Times and the default monospaced font is Courier. The tag, tt, displays text in a fixed-width font.

The header tag, <h*n*>, where *n* = 1 to 6, is equivalent to <font> with certain attributes followed by <p>. So a header tag specifies both spacing and appearance.

The other tags in Table 7.1, bold, italics, underline and strike, produce an appearance that is effectively superimposed on the appearance of the font tag. For example a font of any face, size and color can be bold. The tag, span, uses styles to change the appearance of part of the text enclosed by another tag. Span is demonstrated in the next chapter.

Table 7.2 Escape Sequences

Character		Escape Sequence	
Symbo l	Name	symbolic	numeric
<	less than	&lt;	&#60;
>	greater than	&gt;	&#62;
&	ampersand	&amp;	&#38;
	space	&sp;	&#32;
	nonbreaking space	&nbsp;	&#160;

The tags in category, position, in Table 7.1 are tags which produce subscripts, text that appears below the previous characters by a distance of about half a character, and superscripts, text that appears above the previous characters by the same distance as for a subscript. Subscripts and superscripts have the same size as the previous text, unlike in a word processor where the font size of the subscripts and superscripts is smaller than the preceding text.

7.10.4.3 Escape sequences

An escape sequence in HTML is a series of characters which is an instruction to display a particular character on the screen. Three characters have a special meaning in HTML: <, > and &. The < and > enclose tags and the & is the beginning of an escape sequence. The characters, <, > and &, are command characters rather than data (content) in HTML and do not appear in the output of the browser. These characters, themselves, are inserted into HTML by removing their special meaning. This is done by using escape sequences for the characters. The escape sequence for a special character is used to display the special character as ordinary text. The escape sequences for, <, > and & are shown in Table 7.2. Also, there are many characters not available on a keyboard which can be displayed in HTML with escape sequences of the same form as in Table 7.2. For example, the escape sequence for the french character, é, is &eacute; or &#233;. Search online for, html escape sequences, to find a list of all escape sequences.

The two escape sequences for space- space and nonbreaking space given in Table 7.2. Only one space can be inserted between characters in HTML by pressing the space key on the keyboard. Additional spaces are inserted by using the escape sequence for space. The basic purpose of a nonbreaking space is to connect two words together so that both words appear on the same line. For example, if the text HTML 4 appears at the end of a line, HTML may be displayed at the end of the line and 4 displayed at the beginning of the next line. The nonbreaking space between HTML and 4 will force HTML 4 to appear at the beginning of the next line, that is, on the same line. Also, the nonbreaking space can be used to insert a space anywhere in text, so usually &nbsp; is used where &sp; could be used.

There are two ways to express escape sequences, called character reference and entity reference.

(1) The character reference consists of three parts: ampersand (&), number symbol (#) followed by a decimal number, semicolon (;). The decimal number is the ASCII code for the character. Character reference is called, numeric, for short in Table 7.2.
(2) The entity reference also consists of three parts: ampersand, symbolic name, semicolon. There are symbolic names for only some characters. The symbolic names are case sensitive. Entity reference is called, symbolic, for short in Table 7.2

7.10.4.4 Lists

There are three kinds of lists defined in HTML: unordered, ordered and definition. Lists can be nested, that is, one list can be contained inside another list. In the examples below, the tabs at the beginning of each line are not part of the list. The spacing of items in lists corresponds to spacing in a word processor produced by the use of tab and Enter.

7.10.4.4.1 Unordered - bullet list

The tags used to enclose the unordered list are <ul> and </ul> and each entry in the list is preceded by the tag <li>.

Example of an unordered list:

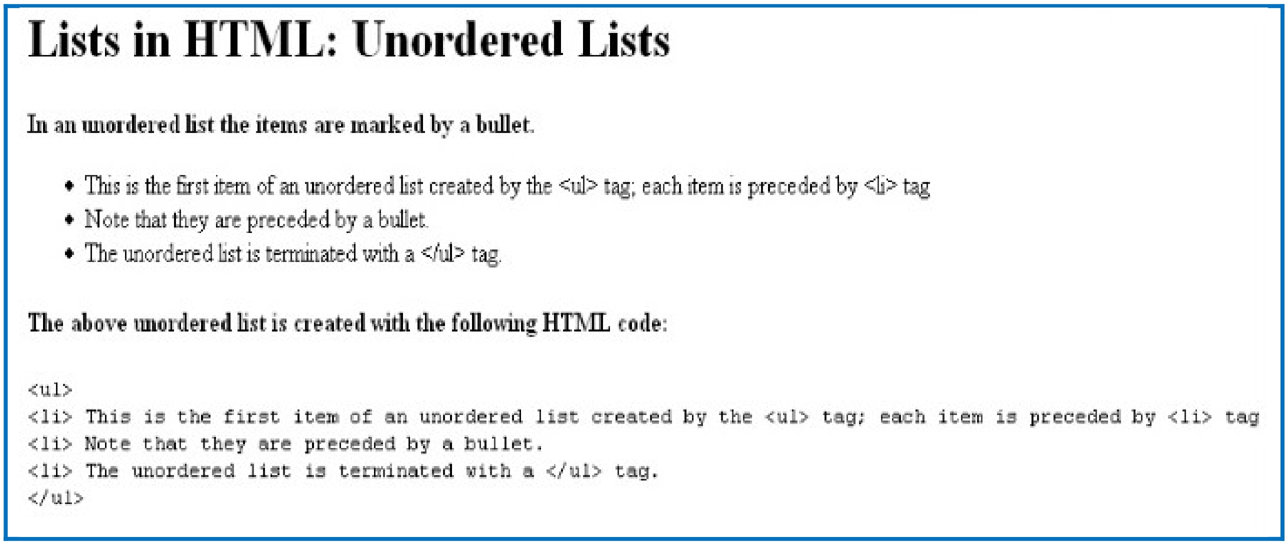
unordered
ordered

definition

7.10.4.4.2 Ordered - numbered list

The tags used to enclose the ordered list are <ol> and </ol> and each entry in the list is preceded by the tag <li>.

Figure 7.7 shows an unordered list and the html code used to produce the list.



Lists in HTML: Unordered Lists

In an unordered list the items are marked by a bullet.

- This is the first item of an unordered list created by the <ul> tag; each item is preceded by <li> tag
- Note that they are preceded by a bullet.
- The unordered list is terminated with a </ul> tag.

The above unordered list is created with the following HTML code:

```
<ul>
<li> This is the first item of an unordered list created by the <ul> tag; each item is preceded by <li> tag
<li> Note that they are preceded by a bullet.
<li> The unordered list is terminated with a </ul> tag.
</ul>
```

Figure 7.7 Unordered list

Example of an ordered list:

1. unordered
2. ordered
3. definition

Figure 7.8 shows an ordered list and the html code used to produce the list.

7.10.4.4.3 Definition - list of term-definition pairs

The tags used to enclose the definition list are <dl> and </dl>. The term is enclosed by the tags <dt> and </dt> with the corresponding definition enclosed by the tags <dd> and </dd>.

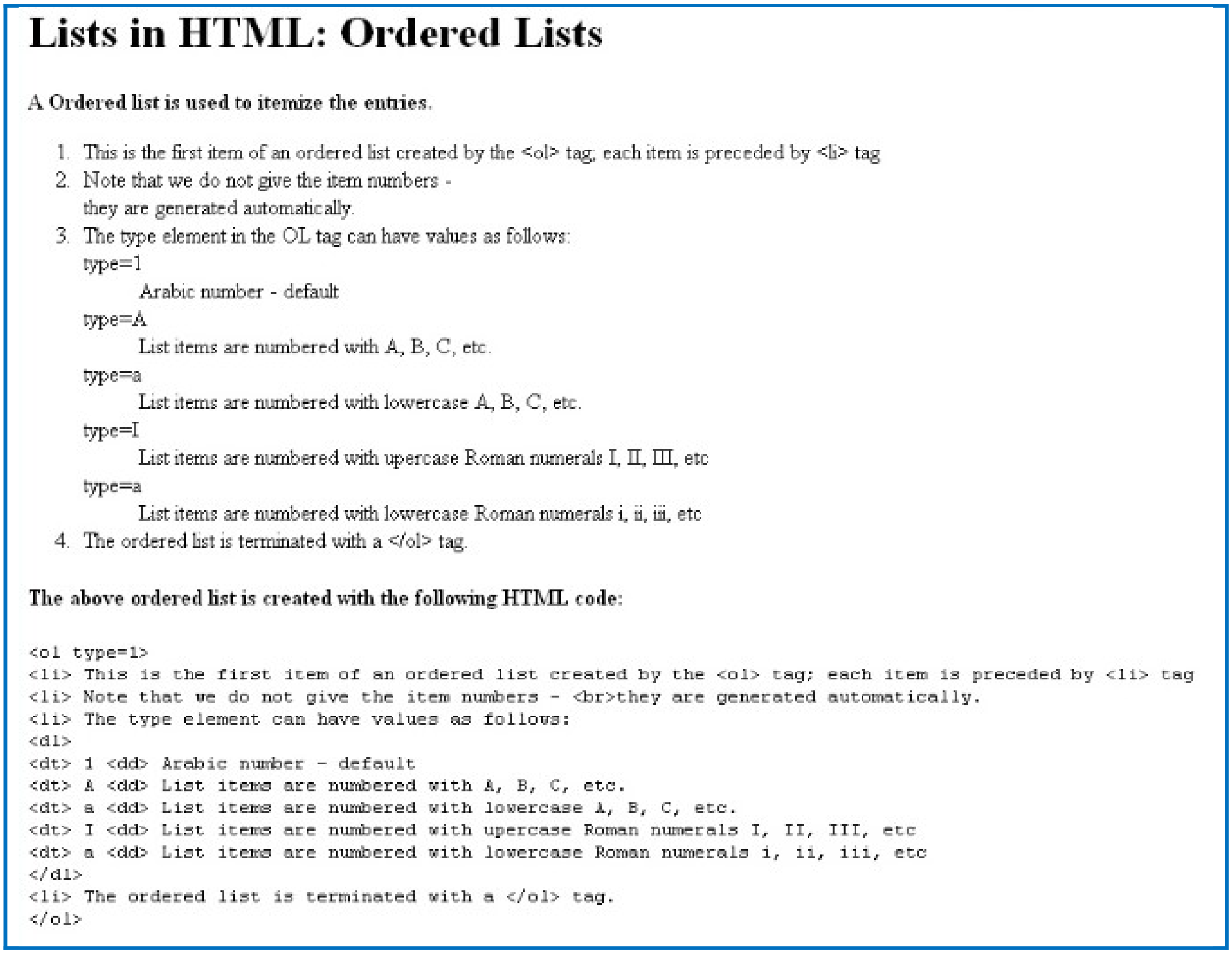
Example of a definition list:

unordered
    list element is marked by a bullet
ordered
    list element is marked by a numbered
definition
    list element is given as a term-definition pairs

Figure 7.9 shows a definition list and the html code used to produce the list.

Web Page 7.4 text.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.
Type the following file, or copy and paste it, in an editor and save it in directory, ch07, using the filename, text.htm. Then change the permissions of the file to 604.



Lists in HTML: Ordered Lists

A Ordered list is used to itemize the entries.

1. This is the first item of an ordered list created by the <ol> tag; each item is preceded by <li> tag
2. Note that we do not give the item numbers -
   they are generated automatically.
3. The type element in the OL tag can have values as follows:
   type=1
       Arabic number - default
   type=A
       List items are numbered with A, B, C, etc.
   type=a
       List items are numbered with lowercase A, B, C, etc.
   type=I
       List items are numbered with upercase Roman numerals I, II, III, etc
   type=a
       List items are numbered with lowercase Roman numerals i, ii, iii, etc
4. The ordered list is terminated with a </ol> tag.

The above ordered list is created with the following HTML code:

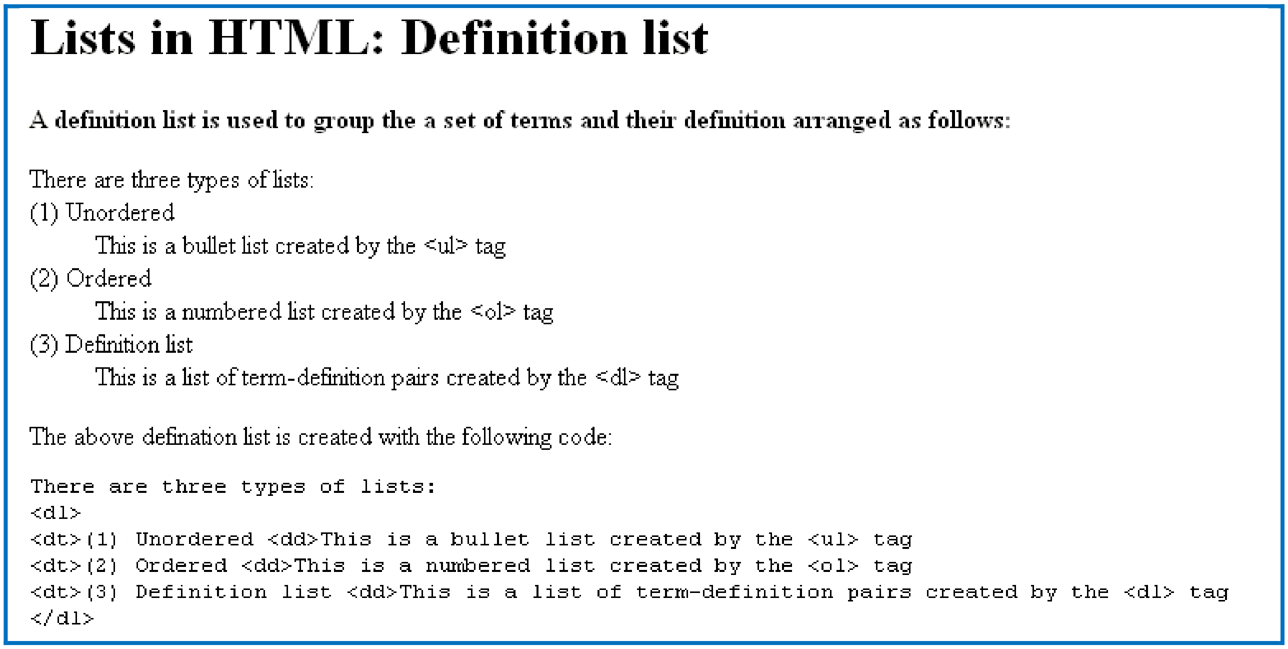
```
<ol type=1>
<li> This is the first item of an ordered list created by the <ol> tag; each item is preceded by <li> tag
<li> Note that we do not give the item numbers - <br>they are generated automatically.
<li> The type element can have values as follows:
<dl>
<dt> 1 <dd> Arabic number - default
<dt> A <dd> List items are numbered with A, B, C, etc.
<dt> a <dd> List items are numbered with lowercase A, B, C, etc.
<dt> I <dd> List items are numbered with upercase Roman numerals I, II, III, etc
<dt> a <dd> List items are numbered with lowercase Roman numerals i, ii, iii, etc
</dl>
<li> The ordered list is terminated with a </ol> tag.
</ol>
```

Figure 7.8 Ordered list

(Type the following web page)

```
<html>
<head>
    <title> Text</title>
</head>
<body>
<h1>Text</h1>
<h3>An ordered list is used to group the text formatting tags
which follow.</h3>
<ol> <!-- default is type=1: arabic numbers (1, 2, . . .) -->
<!--
*****1. Spacing ***** -->
```

```
<li>Spacing
<ol type=i> <!-- i for small roman numerals -->
<li> paragraph: p
<p>This is paragraph 1.
<p>This is paragraph 2.
<p>
<li> line break: br
<br>Mary had a little lamb.
<br>Its fleece was white as snow.
<li> preformatted: pre
<br>This
  is not
     preformatted
```



Lists in HTML: Definition list

A definition list is used to group the a set of terms and their definition arranged as follows:

There are three types of lists:
(1) Unordered
    This is a bullet list created by the <ul> tag
(2) Ordered
    This is a numbered list created by the <ol> tag
(3) Definition list
    This is a list of term-definition pairs created by the <dl> tag

The above defination list is created with the following code:

```
There are three types of lists:
<dl>
<dt>(1) Unordered <dd>This is a bullet list created by the <ul> tag
<dt>(2) Ordered <dd>This is a numbered list created by the <ol> tag
<dt>(3) Definition list <dd>This is a list of term-definition pairs created by the <dl> tag
</dl>
```

Figure 7.9 Definition List

```
<pre>
This
    is
        preformatted.
</pre>
<li> center: center
<center>This is centered</center>
<li> division: div
<div> This is left justified </div>
<div align=center> This is centered</div>
<div align=right> This is right justified </div>
<li> indent: blockquote
<br>The following is indented,
which is a quotation by a contemporary philosopher.
```

```
      <blockquote>
      You can=t always get what you want,<br>
      but if you try sometime<br>
      you just might find<br>
      you get what you need
      </blockquote>
      <li> horizontal rule: hr
      <hr>
      </ol>
<!--
*****2. Appearance ***** -->
    <li> Appearance
       <ol type=I>      <!-- I for capital roman numerals -->
       <li> bold: b
       <br>This <b>word</b> is bold.
       <li> italics: i
       <br>This <i>word</i> is italics.
       <li> underline: u
       <br>This <u>word</u> is underlined.
       <li> strike: s
       <br>This <s>word</s> is striked.
       <li> font: font
       <br>This <font face=arial size=6 color=red>word</font>
       <font face=helvitica size=5 color=green>word</font>
       <font face=futura size=4 color=blue>word</font>
       <font face=garamond size=3 color=orange>word</font>
       shows four different fonts.
       <li> typewriter text: tt
       <br><tt>This sentence is typewriter text.</tt>
       </ol type=a>
<!--
*****3. Position ***** -->
    <li> Position
       <ol type=a>      <!-- a for small letters -->
       <li> subscript: sub
       <br>Water is H<sub>2</sub>O.
       <li> superscript: sup
       <br>One kilobyte is 2<sup>10</sup> bytes.
       </ol>
<!--
*****4. Lists ***** -->
    <li> Lists
    <br>There are three kinds of lists: unordered, ordered,
        definition.<br>
       <ol type=A><!-- A for capital letters -->
       <li> unordered list: ul
       <ul>  <!-- default is type=square: bullet is solid square
-->
       <li> Spacing
       <ul type=disc><!-- disc for solid round bullet -->
       <li> paragraph
       <li> line break
       <li> preformatted
       </ul>
       <li> Appearance
       <ul type=circle><!-- circle for empty round bullet -->
       <li> bold
       <li> italics
       <li> underline
       </ul>
       </ul>
       <li> ordered list: ol
       <br>Ordered lists are used for this entire html document.
            <li> definition list: dl
          <br> A definition list is used to group the following.
          <!--
          *****Definition list*****
          *****Three kinds of lists ***** -->
          <dl>
               <dt>(1) Unordered <dd>This is a bullet list
created by the ul tag
               <dt>(2) Ordered <dd>This is a numbered list
created by the ol tag
               <dt>(3) Definition list <dd>This is a list of
               term-definition pairs created by the dl tag
          </dl>
       <li> list item: li
       <br>List items are used for unordered and ordered lists,
       <br> as demonstrated throughout this html document.
       </ol>
  </ol> <!--end of list which starts at beginning of web page-->
<h2>
<a href="menuhtml.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

(End of the web page)

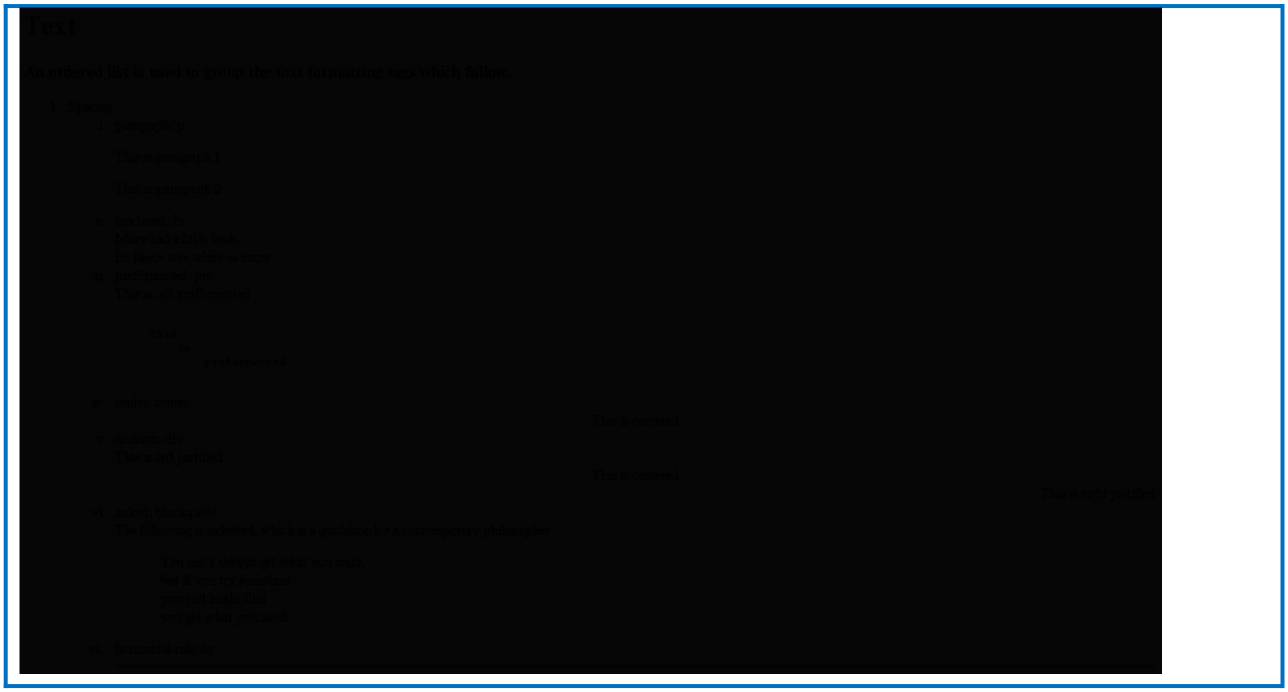
Start the browser, display the menu for web pages in Figure 7.3 and then click, text.htm. Alternatively, type the address of the file, text.htm, in the address bar of the browser and then press Enter.

Figures 7.10, 7.11 and 7.12 show the display of the web page, text.htm.

7.10.5 Links

A link is a connection between two things. A link in a web page is a connection between that web page and another web page. Other terms which mean the same as link in this context are hyperlink, hypertext, and hypertext link. A link is the mechanism of navigation between web pages on different computers on the Internet. The word, web, refers to the intricate pattern of all of the connections between web pages formed by links, which is much more complex than a spider's web.

The tag for a link uses the letter, a, for anchor. Usually a text link is underlined and is blue in color. After a link is clicked the first time it usually changes color.



Text

An ordered list is used to group the text formatting tags which follow.

1. Spacing
   i. paragraph: p

      This is paragraph 1

      This is paragraph 2

   ii. line break: br
      Mary had a little lamb.
      Its fleece was white as snow.
   iii. preformatted: pre
      This is not preformatted

```
        This
           is
              preformatted.
```

   iv. center: center

This is centered

   v. division: div
      This is left justified

This is centered

This is right justified

   vi. indent: blockquote
      The following is indented, which is a quotation by a contemporary philosopher.

> You can't always get what you want,
> but if you try sometime
> you just might find
> you get what you need.

   vii. horizontal rule: hr

Figure 7.10 Text web page, part 1 of 3: text.htm

7.10.5.1 Web pages

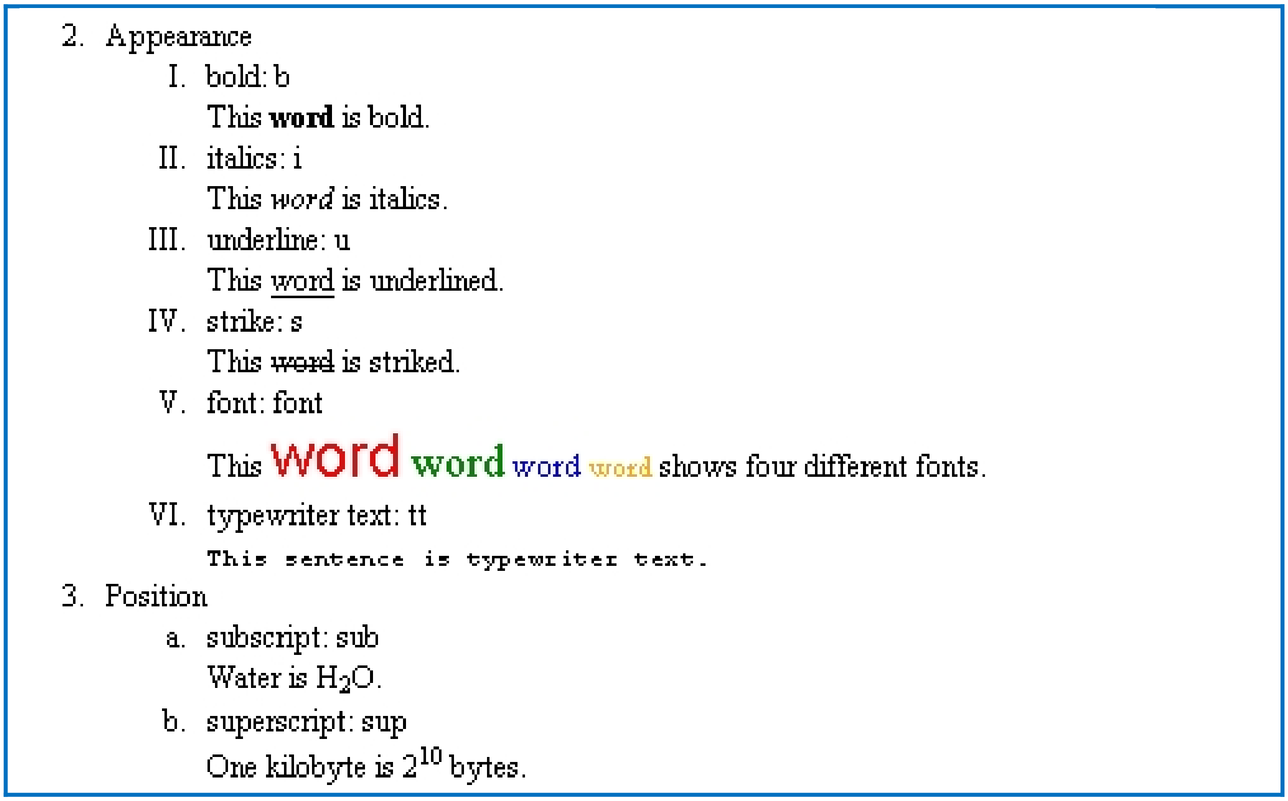
A link to a web page in an HTML file is text or an image which can be clicked to display one of the following web pages.

(a) a new part of the text in the same web page (same file as the link), an internal link, or
(b) another web page (a different file) on the same computer (local host) or
(c) another web page on a different computer elsewhere on the Internet.

7.10.5.2 Email addresses

Most links are to a web page, which are files on a world wide web server, that is, a server that uses http (hypertext transfer protocol). But links can be made to any file on any server as long as it has an address (URL). The most common link besides a link to a web page is a link to an email address on a mail server. This uses the mailto protocol. When the link to an email address is clicked, the browser opens a default email program (for example, Microsoft Outlook Express or Thunderbird) and inserts the destination email address (from the link) and the email address of the sender (user) into the email header. Then the user can type and send an email. For testing purposes, always send an email to yourself.

Web Page 7.5 link.htm
It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.
Type the following file, or copy and paste it, in an editor and save it in directory, ch07, using the filename, link.htm. Then change the permissions of the file to 604. Make sure to type your personal email address in the <a> tag after href=mailto:, instead of, myusername@cs.concordia.ca.



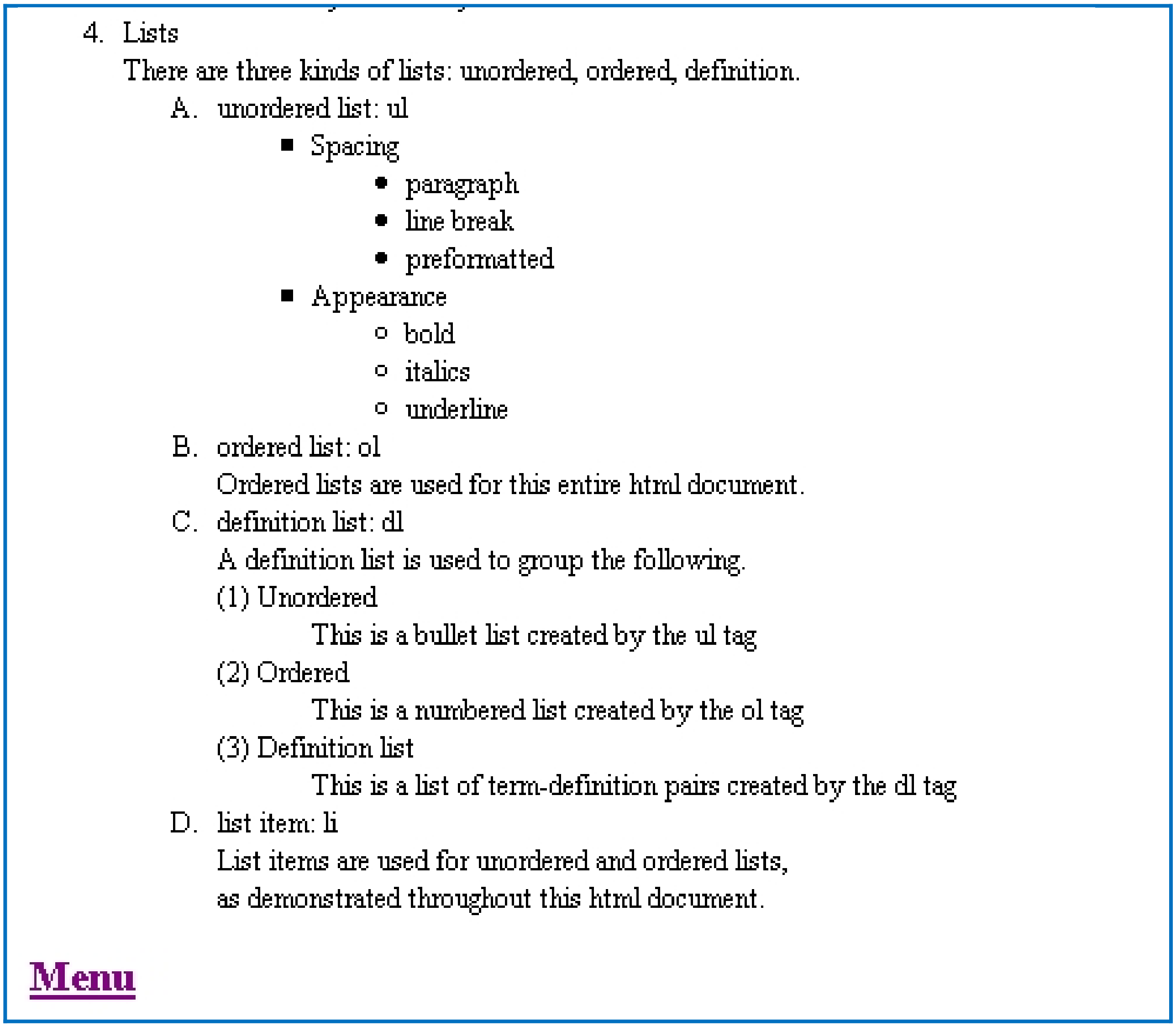
2. Appearance
   I. bold: b
   This **word** is bold.
   II. italics: i
   This *word* is italics.
   III. underline: u
   This word is underlined.
   IV. strike: s
   This ~~word~~ is striked.
   V. font: font
   This word word word word shows four different fonts.
   VI. typewriter text: tt
   `This sentence is typewriter text.`
3. Position
   a. subscript: sub
   Water is H_2O.
   b. superscript: sup
   One kilobyte is 2^{10} bytes.

Figure 7.11 Text web page, part 2 of 3: text.htm

(Type the following web page)

```
<html>
<head>
    <title>Links</title>
</head>
<body>
<h1>Links</h1>
<h3>Click on each link to display the new information.
<br>To return to this page, click on, Back, in the bar above the
address bar.</h3>
<p>
<!--
```

```
*****Start an ordered (numbered) list*****-->
<ol>
```



4. Lists

   There are three kinds of lists: unordered, ordered, definition.

   A. unordered list: ul
      - ■ Spacing
        - ● paragraph
        - ● line break
        - ● preformatted
      - ■ Appearance
        - ○ bold
        - ○ italics
        - ○ underline

   B. ordered list: ol

      Ordered lists are used for this entire html document.

   C. definition list: dl

      A definition list is used to group the following.

      (1) Unordered

         This is a bullet list created by the ul tag

      (2) Ordered

         This is a numbered list created by the ol tag

      (3) Definition list

         This is a list of term-definition pairs created by the dl tag

   D. list item: li

      List items are used for unordered and ordered lists,
      as demonstrated throughout this html document.

Menu

Figure 7.12 Text web page, part 3 of 3: text.htm

```
<!--
*****1.*****-->
<p><li> Link to another part of the same web page
<a href=#bottom>End of web page</a><!--The name of the anchor is,
#end-->
<br>An anchor is created.
<!--
*****2.*****-->
<p><li> Link to another web page on the same (local) computer
<a href=hello.htm>hello.htm</a><!--The relative address is the
filename, hello.htm
```

```
since the file is in the working directory-->
<br>This is a relative address (URL), relative to a directory of
the local computer.
<!--
*****3.*****-->
<p><li> Link to a web page on another computer elsewhere on the
internet
<a
href=http://www.cs.concordia.ca/department/location.html>Computer
Science Department Location</a>
<br>This is an absolute address (URL) of a web page on another
computer.
<!--
*****4.*****-->
<p><li> Link to a web page that does not exist<!--The relative
address, xxx, does not exist-->
<a href=xxx>Nonexistent web page</a>
<!--
*****5.*****-->
<p><li> Link to an email address
<a href=mailto:myusername@cs.concordia.ca>Send email</a>
<!--Change the email address in the link to your email address
for testing purposes-->
</ol>

<h2><a href="menuhtml.htm">Menu</a></h2>
<!--The many line breaks which follow put the anchor for the link
after the top window-->
x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>
x<br>x<br>x<br>x<br>x<br>x<br>
x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>x<br>
x<br>x<br>x<br>x<br>x<br>x<br>
x<br>x<br>
<h3><a name=#bottom>This is the end of this web page</a></h3>
<h2><a href="menuhtml.htm" target=_top>Menu</a></h2>
</body></html>
```

(End of the web page)

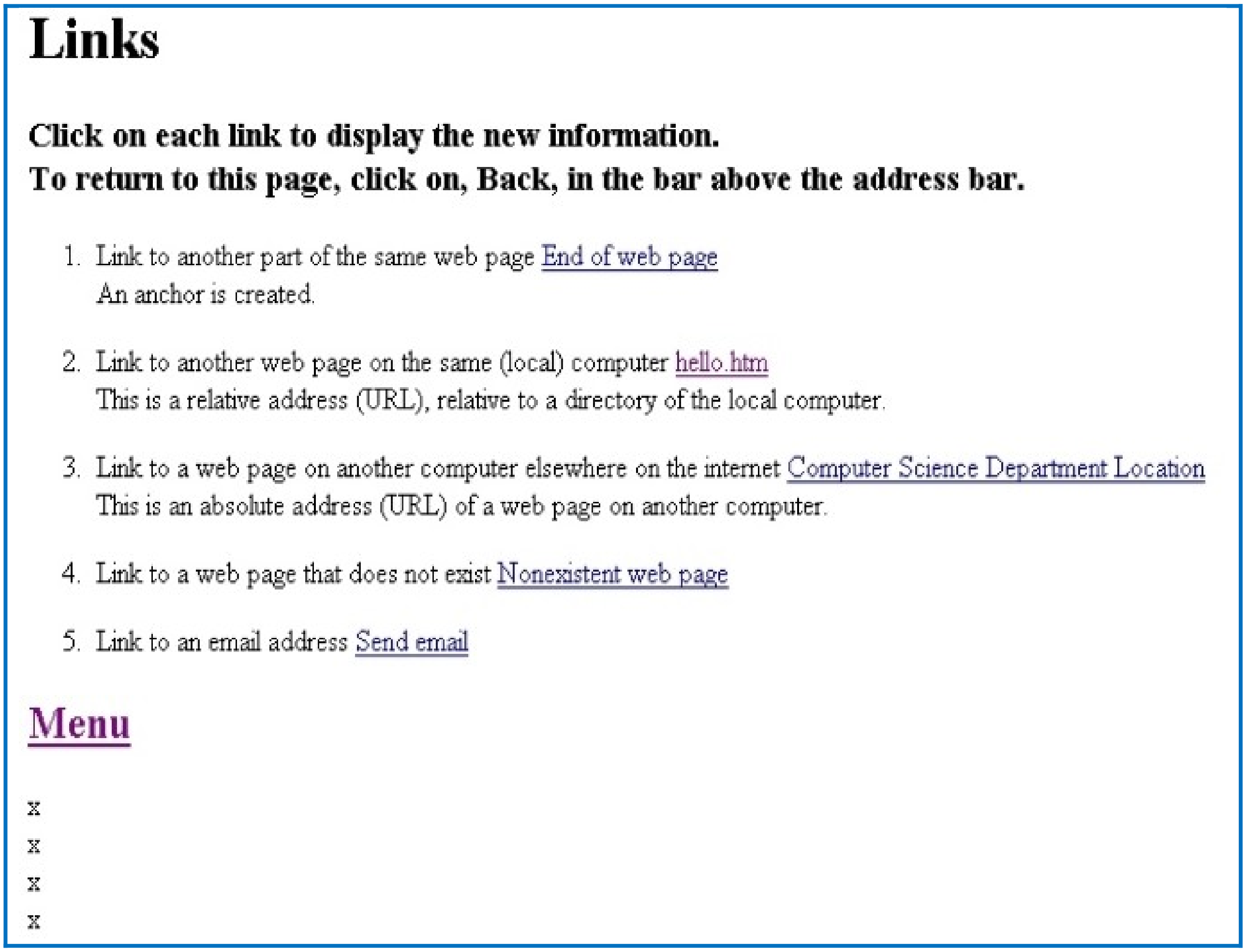
Start the browser, display the menu for web pages in Figure 7.3 and then click, link.htm. Alternatively, type the address of the file, link.htm, in the address bar of the browser and then press Enter.

Figures 7.13 shows the display of the first part of the web page, link.htm.

7.10.6 Tables

Tables consist of rows and columns. The intersection of a row and column is called a cell. The data in a table is stored in the cells of the table. The tags used for a table are <table> and

</table>. Each row of the table is started by the tag, <tr>, and terminated by the tag, </tr>. Each cell in the row (each column of the row) stores the data in the table and is specified by the pair of tags, <td> </td>. Tables can be nested and cells can be combined to create an irregular table.



Links

Click on each link to display the new information.
To return to this page, click on, Back, in the bar above the address bar.

1. Link to another part of the same web page End of web page
   An anchor is created.

2. Link to another web page on the same (local) computer hello.htm
   This is a relative address (URL), relative to a directory of the local computer.

3. Link to a web page on another computer elsewhere on the internet Computer Science Department Location
   This is an absolute address (URL) of a web page on another computer.

4. Link to a web page that does not exist Nonexistent web page

5. Link to an email address Send email

Menu

x
x
x
x

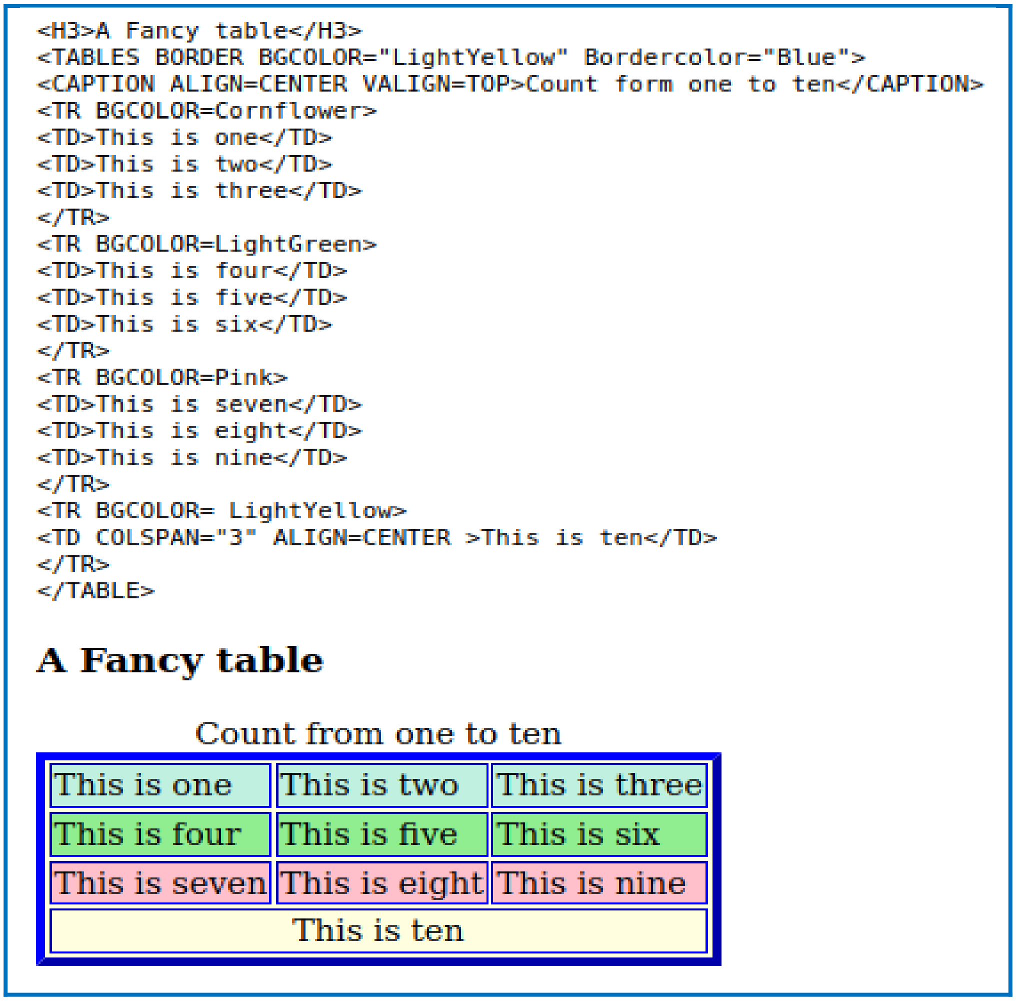
Figure 7.13 Link web page: link.htm

A table is used for two different purposes in a web page. A table is used to present data which is text, like the tables in a word processing document, which is the traditional (or original) purpose of a table. Secondly, a table is used to structure a web page by arranging the location of images and text on the page.

The character, &nbsp;, is a nonbreaking space that can be used in an empty cell of a table so that the cell is displayed as if it was not empty.

Web Page 7.6 table.htm
It is assumed that the user has remotely logged-in to the server and is using the Linux operating system. Type the following file, or copy and paste it, in an editor and save it in directory, c07html, using the filename, table.htm. Then change the permissions of the file to 604.

Figure 7.14 shows the html code to produce a table with three rows and three columns. The output of the code, which is a display of the table, is shown after the html code. The use of the attribute, BGCOLOR, is demonstrated which specifies the background color of one or more cells of a table.



```
<H3>A Fancy table</H3>
<TABLES BORDER BGCOLOR="LightYellow" Bordercolor="Blue">
<CAPTION ALIGN=CENTER VALIGN=TOP>Count form one to ten</CAPTION>
<TR BGCOLOR=Cornflower>
<TD>This is one</TD>
<TD>This is two</TD>
<TD>This is three</TD>
</TR>
<TR BGCOLOR=LightGreen>
<TD>This is four</TD>
<TD>This is five</TD>
<TD>This is six</TD>
</TR>
<TR BGCOLOR=Pink>
<TD>This is seven</TD>
<TD>This is eight</TD>
<TD>This is nine</TD>
</TR>
<TR BGCOLOR= LightYellow>
<TD COLSPAN="3" ALIGN=CENTER >This is ten</TD>
</TR>
</TABLE>
```

A Fancy table

Count from one to ten

This is one	This is two	This is three
This is four	This is five	This is six
This is seven	This is eight	This is nine
This is ten		

Figure 7.14 Code for table and table with background colours

(Type the following web page)

```
<html>
<head>
    <title>Tables</title>
</head>
<body>
<h1>Tables</h1>
<p>
<!--
*****Table with 4 rows and 2 columns.
*****All cells are the same size, the intersection of one row and
one column.
```

```
-->
<h3>Table 1 has 4 rows and 2 columns
<br>All cells have the same size</h3>
<table align=center border=2 cellpadding=3 cellspacing=4>
<!--
align table in center of window
border width of table = 2 pixels
space between cell contents and edge of cell = 3 pixels
space between cells = 4 pixels-->
<caption><h3>Table 1 Escape Sequences</h3></caption>
<tr><!--table row-->
<th> Special Character </th><!--table heading-->
<th> Escape Sequence </th>
</tr>
<tr>
<td align=center> &lt; </td><!--table data-->
<!--align cell contents in center of cell-->
<td> <span>&</span>lt; </td>
</tr>
<tr>
<td align=center> &gt; </td>
<td> <span>&</span>gt; </td>
</tr>
<tr>
<td align=center> &amp; </td>
<td> <span>&</span>amp; </td>
</tr>
<!--
*****Table with 5 rows and 3 columns.
*****All cells are not the same size.
*****One cell occupies one row and two columns,
*****one cell occupies two rows and one column,
*****the other cells occupy one row and one column.
-->
</table>
<br><br><br>
<h3>Table 2 has 5 rows and 3 columns
<br>Not all cells have the same size</h3>
<table cols=3 rows=5 align=center border=2 cellpadding=6
cellspacing=4>
<!--
align table in center of window
border width of table = 2 pixels
space between cell contents and edge of cell = 3 pixels
space between cells = 4 pixels-->
<caption><font size=5>Table 2 &nbsp;&nbsp;Escape
Sequences</font></caption>
```

```
<!--&nbsp; is nonbreaking space which inserts one space-->
<tr><!--table row-->
<th colspan=2> Special Character </th><!--table heading: occupies
2 columns-->
<th rowspan=2 valign=top> Escape Sequence </th><!--table heading:
occupies 2 rows-->
</tr>
<tr><!--table row-->
<th> Symbol </th><!--table heading-->
<th> Name </th>
</tr>
<tr>
<td align=center> &lt; </td><!--table data-->
<!--align cell contents in center of cell-->
<td>less than</td>
<td> <span>&</span>lt; </td>
</tr>
<tr>
<td align=center> &gt; </td>
<td>greater than</td>
<td> <span>&</span>gt; </td>
</tr>
<tr>
<td align=center> &amp; </td>
<td>ampersand</td>
<td> <span>&</span>amp; </td>
</tr>
</table>
<h2>
<a href="menuhtml.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

(End of the web page)

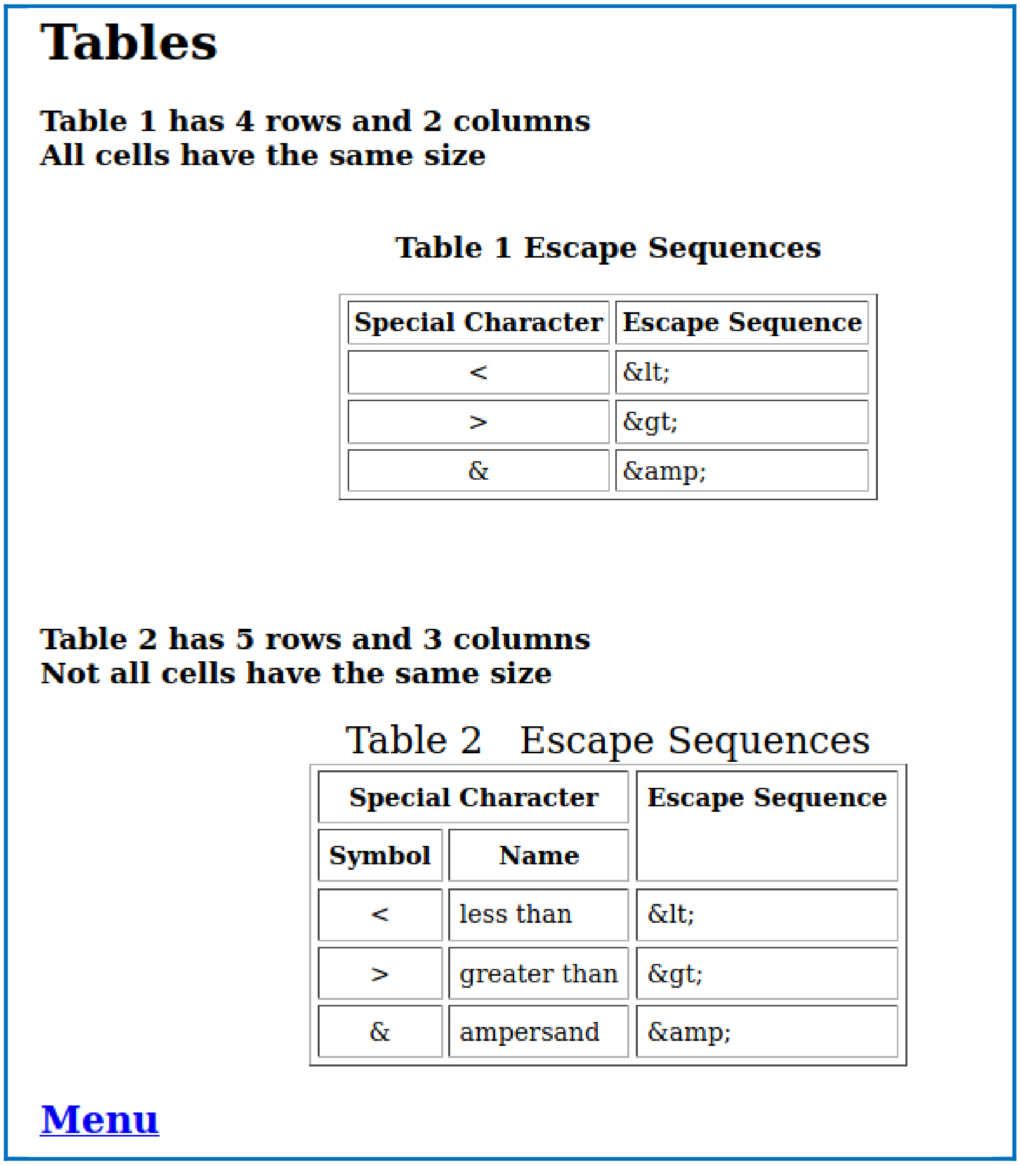
Start the browser, display the menu for web pages in Figure 7.3 and then click, table.htm. Alternatively, type the address of the file, table.htm, in the address bar of the browser and then press Enter.

Figures 7.15 shows the display of the web page, table.htm.

7.10.7 Frames

A frame is a web page that is displayed on only part of the screen. More precisely, a frame is the part of the space on the screen in which a web page is displayed, analogous to the part of the space on a wall of a room in which a window is installed.
There are two kinds of frames.

(a) The entire screen can be occupied by two or more frames. When the entire screen is divided into frames, the frames are specified and positioned by a frameset in a separate web page which replaces the body in that web page. The frames are defined by the tag, frame, inside a frameset. Sometimes a frame inside a frameset is called a normal frame to distinguish it from an inline frame, which is the other kind of frame.



Tables

Table 1 has 4 rows and 2 columns
All cells have the same size

Table 1 Escape Sequences

Special Character	Escape Sequence
<	&lt;
>	&gt;
&	&amp;

Table 2 has 5 rows and 3 columns
Not all cells have the same size

Table 2 Escape Sequences

Special Character		Escape Sequence
Symbol	Name	
<	less than	&lt;
>	greater than	&gt;
&	ampersand	&amp;

Menu

Figure 7.15 Table web page, table.htm

(b) Part of the screen can have one or more frames. In this case the frame is called an inline frame or floating frame, and the frames are defined by the tag, iframe. An inline frame is a web page which is embedded inside another web page which occupies part of or all of the entire screen. Space on the screen for an inline frame is allocated in the same way as for an image.

7.10.7.1 Frameset

All of the frames displayed at the same time on the complete screen are defined in another web page by the tag, frameset. The frames are defined inside the frameset by the tag, frame.

The frameset divides the entire screen into rows and columns, like a table divides part of the space on the screen into rows and columns. At the intersection of each row and column is one frame (space occupied by one web page), which corresponds to a cell of a table.
Framesets can be nested, that is, one frameset can be embedded in another frameset, so as to divide an individual row into two or more columns, or to divide an individual column into two or more rows.

The frameset is in a web page which has no body. The body of a web page is replaced by the frameset, since the frames specified by the frameset occupy the screen as does the body of a web page without a frameset. There is no content in the frameset web page; the content is in the frames.

The structure of a web page that contains a frameset follows. Note that the frameset tag replaces the body tag. The required attribute of <frameset> is either rows or cols or both, not shown below.

```
<HTML>
  <HEAD>
    <TITLE>
         <!B- title -->
    </TITLE>
  </HEAD>
  <FRAMESET>
    <!-- frames -->
  </FRAMESET>
</HTML>
```

In all of the following web pages which demonstrate frames, it is assumed that the user has remotely logged-in to the server and is using the Linux operating system. Type the files in an editor and save them in directory, c07html, using the filenames which are given. Then change the permissions of the files to 604.

Web Page 7.7 frameset1.htm
The web page, frameset1.htm, divides the screen into two rows with a frame (web page) in each row.
(Type the following web page)

```
<html>
<head>
<title>Frameset 1</title>
</head>
<!--
Rows:
Two rows are defined.
First row is 25% of the height of the screen.
```

```
Second row is *, which is the rest of screen (75% of height).
Frames:
First row contains frame1.htm
Second row contains frame2.htm
-->
<frameset rows = "25%, *">
  <frame src = frame1.htm>
  <frame src = frame2.htm>
</frameset>
</html>
```

(End of the web page)

Web Page 7.8 frameset2.htm

The web page, frameset2.htm, divides the screen into two rows with a frame (web page) in the first row and another frameset in the second row. The frameset in the second row divides the row into two columns with a frame (web page) in each column.

(Type the following web page)

```
<html>
<head>
<title>Frameset 2</title>
</head>
<frameset rows = "25%, *">
  <!-- First row -->
  <frame src = frame1.htm>
  <!--
  Second row
  Columns:
  Two columns are defined.
  First column is 15% of the width of the screen.
  Second column is *, which is the rest of screen (85% of
width).
  Frames:
  First column contains frame2.htm
  Second column contains frame3.htm
  -->
  <frameset cols = "15%, *">
    <frame src = frame2.htm>
    <frame src = frame3.htm>
  </frameset>
</frameset>
</html>
```

(End of the web page)

Web Page 7.9 frame1.htm

The web page, frame1.htm, is contained in both framesets, frameset1.htm and frameset2.htm.

(Type the following web page)

```
<html>
<head>
<title>Frame 1</title>
</head>

<body bgcolor=HotPink>
<h1>Frame 1
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="menuhtml.htm"
target=_top>Menu</a>
</h1>
This is frame 1.
</body>
</html>
```

(End of the web page)

Web Page 7.10 frame2.htm

The web page, frame2.htm, is contained in both framesets, frameset1.htm and frameset2.htm.

(Type the following web page)

```
<html>
<head>
<title>Frame 2</title>
</head>
<body bgcolor=DarkSeaGreen>
<h1>Frame 2</h1>
This is frame 2.
</body>
</html>
```

(End of the web page)

Web Page 7.11 frame3.htm

The web page, frame3.htm, is contained in frameset, frameset2.htm.

(Type the following web page)

```
<html>
<head>
<title>Frame 3</title>
</head>
<body bgcolor=DarkOrange>
<h1>Frame 3</h1>
This is frame 3.
</body>
</html>
```

(End of the web page)

Web Page 7.12 framesetch07.htm

The web page, framesetch07.htm, has the same structure as frameset2.htm: two rows with the second row divided into two columns. This is a common organization of frames where the first row is a title, the first (left) column of the second row is a menu and the second (right) column of the second row is the contents of web pages selected from the menu. The frame which will display the contents of more than one web page must be named, using the attribute, name. Then this name is used as the value of attribute, target, in the links to the web pages to be displayed in the frame.

(Type the following web page)

```
<html>
<head>
<title>Chapter 7  HTML</title>
</head>
<frameset rows = "10%, *">
  <frame src = title.htm>          <!-- first row -->
  <frameset cols = "20%, *">       <!-- second row -->
    <frame src = menu.htm>
    <frame src = defaultcontents.htm name=contents>
  </frameset>
</frameset>
</html>
```

(End of the web page)

Web Page 7.13 title.htm

The web page, title.htm, is a frame contained in the first row of the frameset, framesetch07.htm.

(Type the following web page)

```
<html>
<head>
<title>Chapter 7 HTML</title>
</head>
<body bgcolor=yellow>
<h1><center>Chapter 7 HTML</center></h1>
</body>
</html>
```

(End of the web page)

Web Page 7.14 menu.htm

The web page, menu.htm, is a frame contained in the second row, first column of frameset, framesetch07.htm. The links to web pages have a target attribute with a value which is the name of the frame where the web page will be displayed.

(Type the following web page)

```
<html>
<head>
<title>Menu</title>
```

```
</head>
<body bgcolor=cyan>
<h1>Menu</h1>
<h3>
<ol>
  <li><a href=frame1.htm target=contents>Frame 1</a></li>
  <li><a href=frame2.htm target=contents>Frame 2</a></li>
  <li><a href=frame3.htm target=contents>Frame 3</a></li>
  <li><a href=frameset1.htm target=contents>Frameset1</a></li>
  <li><a href=frameset2.htm target=contents>Frameset2</a></li>
  <li><a href=framesetch07.htm
target=contents>Framesetch07</a></li>
  <li><a href=inlineframe.htm target=contents>Inline Frame</a></
li>
  <li><a href=hello.htm target=contents>Hello</a></li>
  <li><a href=header.htm target=contents>Header</a></li>
  <li><a href=text.htm target=contents>Text</a></li>
  <li><a href=link.htm target=contents>Link</a></li>
  <li><a href=table.htm target=contents>Table</a></li>
  <li><a href=form.htm target=contents>Form</a></li>
</ol></h3>
<h2>
<a href="menuhtml.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

(End of the web page)

Web Page 7.15 defaultcontents.htm

The web page, defaultcontents.htm, is a frame contained in the second row, second column of frameset, framesetch07.htm. This is the contents of the frame when the frameset is first displayed.

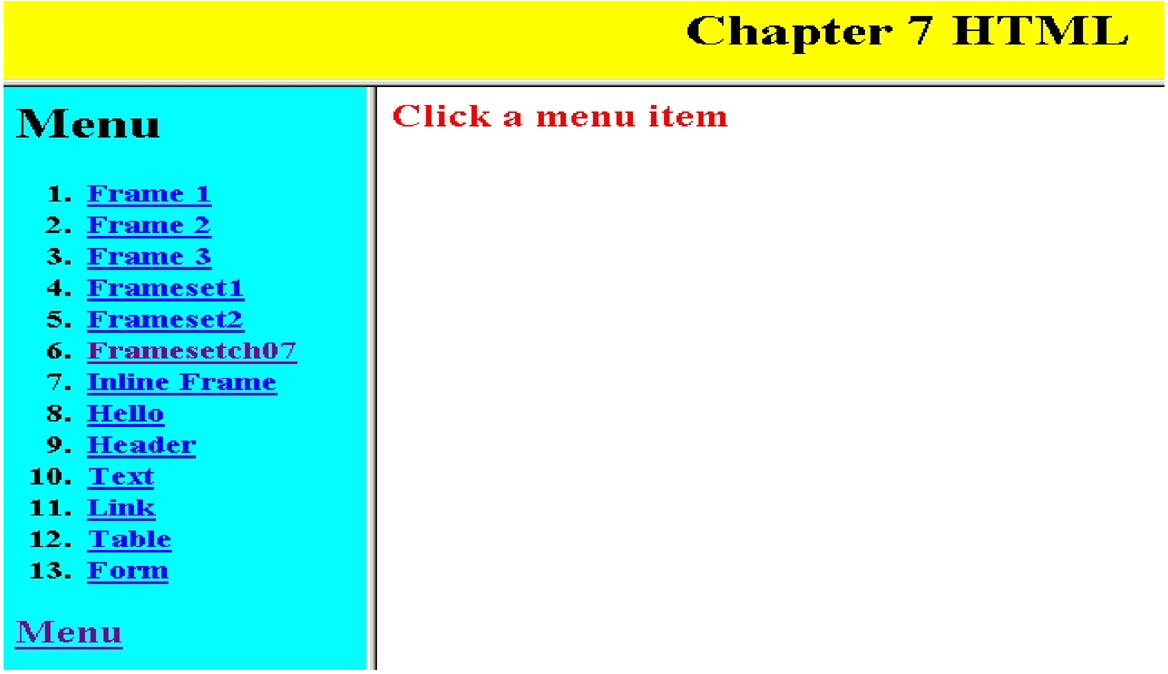
(Type the following web page)

```
<html><head>
<title>Frame Contents</title>
</head>
<body bgcolor=white>
<h2><font color=red>Click a menu item</font></h2>
</body></html>
```

(End of the web page)

In the browser request the web page, frameset1.htm, then frameset2.htm and then framesetch07.htm. Click all of the menu items in framesetch07.htm.

Figures 7.16 shows the display of the web page, framesetch07.htm before any menu items are clicked.



Chapter 7 HTML

Menu

1. Frame 1
2. Frame 2
3. Frame 3
4. Frameset1
5. Frameset2
6. Framesetch07
7. Inline Frame
8. Hello
9. Header
10. Text
11. Link
12. Table
13. Form

Menu

Click a menu item

Figure 7.16 Frameset web page: framesetch07.htm

7.10.7.2 Inline frame

An inline frame is displayed on part of the screen and is defined by the tag, iframe. An inline frame is similar to an image since both are a file which occupy part of a web page. The web page, inlineframe.htm, contains three inline frames which occupy part of the screen.

Web Page 7.16 inlineframe.htm

The web page, inlineframe.htm, contains the three frames, frame1.htm, frame2.htm and frame3.htm.

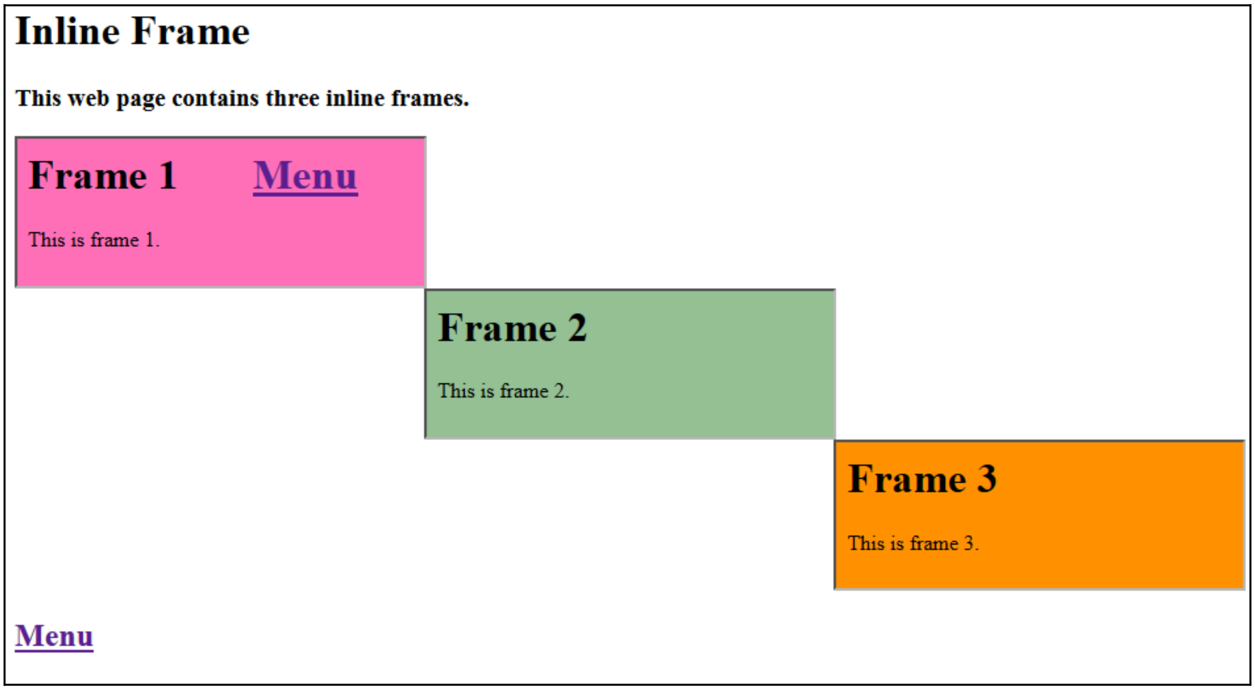
(Type the following web page)

```
<html>
<head>
<title>Inline Frame</title>
</head>
<body>
<h1>Inline Frame</h1>
<h3>This web page contains three inline frames.</h3>
<iframe src="frame1.htm" width=33% height=20%></iframe>
<div align=center>
<iframe src="frame2.htm" width=33% height=20%></iframe>
</div>
<div align=right>
<iframe src="frame3.htm" width=33% height=20%></iframe>
</div>
<h2><a href="menuhtml.htm" target=_top>Menu</a></h2>
```

```
</body>
</html>
```

(End of the web page)

Figures 7.17 shows the display of the web page, inlineframe.htm.



Inline Frame

This web page contains three inline frames.

Frame 1 Menu

This is frame 1.

Frame 2

This is frame 2.

Frame 3

This is frame 3.

Menu

Figure 7.17 Inline frame web page: inlineframe.htm

7.10.8 Forms

A form is the object on a web page by means of which data can be entered by a user from one computer (client) and then sent to another computer (server). The form makes a web page interactive. Essentially from the point of view of the user, the parts of a form are prompts displayed on the screen which ask the user for input from the keyboard or mouse.

The form is the only means by which a user can input data into a web page that is displayed by a browser. The browser both accepts the data from the user and sends that data to another computer. The other computer can be running a web (http) server or mail server. If the data is sent from the form to a web page on a server computer, that web page receives the data and will execute a script that is part of the web page. The script processes the data and may store the data or perform other actions. The most popular scripting language executed by a server computer is currently PHP, which is covered in a later chapter.

The objects on a form in which data can be entered are called elements or controls. The name, element, will be used here. The elements also include a submit button which is clicked to send the data that has been entered in the form and a reset button which is clicked to remove all data that has been entered in the form.

Table 7.3 Elements in a Form

Name	Tag	Description
1. text	input type=text	type data on one line
2. password	input type=password	type data on one line, which is not displayed
3. hidden	input type=hidden	store data not visible to the user
4. text area	textarea	type data on one or more lines, up to about 32 700 characters
5. select	select	select data from a list or drop-down menu (combobox)
6. file upload	input type=file	send file from browser to server
7. radio	input type=radio	select one of a group of buttons
8. checkbox	input type=checkbox	select zero or more of a group of boxes
9. submit	input type=submit	send data from all elements
10. reset	input type=reset	clear data entered in all elements by the user
11. button	input type=button	execute a script

There are 11 different elements that can be used on a form. All of the elements are shown in Table 7.3. There are two basic categories of elements: text and buttons. In Table 7.3 the elements numbered 1 to 6 contain text which is entered by the keyboard (except for select and hidden). In select, text is usually selected by the mouse and in the hidden element text is not accessible by the user. The elements numbered 7 to 11 contain text which is entered by the mouse, by clicking a small circle or box, or a button.

The demonstration html file below, form.htm, does not demonstrate two elements in Table 7.3: input type=file and input type=button. The element, input type=button, is demonstrated in the chapter which covers JavaScript. It runs a script when the button is clicked and so this button essentially executes a command much like icons in a menu when they are clicked.

The form, in the following html file, emails data in the format, *name*=*value*, one name-value pair on a separate line for each element. A form which sends data to another computer for processing by a script will be demonstrated in the chapter which covers PHP.

Web Page 7.17 form.htm

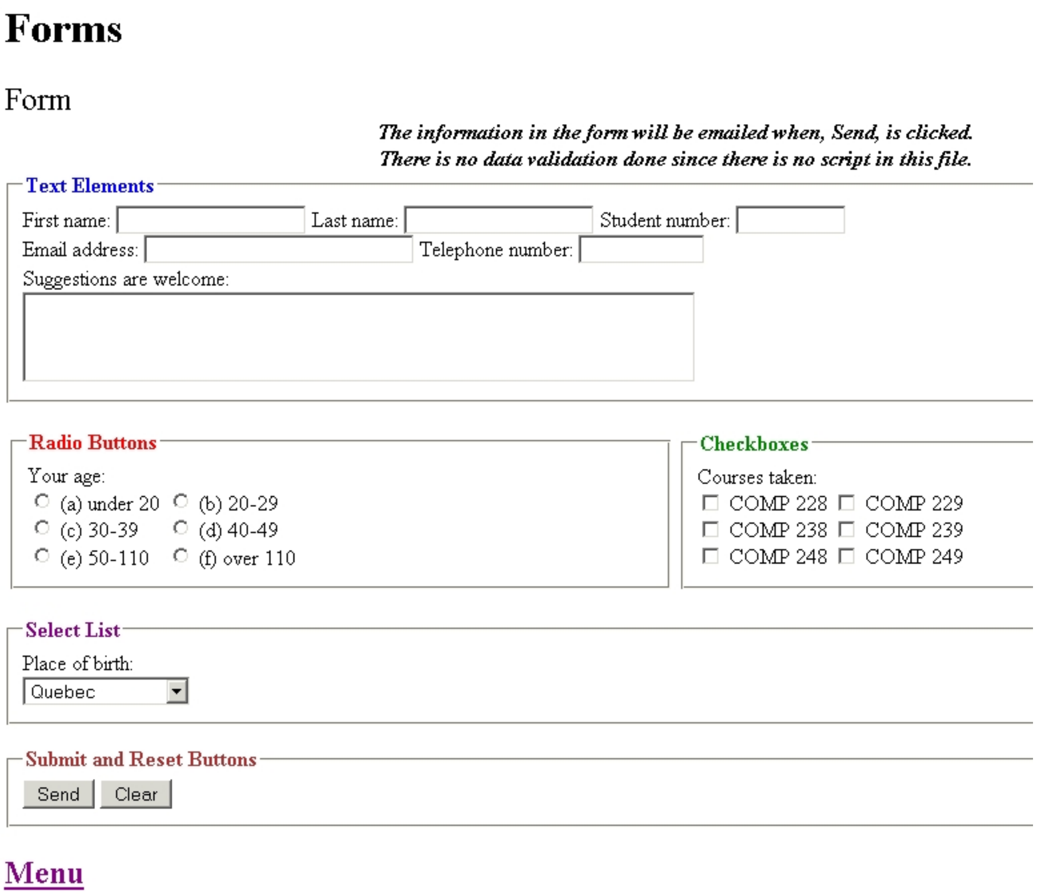
It is assumed that the user has remotely logged-in to the server and is using the Linux operating system. Type the following file, or copy and paste it, in an editor and save it in directory, c07html, using the filename, form.htm. Then change the permissions of the file to 604. Make sure to type your personal email address as the action attribute in the <form> tag, instead of, myusername@cs.concordia.ca.

(Type the following web page)

```
<html>
<head>
    <title>Forms</title>
</head>
<body>
<h1>Forms</h1>
<font size = 5>Form</font>
<center><b><i>The information in the form will be emailed
when, Send, is clicked.<br>There is no data validation done
since there is no script in this file.</i></b></center>
<form method=post enctype="text/plain"
action="mailto:myusername@cs.concordia.ca">
<!--replace the email address, after mailto:, by your email
address-->
<!-- Text Elements -->
<fieldset>
<legend> <font color=blue><b>Text Elements</b></font> </legend>
First name: <input type="text" name="firstname">
Last name: <input type="text" name="lastname">
Student number: <input type="password" name="id" size=10
maxlength=10>
<br>Email address: <input type="text" name="email" size=30>
Telephone number: <input type="text" name="tel" size=12>
<br>Suggestions are welcome:
<br><textarea name="suggestions" rows=3 cols=60></textarea>
</fieldset>
<p>
<table width=100%><tr><td width=50%>
<!-- Radio Buttons -->
<fieldset>
<legend> <font color=red><b>Radio Buttons</b></font> </legend>
Your age:
<br><input type="radio" name="age" value="age&lt;20"> (a) under
20
<input type="radio" name="age" value="age20-29"> (b) 20-29
<br><input type="radio" name="age" value="age30-39"> (c) 30-39
&nbsp;&nbsp;&nbsp;&nbsp;<input type="radio" name="age"
value="age40-49"> (d) 40-49
<br><input type="radio" name="age" value="age50-110"> (e) 50-110
&nbsp;&nbsp;<input type="radio" name="age" value="age&gt;110">
(f) over 110
</fieldset>
</td>
```

```
<td width=50%>
<p>
<!-- Checkboxes -->
<fieldset>
<legend> <font color=green><b>Checkboxes</b></font> </legend>
Courses taken:
<br><input type="checkbox" name="courses" value="c228"> COMP 228
<input type="checkbox" name="courses" value="c229"> COMP 229
<br><input type="checkbox" name="courses" value="c238"> COMP 238
<input type="checkbox" name="courses" value="c239"> COMP 239
<br><input type="checkbox" name="courses" value="c248"> COMP 248
<input type="checkbox" name="courses" value="c248"> COMP 249
</fieldset>
</td></tr></table>
<p>
<!-- Select List -->
<fieldset>
<legend> <font color=purple><b>Select List</b></font> </legend>
Place of birth:
<br><select name="birthplace">
<option value="bc">British Columbia
<option value="ab">Alberta
<option value="sk">Saskatchewan
<option value="mn">Manitoba
<option value="on">Ontario
<option value="qc" selected>Quebec<!--selected indicates the
default-->
<option value="nb">New Brunswick
<option value="ns">Nova Scotia
<option value="nf">Newfoundland
<option value="us">United States
<option value="el">Elsewhere
</select>
</fieldset>
<p>
<!-- Submit and Reset Buttons -->
<fieldset>
<legend> <font color=brown><b>Submit and Reset Buttons</b></font>
</legend>
<input type="submit" value="Send">
<input type="reset" value="Clear">
</fieldset>
</form>
<h2><a href="menuhtml.htm" target=_top>Menu</a></h2>
</body></html>
```

(End of the web page)



Forms

Form

The information in the form will be emailed when, Send, is clicked.
There is no data validation done since there is no script in this file.

Text Elements

First name: Last name: Student number:
Email address: Telephone number:
Suggestions are welcome:

Radio Buttons

Your age:
○ (a) under 20 ○ (b) 20-29
○ (c) 30-39 ○ (d) 40-49
○ (e) 50-110 ○ (f) over 110

Checkboxes

Courses taken:
□ COMP 228 □ COMP 229
□ COMP 238 □ COMP 239
□ COMP 248 □ COMP 249

Select List

Place of birth:
Quebec

Submit and Reset Buttons

Figure 7.18 Form web page, form.htm

Figures 7.18 shows the display of the web page, form.htm.

Make sure that in the script the email address in the action attribute of the form is replaced by your email address.
Start the browser, display the menu for web pages in Figure 7.3 and then click, form.htm. Alternatively, type the address of the file, form.htm, in the address bar of the browser and then press Enter.
Enter some data in the table. Clear the data.
Enter some more data in the table. Send the data. Look at your email.

8 CSS

8.1 Introduction

CSS stands for Cascading Style Sheets. CSS is a part of HTML which is used to describe the presentation of the content of an HTML document. CSS is sometimes regarded as a separate language from HTML since the syntax of CSS is different than for HTML.

CSS is used primarily to separate the two aspects of a web page, content (information in a web page) and presentation (layout of a web page). Content is produced by HTML and presentation by CSS. This separation simplifies the design and maintenance of web pages. Some examples of tags for content are headers, paragraph and lists. Examples of tags used for appearance are font, italics, underline, strike, center. Most tags for appearance are deprecated, meaning that the use of the tags are not recommended because eventually they will not be supported. Instead, CSS should be used.

The two parts of a web page, content and presentation, are analogous to the two basic features of a house, structure and decoration. The structure of a house is like the content of a web page and includes the rooms, and the windows and doors of the rooms. The decoration of a house corresponds to the presentation of a web page and includes the paint on the walls of a room, the floor and window coverings, and the lighting.

Presentation of a web page which uses CSS is done by styles. Styles can be specified in one place in a web page and then the styles can be applied to the entire web page. Also styles can be put in a separate file and then the same styles can be used in more than one web page. An important advantage of using separate files for styles is that styles of web pages can be changed without changing the web pages themselves.

The specification of CSS is defined and maintained by the W3C (World Wide Web Consortium). CSS1 (CSS Level 1) was published in 1996. CSS2 was published in 1998. CSS3 is still under development. Most browsers comply almost fully with the CSS2 standard, while CSS2.1 is a recommendation.

This chapter assumes that the previous chapter, HTML, has been read and understood. XHTML is introduced in the next section.

8.2 XHTML

The HTML standard which was maintained by the WWW Consortium has been ceded to a group consisting of the major vendors of browsers and named Web Hypertext Application Technology Working Group (WHATWG). XHTML (eXtensible Hypertext Markup Language) which was to replace HTML may or may not survive. Its features would be blended into HTML5 which is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2.

XML (eXtensible Markup Language) was used to define XHTML. XML 1.0 is the first version of XML and became a W3C standard in 1998. XML is not a markup language despite its name.

Rather XML is a language that is used to define new markup languages. XML is often described as a data description language and has been used to create a variety of markup languages used for information exchange over the internet, in addition to XHTML. For example, MathML is a markup language which expresses mathematical symbols and formula. Another application of XML is MusicXML which is a markup language which presents music notation. The browser must be able to interpret all of the tags in the markup language which is used.

Almost all of HTML is compatible with XHTML. Assuming that HTML is already understood, XHTML is mastered by learning the differences between HTML and XHTML. The main difference between the two is that the syntax of XHTML is more strict, that is, there are additional rules in XHTML to produce well-formed web pages.

The important differences between HTML and XHTML are presented in the rest of this section. There are other differences besides those that follow.

8.2.1 DTD (Document Type Definition)

There are three different types of XHTML documents: strict, transitional and frameset.
(a) Strict excludes all deprecated tags and attributes.
(b) Transitional includes all deprecated tags and attributes.
(c) Frameset is identical to transitional except the body of a web page is replaced by a frameset.
Deprecated tags and attributes are those in HTML that have been replaced by values of style attributes. Some common tags which are deprecated are <font>, <i>, <u>, <s>, <center>. Some common attributes which are deprecated are bgcolor, align, valign, width, height.

The first line of a web page must specify the DTD that is used to create the page. The declaration of the transitional DTD is the following. This is the DTD that will be used most of the time.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Note that the DTD comes before the tag, <html>, in a web page.

8.2.2 Nesting

Elements must be properly nested. This is done by closing tags in the reverse order as they are opened.

Examples:
Wrong:

```
<b><i>This text is bold and italic</b></i>
```

Right:

```
<b><i>This text is bold and italic</i></b>
```

8.2.3 End tags

All non-empty tags must have a closing tag.

Examples:

Wrong:

```
  <p>This is the first paragraph. <p>This is the second
  paragraph.
```

Right:

```
  <p>This is the first paragraph.</p> <p>This is the second
  paragraph.</p>
```

8.2.4 Empty tags

All empty tags must be closed.
Common empty tags are: <br>, <hr>, <img>, <frame>, <input>. There are other empty tags. Using the break tag for illustration, <br> should be followed by a closing tag and so break would be written,<br> </br>. Fortunately, there is an abbreviation for a closed empty tag, which is for break, <br />. Note that there is an optional space before the / which should always be included to avoid compatibility problems with some browsers.

Examples:
Wrong:

```
  A break <br>
  A horizontal rule <hr>
  An image <img src="bill.gif" alt="Picture of Bill">
```

Right:

```
  A break <br />
  A horizontal rule <hr />
  An image <img src="bill.gif" alt="Picture of Bill" />
```

8.2.5 Case sensitivity

XHTML is case sensitive, whereas HTML is not.
All tags and attributes in XHTML are defined using lowercase letters.

Examples:
Wrong:

```
  <BODY BGCOLOR="red">
  <P>This is a paragraph</P>
  Here's Bill <IMG SRC="bill.gif" ALT="Picture of Bill" />
  </BODY>
```

Right:

```
  <body bgcolor="red">
  <p>This is a paragraph</p>
  Here's Bill <img src="bill.gif" alt="Picture of Bill" />
  </body>
```

8.2.6 Attribute values

Every value of every attribute, including numerical values, must be quoted using double quotes, ".

Examples:

Wrong:

```
   <table align=center border=2 cellpadding=3 cellspacing=4>
```

Right:

```
   <table align="center" border="2" cellpadding="3"
   cellspacing="4">
```

8.2.7 Attributes without values

All attributes must have a value. If an attribute in HTML has no value, the name of the attribute must be used as the value. The value of the attribute makes no difference to the action of the attribute. The presence or absence of a no-value attribute in a tag acts as a boolean value (on/off or yes/no) for the attribute in the tag.

Attributes in forms which have no value are: checked, multiple, selected. There are other attributes in HTML which have no value.

Examples:

Wrong:

```
   <select name="birthplace">
   <option value="on">Ontario</option>
   <option value="qc" selected>Quebec</option>
   </select>
```

Right:

```
   <select name="birthplace">
   <option value="on">Ontario</option>
   <option value="qc" selected="selected">Quebec</option>
   </select>
```

8.3 Styles

A style is a rule that is attached to a tag. The style is interpreted by a browser and the content of the tag is displayed according to the style. Each tag has a number of style properties associated with it. A rule can specify one or more of the style properties which are used to display the contents of the tag.

There are three sources of style information, which the browser will use in general: author, user, user agent.

(a) The author is the programmer who wrote the web page and there are three style methods that an author can use. These methods are covered in this section.

(b) The user can specify a local CSS file which uses options in the web browser. The local CSS file acts as an override of any styles in the web page by the author and are applied to all web pages. For example, visually-impaired users can specify larger text.

(c) The user agent, usually the browser, will apply default style sheets to present elements if none are furnished by the author or user. For the severely visually-impaired, the user agent could be a program to supply an audio representation of a web page, instead of a browser.

There are three methods that an author can use to attach a style to a tag: inline, document-level (also called embedded), external. The three methods are listed in Table 8.1 and described in the rest of this section.

Table 8.1 Style Methods

Name of Method	Location of Styles	Brief Syntax
inline	tag	<*tagname* style="*declarations*">
document-level	head of web page	<style>*rules*</style>
external	separate file	*rules*

8.3.1 Inline

The style attribute is included inside the opening tag. The name of the style attribute is the word, style, and the value of the style attribute is one or more declarations. A declaration is the name of a property associated with the tag and the value of the property.

An inline style mixes content with presentation and should be used only if it is not appropriate to use one of the other two methods. For example, an inline style could be used if a style is applied only once to a particular tag.

Syntax

(a) The syntax of the style attribute is
<*tagname* style="*declaration1*; *declaration2*; . . . ">*content*</*tagname*>
where style="*declaration1*; *declaration2*; . . . " is the style attribute inside the opening tag with tag name, *tagname*.
(b) The syntax of *declaration* is
propertyname:*propertyvalue*

Examples:

```
<body>
...
<p style="color:red">Example1</p>
  <!-- color is the name of a property and red is a value of
  the property, attached to the tag with name, p, and applied
  to the content of the tag, Example1 -->
<p style="color:red; font-weight:bold">Example2</p>
  <!-- two properties are applied in the same tag -->
...
</body>
```

8.3.2 Document-level

The style is a pair of tags, <style> and </style>, which are in the head of the web page. Enclosed in the pair of style tags are one or more rules which are used in the body of the web page. A rule identifies one or more tags to which the rule is applied and one or more declarations, where a declaration is the same as for an inline style.

Syntax

(a) The syntax of a style tag is
<style type="text/css">
rule1
rule2
. . .
</style>
There are other types of style sheets besides CSS and so in the style tag the attribute, type, with value, text/css, should always be used.
(b) The syntax of a rule is
selector1, *selector2*, . . . {*declaration1*; *declaration2*; . . . }
where *selector* is a tag name, and *declaration*, is the same as for an inline style, which is
propertyname:*propertyvalue*
Rules are often written with spacing as follows so that the rule is easier to read.
selector1, *selector2*, . . . {
declaration1;
declaration2;
. . . }

Examples:

```
<head>
...
<style>
/* This is a comment inside the style tag */
p{                /* p is a selector */
  color:blue;     /* color is the name of a property
                  and blue is a value of the property */
}
h1 {
  color:red;
  font-weight:bold
}
h2, h3, h4 {      /* this rule applies to three selectors */
  color:blue;
  font-weight:normal
}
</style>
...
</head>
<body>
...
<h1>This is a heading</h1> <!-- The color of the text is red
                            and the font weight is bold -->
<p>This is the beginning of a paragraph
. . . the end of paragraph</p>   <!-- The color of the text is
                                 blue -->
...
</body>
```

8.3.3 External

One or more rules are contained in a separate file. The file is called a CSS file and has the extension, .css.

The file can be linked to one or more web pages where the styles are applied. A css file is linked by the link tag in the head of a web page.

Syntax

(a) The syntax of the contents of a css file is

rule1
rule2
. . .

where the syntax of *rule* is the same as for the document-level style.

(b) The syntax of the link tag is

<link href="*CSSfilename*" rel="stylesheet" type="text/css" />

Examples

File name: style1.css

File contents:

```
/* This is a comment in a CSS file */
p{                    /* p is a selector */
  color:blue;         /* color is the name of a property and
                      blue is a value of the property */
}
h1 {
  color:red;
  font-weight:bold
}
```

Link of css file in the head of a web page:

```
<head>
...
<link href="style1.css" rel="stylesheet" type="text/css" />
...
</head>
```

8.4 Style classes

Style classes allow

(a) different styles to be applied to the same tag or
(b) the same style to be applied to different tags

for a document-level style or external style.

A class name must be begin with a letter followed by zero or more letters (a-zA-Z), digits (0-9) and hyphens (-). Unlike a selector which must be in lowercase, a class name can be uppercase or lowercase or any combination of the two. Also, the class name is case sensitive and so, for example, the class names, Center and center, are two different class names.

The class attribute is used in tags to apply the style to a tag. The syntax of the class attribute is

class=*classname*

8.4.1 Regular class

A regular class can be used to give different styles to the same tag. In the example below, one paragraph can be centered and another paragraph can be right justified.

Syntax of selector

tagname.classname

Examples for document-level styles

```
<head>
...
<style>
p.center{
  text-align:center
}
p.right{
  text-align:right
}
</style>
...
</head>
<body>
...
<p class="center">The text on each line<br />
of this paragraph is centered.</p>
<p class="right">The text on each line<br />
of this paragraph is right justified.</p>
...
</body>
```

8.4.2 Generic class

A generic class is a class that is not associated with any particular tag. So the same style can be applied to different tags.

Syntax of selector

.classname

Examples for document-level styles

```
<head>
...
<style>
.italic{
  font-style:italic
}
</style>
...
<body>
...
<h2 class="italic">This header is in italics</h2>
```

```
<p class="italic">This paragraph is also in italics</p>
...
</body>
```

8.4.3 Id class

The id class is the same as the regular and generic class except instead of the period, ., the character, #, is used. Also, the id attribute is used in a tag instead of the class attribute. The syntax of the id attribute is

id=*classname*

The id class gives the same results as the regular and generic classes. The regular and generic classes are used most often and are used in this chapter.

Syntax of selector

tagname#classname
#classname

Examples for document-level styles

```
<head>
...
<style>
h1#red{
  color:red
{
#italic{
  font-style:italic
}
</style>
...
<body>
...
<h2 id="red">This header is red.</h1>
<h2 id="italic">This header is in italics</h2>
<p id="italic">This paragraph is also in italics</p>
...
</body>
```

8.4.4 Pseudoclass

Pseudoclass are styles which are displayed for the states of certain tags. The pseudoclass names are predefined.

Three of the predefined pseudoclasses are associated with the tag, <a>, which puts a link in a web page.

Syntax of selector

tagname:*pseudoclassname*
tagname.*classname*:*pseudoclassname*

The three most common pseudoclasses distinguish the three states of a hyperlink, which is tag, <a>.

a:link - not yet visited
a:active - being visited
a:visited - already visited

Another common pseudoclass changes the appearance of a tag when the mouse is over the tag.

tagname:hover

Examples for document-level styles

```
<head>
...
<style>
a:link {color:blue}
a:active {color:red}
a:visited {color:green}
a.hover {color:yellow}
a.important:link{font-weight:bold}
a.important:active{font-size:150%}
a.important:visited{font-weight:normal}
a.important:hover{color:red}
</style>
...
</head>
<body>
...
<a href="webpage1.htm">Ordinary web page</a>
<a href="webpage2.htm" class="important">Special web page</a>
...
</body>
```

8.5 Style precedence

When a style is specified by two or more methods for the same tag, the style which is closer to the tag takes precedence. The word, cascading, in the name, cascading style sheets (CSS), refers to more than one style for the same tag changing (cascading) into one style according to the precedence rules.

Precedence rules:

(1) inline takes precedence over document-level,
(2) document-level takes precedence over external,
(3) external takes precedence over the browser default style and
(4) the browser default style is used when a style is not specified.

8.6 Style properties

Style properties are the means by which the appearance of a web page is generated. Each property has a name and one or more values. The names of most of the style properties are shown in Figure 8.1. There are other style properties in addition to those in the figure. Most of the values permitted for each property in Figure 8.1 are listed as comments in the web demonstrations for each of the categories and some are demonstrated.

There are eight categories of style properties in Figure 8.1. Most of the categories are used in the references listed in the last section of the chapter. However, the category, Other, is not descriptive but is used to limit the number of categories and includes all properties that do not belong in the rest of the categories.

A brief description of each category follows.

8.6.1 Color and Background

The property, color, is the foreground color of the element content, usually text.
The background properties of the element content can be either a color or an image. If the background is an image, the position of the image can be specified.

8.6.2 Font

In the printing industry font means the style and size of type, where style refers to the form or appearance of the type. The meaning is the same here where type is text, but there is no universally accepted list of characteristics of a font. Furthermore, the terminology for the names of properties and values of properties is not standardized. For example, the values, italic and oblique, may or may not be the same.

8.6.3 Text

The text properties affect the position of the text in contrast to the font properties which determine the appearance of the text. For example, text properties specify the spacing between characters in a word and the spacing between words. Also, text properties indent text and align text in a paragraph.

8.6.4 Box

The box properties describe the appearance of a rectangular box with content which is text or an image.
The box model, Figure 8.2, is copied from the W3C CSS2.1 specification, at the web site given in the references in the last section. The box properties are divided into three subcategories in Figure 8.2: border, margin and padding.

8.6.5 List

The list properties format lists by specifying the marker type and the marker position. There are three types of markers: glyph, numeric, alphabetic. There are three types of glyphs: disc, circle, square. The numeric types are: decimal, roman, georgian, armenian. The alphabetic types are: latin (a,b,c,...), greek. A marker can also be an image.

8.6.6 Dimension

The dimension properties control the height and width of the rectangle which encloses the element content. This rectangle is the box which contains the content in Figure 8.2. This rectangle is also called the containing block in the W3C CSS2.1 specification. The dimension properties also include the line height within the element content. If there are lines of text in an element, the line height can be used, for example, to double space the text.

8.6.7 Positioning

The positioning property, position, determines the location of the rectangle (box) which encloses the element content. There are four different position schemes: static (normal), relative, absolute, fixed. In particular, if the value, fixed, is used, the element content does not scroll but remains at a fixed position on the screen. The positioning property, vertical-align, determines the position of text in an element and can be used to produce superscripts and subscripts. The positioning property, z-index, determines the stacking order of boxes. For example, if two boxes overlap, the box that is completely visible (the box on top) is specified by z-index.

8.6.8 Other

The other properties are miscellaneous properties that do not belong in the previous seven categories. The property, cursor, has several values which change the appearance of the cursor. The property, float, positions one box, for example an image, inside another box. The pseudoclass properties give the familiar appearance of links.

8.7 Block and Inline Formatting

Content is grouped as either block-level elements or inline elements. All tags on a web page are either block formatted or inline formatted, but not both at the same time. The main difference between block-level and inline elements is that block-level elements are positioned vertically one after the other and inline elements are positioned horizontally one after the other. These two different qualities give structure to a web page.

Examples of block-level tags are p, h1, h2, . . . h6, blockquote, ul, ol, form. Examples of inline tags are em, strong, cite, a. The tags in the examples present content in some way in addition to block or inline formatting. However, there are two tags which only provide block and inline formatting without affecting the presentation of content: div and span. The tag, div, provides block formatting only. The tag, span, provides inline formatting only.

8.7.1 div

The tag, div, specifies block-level elements. The syntax of an element using div is

<div [*attributes*] >*content*</div>

where the one or more attributes are optional, indicated by the square bracket. Div can be enclosed in block tags (including div) but cannot be enclosed in an inline tag.

The content of the div element, *content*, begins on new line, that is, a line break is inserted by the browser before the block. Also the block ends with a new line, that is, a line break is inserted by the browser after the block. So two or more consecutive block elements are positioned below one another.

The content, *content*, enclosed by div can contain other block tags (including div) or inline tags.

The div tag is used mainly with styles using the class or id attribute. The syntax using attribute, class, is

<div class=*classname* >*content*</div>

Also, inline styles can be used with div using the attribute, style.

There are other attributes, in addition to class, id and style, which can be used with div. The attribute, align, is the most useful of the other attributes. The syntax is

<div align=*alignvalue* >*content*</div>

There are four values for align: left (the default), center, right, justify. The value, justify, aligns both the left and right margins.

There are many styling options that can be used with div. Some are color, font properties, width, height, borders, padding, margins, positioning on web page.

Examples using document-level styles

```
   <head>
   ...
   <style>
   div.center{
     text-align:center
   }
   div.right{
     text-align:right
   }
   </style>
   ...
   </head>
   <body>
   ...
   <div class="center">This block is centered<br />using
styles.</div>
   <div class="right">This block is right justified<br />
             using styles.</div>
   <div align="center">This block is centered<br />
             without using styles.</div>
   <div align="right">This block is right justified<br />
             without using styles.</div>
   ...
   </body>
```

8.7.2 span

The tag, span, specifies inline elements. The syntax of an element using span is

<span [*attribute*s] >*content*</span>

Span can be enclosed inside both block tags and inline tags.

The content of the span element, *content*, begins on the same line as the element, that is, a line break is not inserted by the browser before the element. Also the block ends on the same line, that is, a line break is not inserted by the browser after the element. So two or more consecutive inline elements are positioned beside each other.

CSS Properties

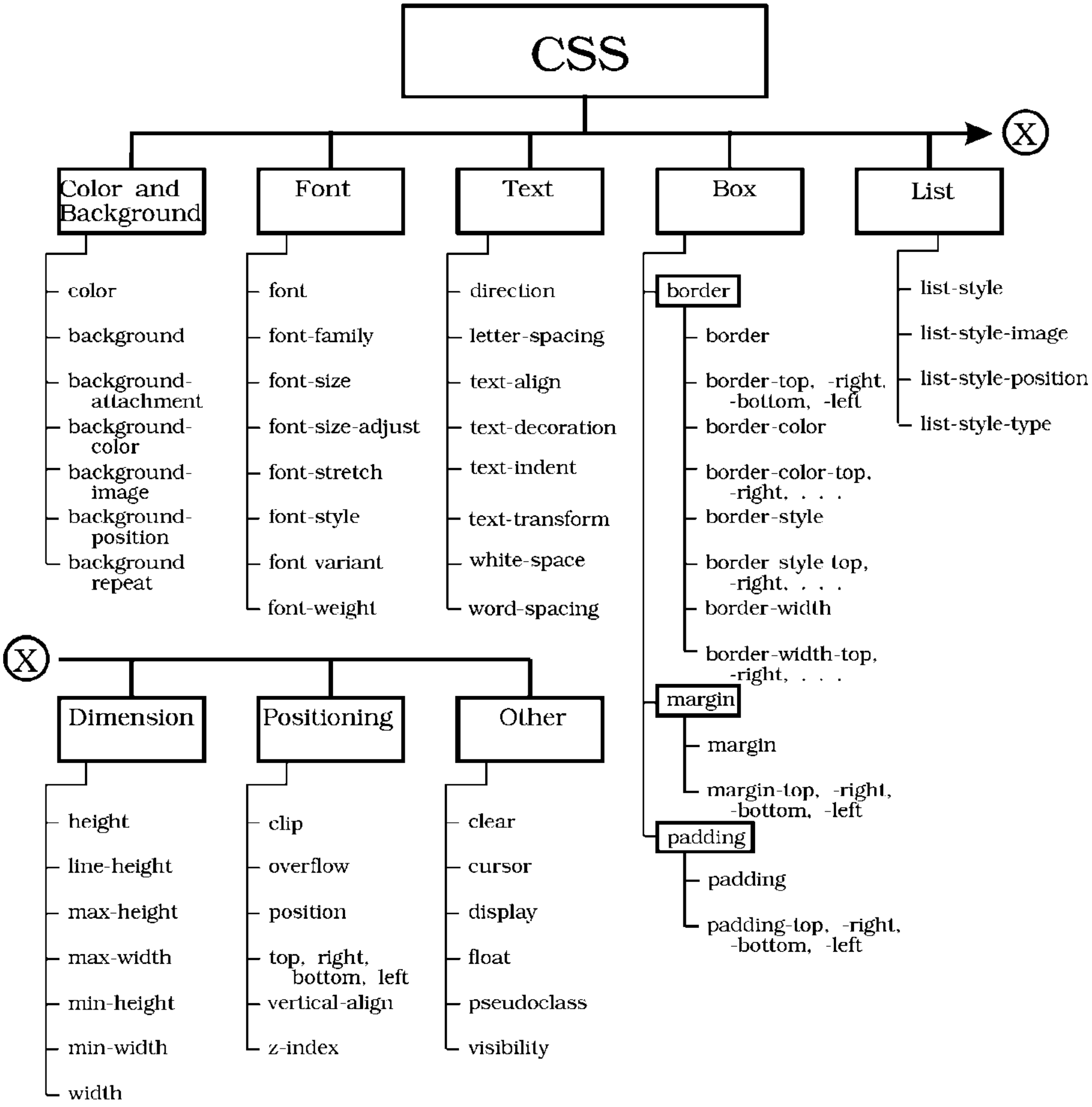


Figure 8.1 CSS Properties

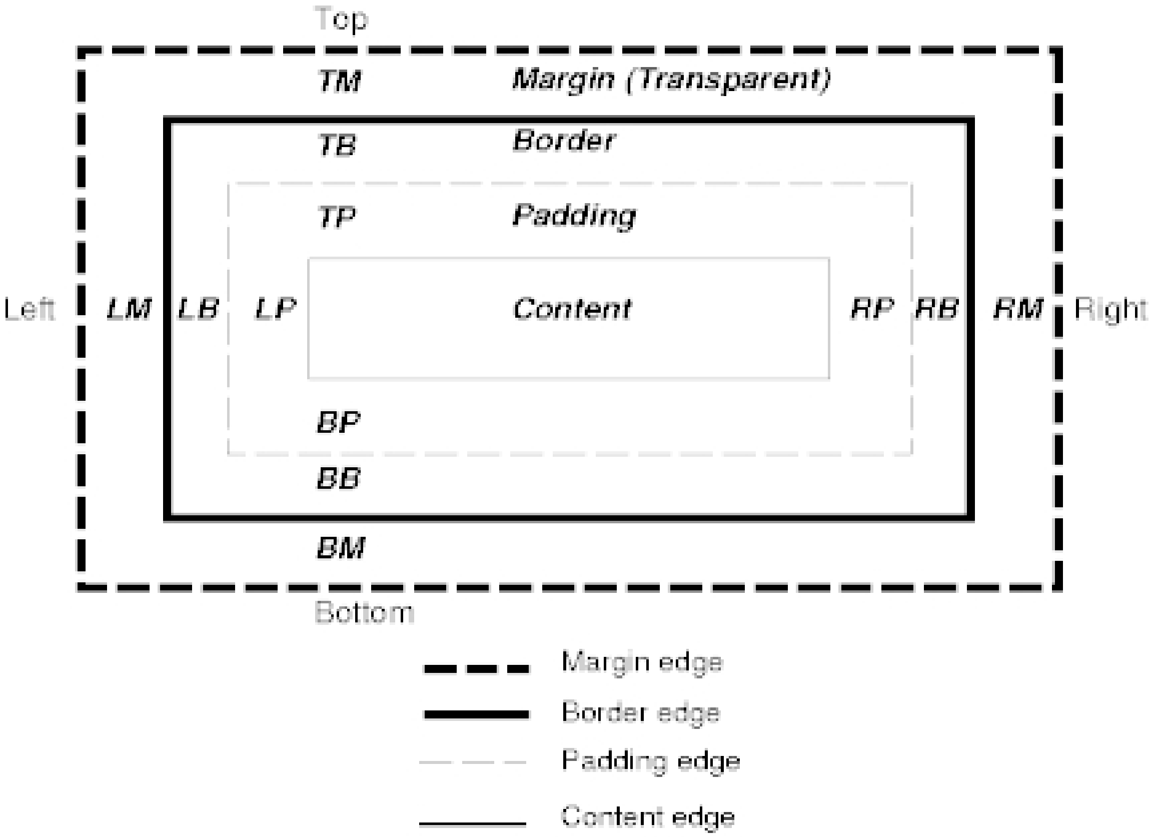


Figure 8.2 Box Model

The content, *content*, enclosed by span can contain only other inline tags. Typically, span encloses part of the content that is enclosed by another tag, to alter the appearance of a few words.

The span tag is used exclusively with styles using the class or id attribute. The syntax using attribute, class, is

<span class=*classname* >*content*</span>

Also, inline styles can be used with span using the attribute, style.

The styling options that can be used with span are limited. Two are color, font properties.

Examples using document-level styles

```
<head>
...
<style>
div.center{
   text-align:center
}
div.right{
   text-align:right
   }
   span.without{
     color:red
   }
   span {
     color:blue
   }
   </style>
   ...
   </head>
   <body>
   ...
   <div class="center">This block is <span>centered<br />
               using</span> styles.</div>
   <div class="right">This block is <span>right justified<br />
               using</span> styles.</div>
   <div align="center">This block is <span
class="without">centered<br />
               without using</span> styles.</div>
   <div align="right">This block is <span class="without">
               right justified<br />without using</span>
styles.</div>
   ...
   </body>
```

8.8 Web Page Demonstrations

There are eight different web pages, one for each category of properties in Figure 8.1. For convenience a web page, called a menu, has links to all of the web page demonstrations and each web page has a link back to the menu web page. The links in the web pages assume that all web pages are stored in the same directory. The display of these web pages by the browser is shown in figures in this chapter, which give the name of the web page (file name) in the caption.

There are two links on the menu for each of the eight demonstration web pages. One link, html, displays the web page on the full screen. The other link, php, is an interactive web page which displays the code of the demonstration web page on the left side and the output of the code on the right side. The code on the left side can be changed and then Submit clicked to see the output of the new code on the right. The original code and its output can be displayed by clicking, original page.

These web pages can be stored on a server and then can be viewed by typing the URL of the web page in a browser. Alternatively, the web pages can be stored on a client (computer with a browser) and viewed by clicking, File/Open, on the menu bar of the browser and then selecting an HTML file. On the client make a directory, C:\websitecss, for example if Windows is used, and then store all web pages in the directory.

8.8.1 Menu

The web page, menucss.htm, is a menu which gives the links to demonstration programs in this section.

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

In the web home directory on the server, create the directory, c08css, and then set the permissions of the directory to 707 (write permission is required because the interactive web page writes files on the server). After, change the working directory to c08css. This is done by the following series of Linux commands.

```
        mkdir c08css                    (create directory, c08css)
        chmod 707 c08css                (change permissions of directory, c08css)
        cd c08css                       (change working directory to c08css)
```

Web Page 8.1 menucss.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

Start a text editor in Linux, for example, emacs. Type the following file and save it using the filename, menucss.htm, in directory, c08css. In Linux it is necessary to set the permissions of all web pages to 604. Set the permissions for the menu web page after typing and saving the file. View the permissions before and after changing the permissions by the command, ls -l.

```
        emacs menucss.htm               (type and save the file, then exit emacs)
        ls -l m*                        (display permissions of all files starting with letter,
                                        m)
        chmod 604 menucss.htm           (change permissions of file, menucss.htm)
        ls -l m*                        (display permissions of all files starting with letter,
                                        m)
```

(Type the following web page)

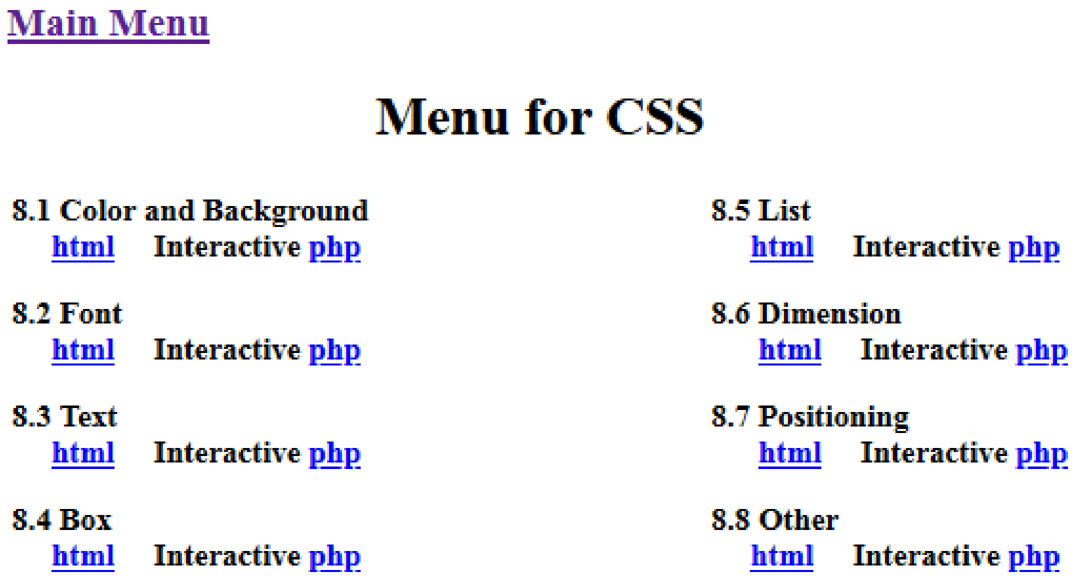
```
<html>
<head>
<title>Menu for CSS</title>
</head>
<body>
<h2><a href="../index.html">Main Menu</a></h2>

<table>
<tr><td></td><td><h1>Menu for CSS</h1></td><td></td></tr>
<tr><td>
      <h3>
8.1 Color and Background <br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="colorbackgr.htm"
target=_top>html</a>
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=colorbackgr.htm">php</a>
</h3><h3>
8.2 Font<br /> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="font.htm">html</a>
```

```
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=font.htm">php<a>
</h3><h3>
8.3 Text <br /> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="text.htm">html</a>
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=text.htm">php</a>
</h3><h3>
8.4 Box <br /> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="box.htm">html</a>
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=box.htm">php</a>
</h3>
    </td>

<td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&
nbsp;&nbsp;</td>
    <td><h3>
      8.5 List <br /> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="list.htm">html</a>
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=list.htm">php</a>
</h3><h3>
8.6 Dimension<br /> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <a
href="dimension.htm">html</a>
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=dimension.htm">php</a>
</h3><h3>
8.7 Positioning<br /> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <a
href="positioning.htm">html</a>
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=positioning.htm">php</a>
</h3><h3>
8.8 Other <br /> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="other.htm">html</a>
&nbsp;&nbsp;&nbsp;&nbsp;Interactive <a href="interactive.php?
fn=other.htm">php</a>
</h3>
</td>  </tr></table>
</body></html>
```

(End of the web page)

Start a browser, for example, Internet Explorer or Firefox in Windows or Firefox in Linux. Type the address (address is used in this context to mean URL) of the file, menucss.htm, in the address bar of the browser and then press Enter. The address has the form, http://*servername*/*directoryname(s)*/c08css/menucss.htm. Note that the title appears in the title bar at the top of the screen.

Figure 8.3 shows the menu as displayed by the browser. The web page, menucss.htm, also includes Figure 8.1, CSS Properties, but this is not shown in Figure 8.3. There are two web pages for each demonstration. For example, the demonstration of fonts, has the two web pages with links, html and php. The web page with link, html, shows only the output of the HTML source code. The web page with link, php, is an interactive web page which shows the source code on the left side and the output of the source code on the right side. This web page can be used to associate the lines of source code with the output of the corresponding source code. But more importantly, the source code on the left can be changed, Submit clicked and then the output of the changed source code appears on the right. The original source code and its output can be seen by clicking the link, original page.



Main Menu

Menu for CSS

8.1 Color and Background
html Interactive php

8.2 Font
html Interactive php

8.3 Text
html Interactive php

8.4 Box
html Interactive php

8.5 List
html Interactive php

8.6 Dimension
html Interactive php

8.7 Positioning
html Interactive php

8.8 Other
html Interactive php

Figure 8.3 Menu for web pages: menucss.htm

8.8.2 Color and Background

The color and background properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.2 colorbackgr.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

Start a text editor and type the following file and save it in directory, c08css, using the filename, colobackgr.htm. Then change the permissions of the file to 604.

(Type the following web page)

```
<html>
<head>
<title>Color and background properties</title>
<style type="text/css">
```

```
/************** 1. *************
********** background **********
********************************
values:
default value:
*******************************/
/***************** 2. *********************
********** background-attachment **********
*******************************************
values: scroll | fixed
default value: scroll
******************************************/

body {
background-attachment: fixed;
/*************** 3. ******************
********** background-color **********
**************************************
values: rgb(<r>,<g>,<b>) | #<hex> | <color-name> | transparent
default value: transparent
*************************************/
background-color: #ffffcc;
/************** 4. *******************
********** background-image **********
**************************************
values: url(<url-value>) | none
default value: none
*************************************/
background-image: url('logo.jpg');
/***************** 5. *******************
********** background-position **********
*****************************************
values: top left | top center | top right | center left | center
center | center right |
bottom left | bottom center | bottom right | x-<percent>
y-<percent> | x-<pos> y-<pos>
default value: top left
****************************************/
background-position: center top;
/***************** 6. *****************
********** background-repeat **********
***************************************
values: repeat | repeat-x | repeat-y | no-repeat
default value: repeat
**************************************/
background-repeat: repeat-x
}
```

```
/*********** 7. ***********
********** color **********
***************************
values: rgb(<r>,<g>,<b>) | #<hex> | <color-name>
default value: usually black, depends on browser
**************************/
h1 {
color: yellow;
background-color: #00ff00
}
h2 {
background-color: transparent
}
p {
background-color: rgb(255,0,255)
}
span.whitebg {
background-color:white
}
span.redbg {
background-color:red;
color:white
}
.greenbg {
background-color:green;
color:white
}
.bluebg {
background-color:blue;
color:white
}
</style>
</head>
<body>
<pre><span class=whitebg>
*****Color and Background Properties*****
1. background            - all background properties which follow
can be set at the same time
2. background-attachment- background image does or does not
scroll with page
3. background-color      - background color of an element (tag)
4. background-image      - background image is set
5. background-position   - background image starting position
6. background-repeat     - background image repeated or not
7. color                 - foreground color of text
Not demonstrated: 1
</span></pre>
```

```
<h1>h1: color, background-color</h1>
<h2>h2: background-color</h2>
<p>p: background-color</p>
<p>p: background-color</p>
<span class=whitebg>span: background-color</span><span
class=redbg>span: background-color, color</span>
<span class=greenbg>span: background-color, color</span><span
class=bluebg>span: background-color, color</span>
<p class=greenbg>p: background-color, color</p>
<p class=bluebg>p: background-color, color</p>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
</body>
</html>
```

(End of the web page)

Click the link, html, for Color and Background in the menu.

Figure 8.4 shows the display of the the web page, colorbackgr.htm.

8.8.3 Font

The font properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.3 font.htm

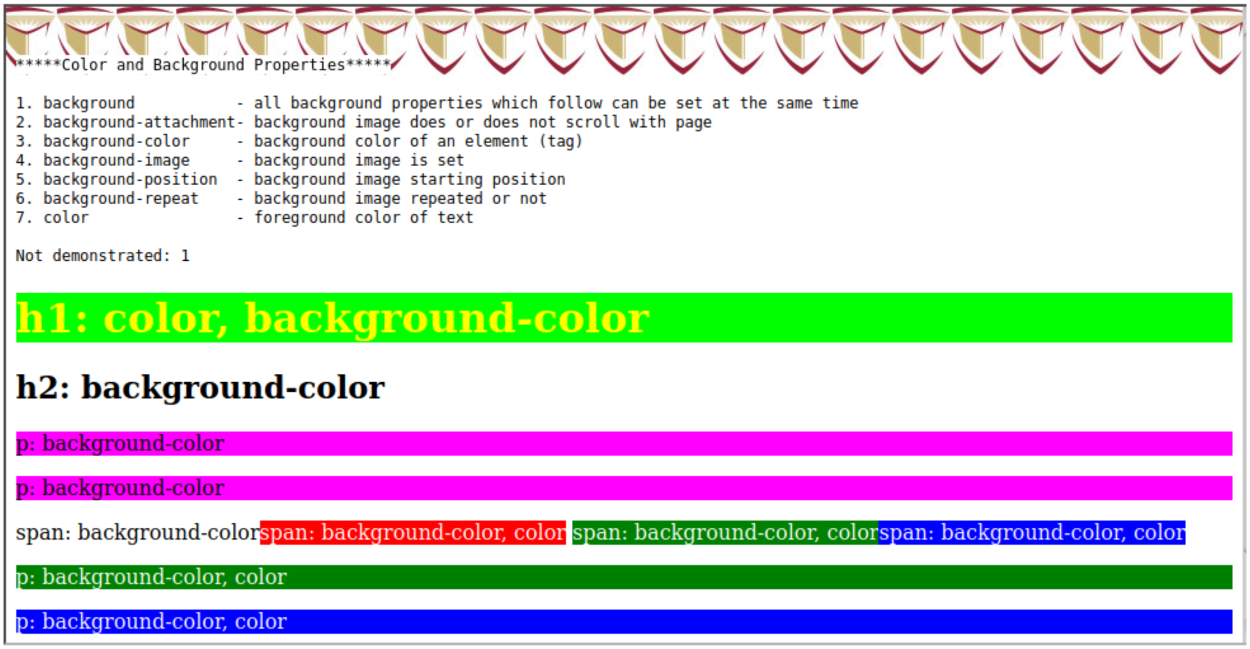
It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

Type the following file in an editor and save it in directory, c08css, using the filename, font.htm. Then change the permissions of the file to 604.

(Type the following web page)

```
<html>
<head>
<title>Font properties</title>
<style type="text/css">
/********** background-color **********/
body  {
  background-color: #ffffcc;    /* light grey */
}
span.whitebg {
  background-color:white
}
div.whitebg {
  background-color:white
}
/********** 1. ***********
********** font **********
*************************
```

```
values: <font-style> | <font-variant> | <font-weight> |
<font-size>/<line-height> |
<font-family> ] | caption | icon | menu | message-box |
small-caption | status-bar
default value: depends on browser
*************************/
```



```
*****Color and Background Properties*****

1. background           - all background properties which follow can be set at the same time
2. background-attachment- background image does or does not scroll with page
3. background-color     - background color of an element (tag)
4. background-image     - background image is set
5. background-position  - background image starting position
6. background-repeat    - background image repeated or not
7. color                - foreground color of text

Not demonstrated: 1
```

h1: color, background-color

h2: background-color

p: background-color

p: background-color

span: background-colorspan: background-color, color span: background-color, colorspan: background-color, color

p: background-color, color

p: background-color, color

Figure 8.4 Color and Background web page: colorbackgr.htm

```
/************* 2. ***************
********** font-family **********
*********************************
values: <family-name> | <generic-family>
default value: depends on browser
********************************/
h1 {
   font-family: times
}
.fam-courier {
   font-family: courier
}
.fam-sansserif {font-family: sans-serif
}
/************* 3. *************
********** font-size **********
*******************************
values: xx-small | x-small | small | medium | large |
x-large | xx-large | smaller | larger | <length> | <x>%
default value: medium
```

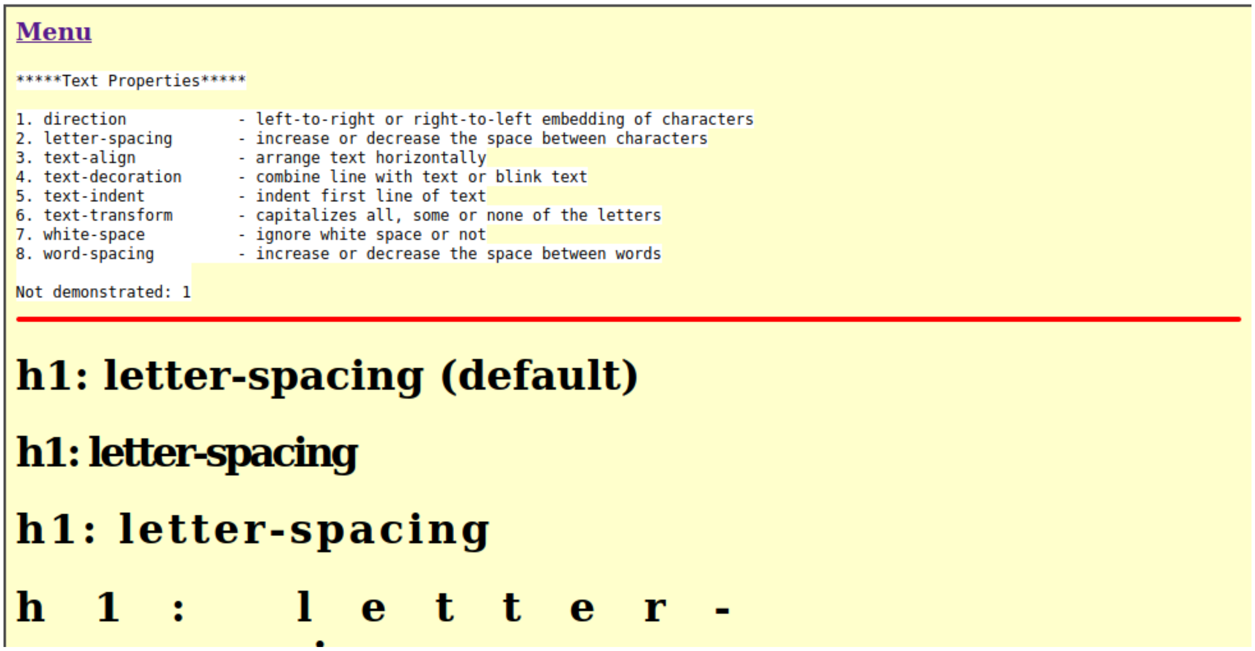
```
******************************/
.size-xxsmall {font-size: xx-small
}
.size-xxlarge {font-size: xx-large
}
.size-medium {font-size: medium
}\
/***************** 4. ****************
********** font-size-adjust **********
**************************************
values: none | <number>
default value: none
*************************************/
/*************** 5. **************
********** font-stretch **********
**********************************
values: normal | wider | narrower | ultra-condensed |
extra-condensed | condensed |
  semi-condensed | semi-expanded | expanded | extra-expanded |
ultra-expanded
default value: normal
*********************************/
/************* 6. **************
********** font-style **********
********************************
values: normal | italic | oblique
default value: normal
*******************************/
.style-normal {font-style: normal
}
.style-italic {font-style: italic
}
.style-oblique {font-style: oblique
}
/************** 7. ***************
********** font-variant **********
**********************************
values: normal | small-caps
default value: normal
*********************************/
.variant-normal {font-variant: normal
}
.variant-smallcaps {font-variant: small-caps
}
/************** 8. **************
********** font-weight **********
*********************************
```

```
values: normal | bold | bolder | lighter | 100 | 200 | 300 | 400
| 500 | 600 | 700 | 800 | 900
default value: normal
********************************/
.weight-normal {font-weight: normal     /* normal is the same as
400*/
}
.weight-bold {font-weight: bold         /* bold is the same as
700*/
}
</style>
</head>
<body>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
<pre><div class=whitebg>*****Font Properties*****
1. font         - all font properties can be set at the same time
2. font-family  - font face is set
3. font-size    - font size is set
4. font-size-adjust - aspect value (height of lowercase
'x'/height of font-size} of first choice of font is set
5. font-stretch - font family is contracted or expanded
6. font-style   - font style is set
7. font-variant - text is displayed in small caps or not
8. font-weight  - font weight is set
Not demonstrated: 1, 4, 5
</div></pre>
<!-- DEMONSTRATIONS START HERE -->
<hr color=CornflowerBlue size=5px />
<h1>h1: font-family</h1>
<h1 class=fam-courier>h1: font-family</h1>
<h1 class=fam-sansserif>h1: font-family</h1>
<hr color=CornflowerBlue size=5px />
<h1 class=size-xxsmall>h1: font-size</h1>
<h1 class=size-medium>h1: font-size</h1>
<h1 class=size-xxlarge>h1: font-size</h1>
<hr color=CornflowerBlue size=5px />
<h1 class=style-normal>h1: font-style</h1>
<h1 class=style-italic>h1: font-style</h1>
<h1 class=style-oblique>h1: font-style</h1>
<hr color=CornflowerBlue size=5px />
<h1 class=variant-normal>h1: FONT-variant</h1>
<h1 class=variant-smallcaps>h1: FONT-variant</h1>
<hr color=CornflowerBlue size=5px />
<h1 class=weight-normal>h1: font-weight</h1>
<h1 class=weight-bold>h1: font-weight</h1>
<p class=weight-normal>p: font-weight</p>
<p class=weight-bold>p: font-weight</p>
```

```
<hr color=CornflowerBlue size=5px />
</body>
</html>
```

(End of the web page)
Click the link, html, for Font in the menu.

Figure 8.5 shows the display of the top part of the web page, font.htm.



Menu

```
*****Text Properties*****

1. direction          - left-to-right or right-to-left embedding of characters
2. letter-spacing     - increase or decrease the space between characters
3. text-align         - arrange text horizontally
4. text-decoration    - combine line with text or blink text
5. text-indent        - indent first line of text
6. text-transform     - capitalizes all, some or none of the letters
7. white-space        - ignore white space or not
8. word-spacing       - increase or decrease the space between words

Not demonstrated: 1
```

h1: letter-spacing (default)

h1: letter-spacing

h1: letter-spacing

h1: letter-

Figure 8.5 Font web page, top part: font.htm

8.8.4 Text

The text properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.4 text.htm
It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.
Type the following file in an editor and save it in directory, c08css, using the filename, text.htm. Then change the permissions of the file to 604.

(Type the following web page)

```
<html>
<head>
<title>Text properties</title>
<style type="text/css">
/********** background-color **********/
body  {
   background-color: #ffffcc;    /* light grey */
}
```

```
span.whitebg {
  background-color:white
}
div.whitebg {
  background-color:white
}
/************* 1. *************
********** direction **********
*******************************
values: ltr | rtl
default value: ltr
******************************/

/************** 2. *****************
********** letter-spacing **********
************************************
values: normal | <length>
default value: normal
relative length units: em, ex, px
absolute length units: in, cm, mm, pt, pc
***********************************/
.text-letterspacing1 { letter-spacing: -3px
}
.text-letterspacing2 { letter-spacing: 3px
}
.text-letterspacing3 { letter-spacing: 1cm
}
/************* 3. ***************
********** text-align **********
********************************
values: left | right | center | justify
default value:  left if 'direction' is ltr;
     right if 'direction' is rtl
*******************************/
.text-alignleft {text-align: left;
     color: red
}
.text-aligncenter {text-align: center;
     color: white
}
.text-alignright {text-align: right;
     color: blue
}
/*************** 4. *****************
********** text-decoration **********
*************************************
values: none | underline | line-through | underline | blink
```

```
default value: none
************************************/
.text-decorationover {text-decoration: overline; /
  color: yellow
}
.text-decorationthrough {text-decoration: line-through;
  color: olive
}
.text-decorationunder {text-decoration: underline;
  color: purple
}
.text-decorationblink {text-decoration: blink
}
/************** 5. **************
********** text-indent **********
*********************************
values: <length> | <percentage>
default value: 0
********************************/
.text-indent1 {text-indent: 1cm
}
/*************** 6. ****************
********** text-transform **********
************************************
values: capitalize | uppercase | lowercase | none
default value: none
***********************************/
.text-transformcap {text-transform: capitalize
}
.text-transformupper {text-transform: uppercase
}
.text-transformlower {text-transform: lowercase
}
/************** 7. **************
********** white-space **********
*********************************
values: normal | pre | nowrap | pre-wrap | pre-line
default value: normal
********************************/
.text-whitespacepre {white-space: pre
}
/************* 8. ****************
********** word-spacing **********
**********************************
values: normal | <length>
default value: normal */
/* normal is the same as 0 length
```

```
*********************************/
.text-wordspacing0 {word-spacing: normal
}
.text-wordspacing1 {word-spacing: 1cm;
     color:red
}
.text-wordspacing2 {word-spacing: 4px;
     color:white
}
.text-wordspacing3 {word-spacing: -4px;
     color:blue
}
</style>
</head>
<body>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
<pre><span class=whitebg>*****Text Properties*****
1. direction            - left-to-right or right-to-left
embedding of characters
2. letter-spacing       - increase or decrease the space between
characters
3. text-align           - arrange text horizontally
4. text-decoration      - combine line with text or blink text
5. text-indent          - indent first line of text
6. text-transform       - capitalizes all, some or none of the
letters
7. white-space          - ignore white space or not
8. word-spacing         - increase or decrease the space between
words
Not demonstrated: 1
</span></pre>
<!-- DEMONSTRATIONS START HERE -->
<hr color=CornflowerBlue size=5px />
<h1>h1: letter-spacing (default)</h1>
<h1 class=text-letterspacing1>h1: letter-spacing</h1>
<h1 class=text-letterspacing2>h1: letter-spacing</h1>
<h1 class=text-letterspacing3>h1: letter-spacing</h1>
<hr color=CornflowerBlue size=5px />
<h1 class=text-alignleft>h1: text-align</h1>
<h1 class=text-aligncenter>h1: text-align</h1>
<h1 class=text-alignright>h1: text-align</h1>
<span class=text-alignleft>span: text-align</span>      <!-- to
get the expected results on one line, -->
<span class=text-aligncenter>span: text-align</span>    <!-- a
table must be used with one row and three columns, -->
<span class=text-alignright>span: text-align</span>     <!-- and
div is used instead of span -->
```

```
<table width=100%><tr><td><div class=text-alignleft>div:
text-align</div></td>
<td><div class=text-aligncenter>div: text-align</div></td>
<td><div class=text-alignright>div:
text-align</div></td></tr></table>
<hr color=CornflowerBlue size=5px />
<div class=text-decorationover>div: text-decoration</div>
<div class=text-decorationthrough>div: text-decoration</div>
<div class=text-decorationunder>div: text-decoration</div>
<div class=text-decorationblink>div: text-decoration</div>
<hr color=CornflowerBlue size=5px />
<p class=text-indent1>p: text-indent<br />line 1<br />line 2</p>
<p class=text-indent2>p: text-indent<br />line 1<br />line 2</p>
<p class=text-indent3>p: text-indent<br />line 1<br />line 2</p>
<hr color=CornflowerBlue size=5px />
<h1 class=text-transformcap>h1: text-transform ***This is a
heading***</h1>
<h1 class=text-transformupper>h1: text-transform ***This is a
heading***</h1>
<h1 class=text-transformlower>h1: text-transform ***This is a
heading***</h1>
<hr color=CornflowerBlue size=5px />
<p class=text-whitespacepre>p: white-space
The property, pre,
   has the same effect
      as the tag, pre,
         in HTML.</p>
<hr color=CornflowerBlue size=5px />
<p class=text-wordspacing0>p: word-spacing This is four words
(normal spacing)</p>
<p class=text-wordspacing1>p: word-spacing This is four words</p>
<p class=text-wordspacing2>p: word-spacing This is four words</p>
<p class=text-wordspacing3>p: word-spacing This is four words</p>
<div class=text-wordspacing0>div: word-spacing This is four words
(normal spacing)</div>
<div class=text-wordspacing1>div: word-spacing This is four
words</div>
<div class=text-wordspacing2>div: word-spacing This is four
words</div>
<div class=text-wordspacing3>div: word-spacing This is four
words</div>
<span class=text-wordspacing0>span: word-spacing This is four
words (normal spacing)</span>
<span class=text-wordspacing1>span: word-spacing This is four
words</span>
<span class=text-wordspacing2>span: word-spacing This is four
words</span>
```

```
<span class=text-wordspacing3>span: word-spacing This is four
words</span>
<hr color=CornflowerBlue size=5px />
</body></html>
```

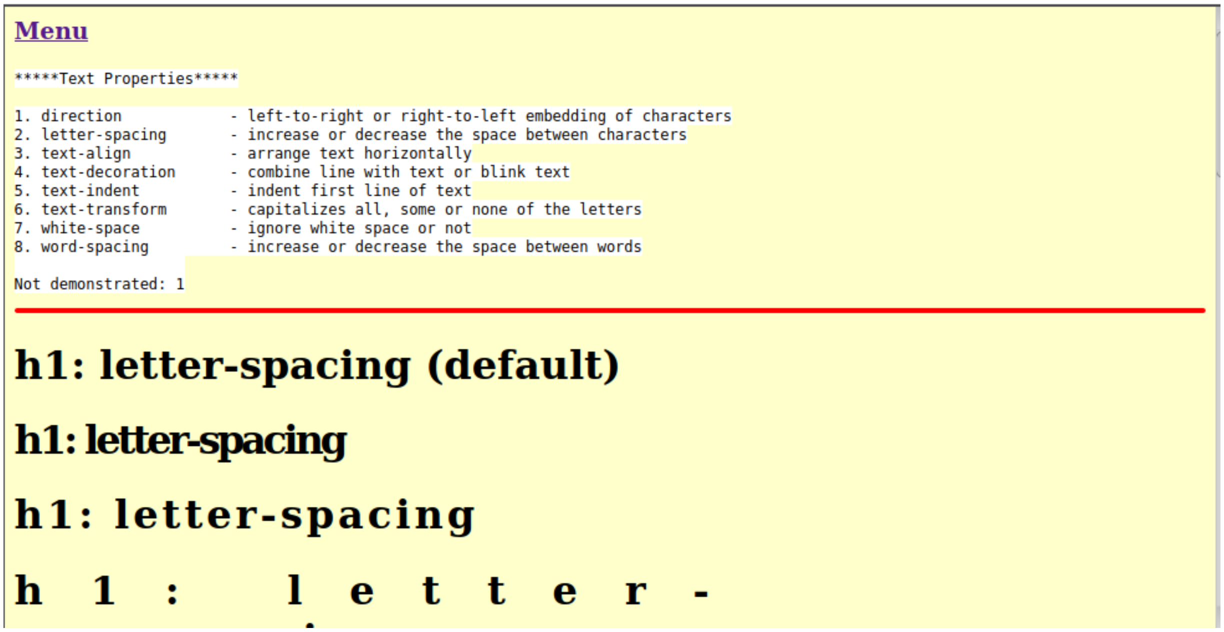
(End of the web page)

Click the link, html, for Text in the menu.
Figure 8.6 shows the display of the top part of the web page, text.htm.

8.8.5 Box

The box properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.5 box.htm
It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.



Menu

```
*****Text Properties*****

1. direction          - left-to-right or right-to-left embedding of characters
2. letter-spacing     - increase or decrease the space between characters
3. text-align         - arrange text horizontally
4. text-decoration    - combine line with text or blink text
5. text-indent        - indent first line of text
6. text-transform     - capitalizes all, some or none of the letters
7. white-space        - ignore white space or not
8. word-spacing       - increase or decrease the space between words

Not demonstrated: 1
```

h1: letter-spacing (default)

h1: letter-spacing

h1: letter-spacing

h1: letter-

Figure 8.6 Text web page, top part: text.htm

(Type the following web page)
Type the following file in an editor and save it in directory, c08css, using the filename, box.htm. Then change the permissions of the file to 604.

```
<html>
<head>
<title>Box properties</title>
<style type="text/css">
/********** background-color **********/
body  { background-color: #ffffcc;      /* light grey */ }
span.whitebg { background-color:white }
```

```
div.whitebg { background-color:white }
/*********** 1. ************
********** border **********
****************************
values: <border-width> | <border-style> | <border-color>
default value: same as individual properties
***************************/
.box-borderall {border: thick dashed blue
}
/*********** 2.
**********************************************************
********** border-top, border-right, border-bottom, border-left
**********
*****************************************************************
*********
values: <border-width> | <border-style> | <border-color>
default value: same as individual properties
*****************************************************************
********/
/************* 3. ****************
********** border-color **********
**********************************
values: <color> | transparent
default value: value of the color property
*********************************/
.box-bordercolorred {border-color: red;
     border-style: dotted }
.box-bordercolorwhite {border-color: white;
     border-style: dashed }
.box-bordercolorblue {border-color: blue;
     border-style: solid        }
.box-bordercolorgreen {border-color: green;
     border-style: double }
.box-bordercolorredwhitebluegreen {border-color: red white blue
green; border-style: double }
/************* 4.
*****************************************************************
****************
********** border-top-color,  border-right-color,
border-bottom-color, border-left-color **********
*****************************************************************
**********************************
values: <color> | transparent
default value: value of the color property
*****************************************************************
*********************************/
/************** 5. ***************
```

```
********** border-style **********
**********************************
values: none | hidden | dotted | dashed | solid | double | groove
| ridge | inset | outset
default value: none
*********************************/
.box-borderstyledot { border-style: dotted }
.box-borderstyledash { border-style: dashed }
.box-borderstylesolid { border-style: solid }
.box-borderstyledouble { border-style: double }
.box-borderstyledotdashsoliddouble { border-style: dotted dashed
solid double    }
/************** 6. ***************
********** border-top-style, border-right-style,
border-bottom-style, border-left-style  **********
**********************************
values: none | hidden | dotted | dashed | solid | double | groove
| ridge | inset | outset
default value: none
*********************************/
/************** 7. ***************
********** border-width **********
**********************************
values: thin | medium | thick | <length>
default value:  medium
*********************************/
.box-borderwidththin {border-width: thin;
     border-style: dotted }
.box-borderwidthmedium {border-width: medium;
     border-style: dashed }
.box-borderwidththick {border-width: thick;
     border-style: solid }
.box-borderwidththinmediumthickmedium {border-width: thin medium
thick medium;   border-style: solid }
.box-borderwidthpixels {border-width: 5px 10px 15px 20px;
     border-style: solid }
/************** 8.
*****************************************************************
**************
********** border-top-width, border-right-width,
border-bottom-width, border-left-width **********
*****************************************************************
*********************************
values: thin | medium | thick | <length>
default value:  medium
*****************************************************************
********************************/
```

```
/*********** 9. ************
********** margin **********
****************************
values: <margin-top> | <margin-right> | <margin-bottom> |
<margin-left>
default value:  0
***************************/
.box-marginallnone {border-style: solid }
.box-marginall {margin: 1cm 2cm 3cm 4cm;
     border-style: solid }
/************* 10.
*******************************************************
********** margin-top, margin-right, margin-bottom, margin-left
**********
*********************************
values: auto | <length> | <percentage>
default value:  0
*****************************************************************
********/
.box-marginleft {margin-left: 2cm;
     border-style: solid }
/*********** 11. ***********
********** padding **********
****************************
values: <length> | <percentage>
default value:  0
***************************/
.box-paddingall1 {border-style: solid;
     background-color: #ffffc6 }
.box-paddingall2 {padding: 2cm;
     border-style: solid;
     background-color: #ffffc6 }
/************* 12. **************
********** padding-top, padding-right, padding-bottom,
padding-left **********
*********************************
values: <length> | <percentage>
default value:  0
********************************/
.box-paddingleft1 {padding-left: 2cm;
     border-style: solid;
     background-color: #ffffc6 }
.box-paddingleft2 {padding-left: 2cm;
     border-style: solid;
     margin-left: 2cm;
     background-color: #ffffc6 }
.box-paddingleft3 {padding-left: 2cm;
```

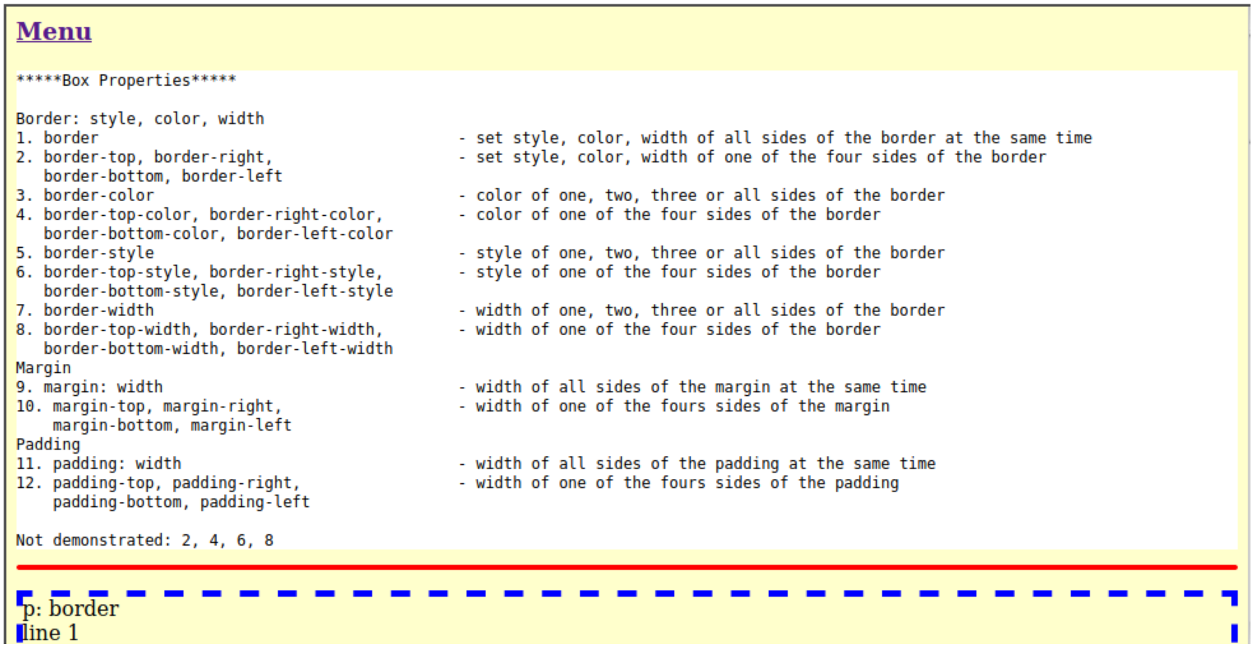
```
     border-style: solid;
     border-left-width: 1cm;
     margin-left: 2cm;
     background-color: #ffffc6 }
</style>
</head>
<body>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
<pre><div class=whitebg>*****Box Properties*****
Border: style, color, width
1. border                                       - set style, 
color, width of all sides of the border at the same time
2. border-top, border-right,                    - set style, 
color, width of one of the four sides of the border
   border-bottom, border-left
3. border-color                                 - color of one, 
two, three or all sides of the border
4. border-top-color, border-right-color,        - color of one of 
the four sides of the border
   border-bottom-color, border-left-color
5. border-style                                 - style of one, 
two, three or all sides of the border
6. border-top-style, border-right-style,        - style of one of 
the four sides of the border
   border-bottom-style, border-left-style
7. border-width                         - width of one, two, 
three or all sides of the border
8. border-top-width, border-right-width,        - width of one of 
the four sides of the border
   border-bottom-width, border-left-width
Margin
9. margin: width                                - width of all 
sides of the margin at the same time
10. margin-top, margin-right,                   - width of one of 
the fours sides of the margin
    margin-bottom, margin-left
Padding
11. padding: width                              - width of all 
sides of the padding at the same time
12. padding-top, padding-right,         - width of one of the 
fours sides of the padding
    padding-bottom, padding-left
Not demonstrated: 2, 4, 6, 8
</div></pre>
<!-- DEMONSTRATIONS START HERE -->
<hr color=CornflowerBlue size=5px />
<p class=box-borderall>p: border<br />line 1<br />line 2</p>
```

```
<hr color=CornflowerBlue size=5px />
<p class=box-borderstyledot>p: border-style</p>
<p class=box-borderstyledash>p: border-style</p>
<p class=box-borderstylesolid>p: border-style</p>
<p class=box-borderstyledouble>p: border-style</p>
<p class=box-borderstyledotdashsoliddouble>p: border-style</p>
<hr color=CornflowerBlue size=5px />
<p class=box-bordercolorred>p: border-color</p>
<p class=box-bordercolorwhite>p: border-color</p>
<p class=box-bordercolorblue>p: border-color</p>
<p class=box-bordercolorgreen>p: border-color</p>
<p class=box-bordercolorredwhitebluegreen>p: border-color</p>
<hr color=CornflowerBlue size=5px />
<p class=box-borderwidththin>p: border-width</p>
<p class=box-borderwidthmedium>p: border-width</p>
<p class=box-borderwidththick>p: border-width</p>
<p class=box-borderwidththinmediumthickmedium>p: border-width</p>
<p class=box-borderwidthpixels>p: border-width</p>
<hr color=CornflowerBlue size=5px />
<p class=box-marginallnone>p: margin (no margin) one two three
four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=box-marginall>p: margin (all sides) one two three four
five six seven eight nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=box-marginallnone>p: margin (no margin) one two three
four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
<p class=box-marginallnone>p: margin-left (no margin) one two
three four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=box-marginleft>p: margin-left (left margin) one two
three four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
```

```
<p class=box-paddingall1>p: padding (no padding) one two three
four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=box-paddingall2>p: padding (all sides same) one two
three four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
<p class=box-paddingleft1>p: padding (left padding) one two three
four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=box-paddingleft2>p: padding (left padding and left
margin) one two three four five six seven eight nine ten eleven
twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=box-paddingleft3>p: padding (left padding, left border
and left margin) one two three four five six seven eight nine ten
eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
</body>
</html>
```

(End of the web page)
Click the link, html, for Box in the menu.

Figure 8.7 shows the display of the top part of the web page, box.htm.



Menu

```
*****Box Properties*****

Border: style, color, width
1. border                                 - set style, color, width of all sides of the border at the same time
2. border-top, border-right,              - set style, color, width of one of the four sides of the border
   border-bottom, border-left
3. border-color                           - color of one, two, three or all sides of the border
4. border-top-color, border-right-color,  - color of one of the four sides of the border
   border-bottom-color, border-left-color
5. border-style                           - style of one, two, three or all sides of the border
6. border-top-style, border-right-style,  - style of one of the four sides of the border
   border-bottom-style, border-left-style
7. border-width                           - width of one, two, three or all sides of the border
8. border-top-width, border-right-width,  - width of one of the four sides of the border
   border-bottom-width, border-left-width
Margin
9. margin: width                          - width of all sides of the margin at the same time
10. margin-top, margin-right,             - width of one of the fours sides of the margin
    margin-bottom, margin-left
Padding
11. padding: width                        - width of all sides of the padding at the same time
12. padding-top, padding-right,           - width of one of the fours sides of the padding
    padding-bottom, padding-left

Not demonstrated: 2, 4, 6, 8
```

p: border
line 1

Figure 8.7 Box web page, top part: box.htm

8.8.6 List

The list properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.6 list.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.
Type the following file in an editor and save it in directory, c08css, using the filename, list.htm.
Then change the permissions of the file to 604.

(Type the following web page)

```
<html>
<head>
<title>List properties</title>

<style type="text/css">
/********** background-color **********/
body  {
  background-color: #ffffcc;    /* light grey */
}
span.whitebg {
  background-color:white
}
div.whitebg {
  background-color:white
}
/************* 1. **************
********** list-style **********
```

```
********************************
values: <list-style-type> | <list-style-position> |
<list-style-image>
default value: same as individual properties
*******************************/
.list-borderall {border: thick dashed blue
}
/**************** 2. *****************
********** list-style-image **********
**************************************
values: none | <url>
default value: none
*************************************/
.list-styleimage { list-style-image: url('arrow.jpg')
}
/***************** 3. *******************
********** list-style-position **********
*****************************************
values: inside | outside
default value: outside
****************************************/
.list-stylepositionin {list-style-position: inside
}
.list-stylepositionout {list-style-position: outside
}
/**************** 4. ****************
********** list-style-type **********
*************************************
values: none | disc | circle | square | decimal |
decimal-leading-zero | lower-roman | upper-roman |
lower-greek | lower-latin | upper-latin | armenian | georgian |
lower-alpha | upper-alpha
default value:  disc
***********************************/
.list-styletypedisc {list-style-type: disc
}
.list-styletypecircle {list-style-type: circle
}
.list-styletypesquare {list-style-type: square
}
.list-styletypenone {list-style-type: none
}
.list-styletypedec {list-style-type: decimal
}
.list-styletypelowerlat {list-style-type: lower-latin
}
.list-styletypeupperlat {list-style-type: upper-latin
}
.list-styletypelowerrom {list-style-type: lower-roman
}
.list-styletypeupperrom {list-style-type: upper-roman
}
.list-styletypelowergreek {list-style-type: lower-greek
}
</style>
</head>
<body>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
<pre><div class=whitebg>*****List Properties*****
1. list-style           - all list properties set at the same
time
2. list-style-image     - image used as list-item marker
3. list-style-position  - position of list-item marker
4. list-style-type      - type of list-item marker
Not demonstrated: none
</div></pre>
<!-- DEMONSTRATIONS START HERE -->
<hr color=CornflowerBlue size=5px />
<table width=100%><tr>
<!-- First Column -->
<td valign=top>
ul: list-style-type
<ul class=list-styletypedisc>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypecircle>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypesquare>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypenone>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypedec>
<li>Item 1</li>
```

```
<li>Item 2</li>
<li>Item 3</li>
</ul>
</td>
<!-- Second Column -->
<td valign=top>
ol: list-style-type
<ul class=list-styletypedec>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypelowerlat>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypeupperlat>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypelowerrom>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypeupperrom>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-styletypelowergreek>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
</td>
<!-- Third Column -->
<td valign=top>
ul: list-style-position
<ul class=list-stylepositionin>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
<ul class=list-stylepositionout>
```

```
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
</td><!-- Fourth Column -->
<td valign=top>
ul: list-style-image
<ul class=list-styleimage>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>
</td></tr></table>
<hr color=CornflowerBlue size=5px />
</body>
</html>
```

(End of the web page)

Click the link, html, for List in the menu.

Figure 8.8 shows the display of the web page, list.htm.

8.8.7 Dimension

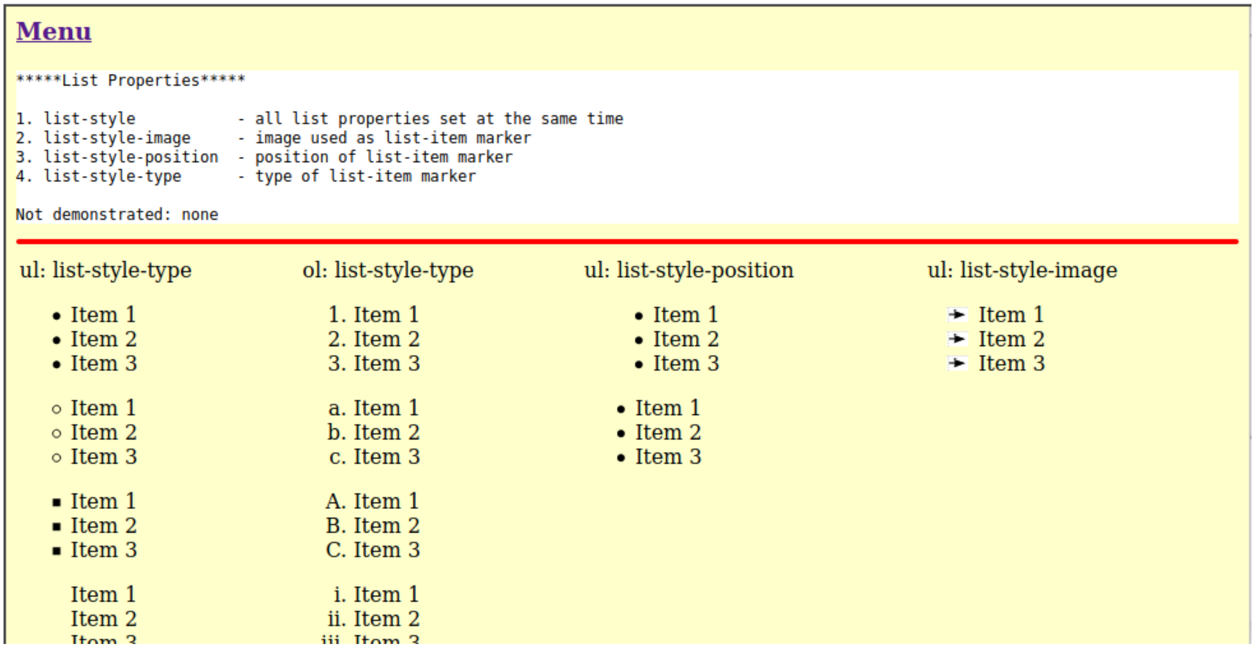
The dimension properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.7 dimension.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

Type the following file in an editor and save it in directory, c08css, using the filename, dimension.htm. Then change the permissions of the file to 604.

```
<html>
<head>
<title>Dimension properties</title>
<style type="text/css">
/********** background-color **********/
body  { background-color: #ffffcc;      /* light grey */ }
span.whitebg { background-color:white }
div.whitebg { background-color:white }
/*********** 1. ************
********** height **********
***************************
values: <length> | <percentage> | auto
default value: auto
***************************/
```



Menu

```
*****List Properties*****

1. list-style           - all list properties set at the same time
2. list-style-image     - image used as list-item marker
3. list-style-position  - position of list-item marker
4. list-style-type      - type of list-item marker

Not demonstrated: none
```

ul: list-style-type

- Item 1
- Item 2
- Item 3

○ Item 1
○ Item 2
○ Item 3

▪ Item 1
▪ Item 2
▪ Item 3

Item 1
Item 2
Item 3

ol: list-style-type

1. Item 1
2. Item 2
3. Item 3

a. Item 1
b. Item 2
c. Item 3

A. Item 1
B. Item 2
C. Item 3

i. Item 1
ii. Item 2
iii. Item 3

ul: list-style-position

- Item 1
- Item 2
- Item 3

- Item 1
- Item 2
- Item 3

ul: list-style-image

➔ Item 1
➔ Item 2
➔ Item 3

Figure 8.8 List web page: list.htm

```
.dim-height1 {height: 0.75cm; border-style: solid }
.dim-height2 {height: 2cm; border-style: solid }
/************** 2. **************
********** line-height **********
**********************************
values: normal | <number> | <length> | <percentage>
default value: normal
**********************************/
.dim-lineheightnormal {line-height: normal;
  border-style: solid }
.dim-lineheighthalf {line-height: 50%;
  border-style: solid }
.dim-lineheightdouble {line-height: 200%;
  border-style: solid }
/************* 3. **************
********** max-height **********
*********************************
values: <length> | <percentage> | none
default value: none
*********************************/
.dim-maxheight1 {max-height: 0.75cm;
  border-style: solid }
.dim-maxheight2 {max-height: 2cm;
  border-style: solid }
/************ 4. **************
********** max-width **********
********************************
```

```
values: <length> | <percentage> | none
default value: none
******************************/
.dim-maxwidth1 {max-width: 20cm;
  border-style: solid }
.dim-maxwidth2 {max-width: 10cm;
  border-style: solid }
/************* 5. **************
********** min-height **********
********************************
values: <length> | <percentage>
default value:  0
*******************************/
.dim-minheight1 {min-height: 0.75cm;
  border-style: solid }
.dim-minheight2 {min-height: 2cm;
  border-style: solid }
/************ 6. **************
********** min-width **********
*******************************
values: <length> | <percentage>
default value:  0
******************************/
.dim-minwidth1 {min-width: 20cm;
  border-style: solid }
.dim-minwidth2 {min-width: 10cm;
  border-style: solid }
/*********** 7. ***********
********** width **********
***************************
values: <length> | <percentage> | auto
default value: auto
**************************/
.dim-width1 {width: 20cm;
  border-style: solid }
.dim-width2 {width: 10cm;
  border-style: solid }
</style>
</head>
<body>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
<pre><div class=whitebg>*****Dimension Properties*****
1. height               - height of content of box
2. line-height          - height of each line in a box
3. max-height           - maximum height of content of box
4. max-width            - maximum width of content of box
5. min-height           - minimum height of content of box
```

```
6. min-width            - minimum width of content of box
7. width                - width of content of box

Not demonstrated: none
</div></pre>
<!-- DEMONSTRATIONS START HERE -->
<hr color=CornflowerBlue size=5px />
<p class=dim-lineheightnormal>p: line-height (normal) one two
three four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=dim-lineheighthalf>p: line-height (half) one two three
four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=dim-lineheightdouble>p: line-height (double) one two
three four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
<p class=dim-height1>p: height (0.75cm) one two three four five
six seven eight nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=dim-height2>p: height (2cm) one two three four five six
seven eight nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
<p class=dim-maxheight1>p: max-height (0.75cm) one two three four
five six seven eight nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=dim-maxheight2>p: max-height (2cm) one two three four
five six seven eight nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
<p class=dim-minheight1>p: min-height (0.75cm) one two three four
five six seven eight nine ten eleven twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=dim-minheight2>p: min-height (2cm) one two three four
five six seven eight nine ten eleven twelve thirteen fourteen
```

```
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<hr color=CornflowerBlue size=5px />
<p class=dim-width1>p: width (20cm) one two three four five</p>
<p class=dim-width2>p: width (10cm) one two three four five</p>
<hr color=CornflowerBlue size=5px />
<p class=dim-maxwidth1>p: max-width (20cm) one two three four
five</p>
<p class=dim-maxwidth2>p: max-width (10cm) one two three four
five</p>
<div class=dim-maxwidth1>div: max-width (20cm) one two three four
five</div>
<div class=dim-maxwidth2>div: max-width (10cm) one two three four
five</div>
<hr color=CornflowerBlue size=5px />
<p class=dim-minwidth1>p: min-width (20cm) one two three four
five</p>
<p class=dim-minwidth2>p: min-width (10cm) one two three four
five</p>
<div class=dim-minwidth1>div: min-width (20cm) one two three four
five</div>
<div class=dim-minwidth2>div: min-width (10cm) one two three four
five</div>
<hr color=CornflowerBlue size=5px />
</body>
</html>
```

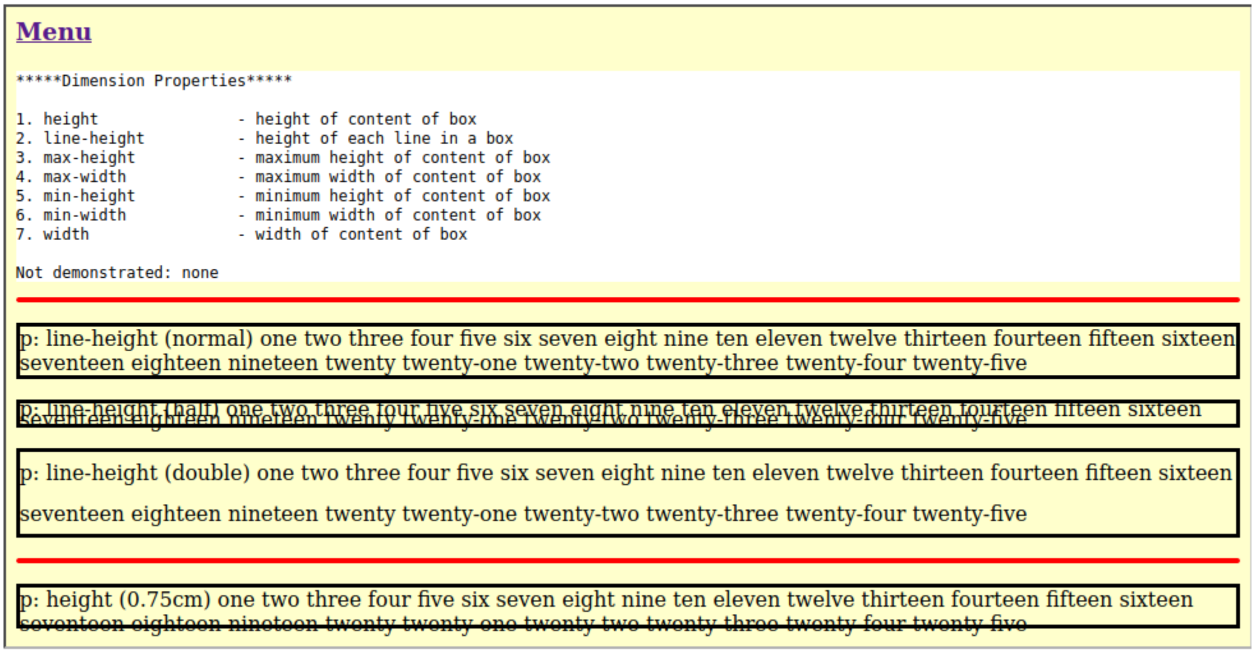
(End of the web page)

Click the link, html, for Dimension in the menu.
Figure 8.9 shows the display of the top of the web page, dimension.htm.

8.8.8 Positioning

The positioning properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.8 positioning.htm
It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.
Type the following file in an editor and save it in directory, c08css, using the filename, positioning.htm. Then change the permissions of the file to 604.



Menu

```
*****Dimension Properties*****

1. height              - height of content of box
2. line-height         - height of each line in a box
3. max-height          - maximum height of content of box
4. max-width           - maximum width of content of box
5. min-height          - minimum height of content of box
6. min-width           - minimum width of content of box
7. width               - width of content of box

Not demonstrated: none
```

p: line-height (normal) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

p: line-height (half) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

p: line-height (double) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

p: height (0.75cm) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

Figure 8.9 Dimension web page, top part: dimension.htm

(Type the following web page)

```
<html>
<head>
<title>Positioning properties</title>
<style type="text/css">
/********** background-color **********/
body  {
  background-color: #ffffcc;   /* light grey */
}
span.whitebg {
  background-color:white
}
div.whitebg {
  background-color:white
}
/********** 1. ***********
********** clip **********
**************************
values: <shape> | auto
<shape>: rect(<top>, <right>, <bottom>, <left>)
default value: auto
*************************/
/************ 2. *************
********** overflow **********
******************************
values: visible | hidden | scroll | auto
default value:  visible
```

```
*****************************/
/************ 3. *************
********** position **********
******************************
values: static | relative | absolute | fixed
default value: static
*****************************/
.pos-positionstatic{position: static;
  border-style: solid
}
/************ 4. *****************************
********** top, right, bottom, left **********
**********************************************
values: <length> | <percentage> | auto
default value: auto
*********************************************/
.pos-positionrelative1{position: relative;
  left: -1cm;
  border-style: solid
}
.pos-positionrelative2{position: relative;
  left: 1cm;
  border-style: solid
}
.pos-positionabsolute1 {position: absolute;
  left: 15cm;
  top: 5cm;
  border-style: solid
}
.pos-positionfixed1 {position: fixed;
  left: 15cm;
  top: 3cm;
  border-style: solid
}
/************** 5. *****************
********** vertical-align **********
************************************
values: baseline | sub | super | top | text-top | middle | bottom
| text-bottom | <percentage> | <length>
default value: baseline
***********************************/
.pos-verticalalignsub {vertical-align: sub
}
.pos-verticalalignsuper {vertical-align: super
}
.pos-verticalaligntop {vertical-align: top
}
```

```
.pos-verticalalignmid {vertical-align: middle
}
.pos-verticalalignbot {vertical-align: bottom
}
/************ 6. ************
********** z-index **********
****************************
values: auto | <integer>
default value: auto
****************************/
.pos-zindexfront {z-index: 1;
  position: absolute;
  left: 9cm
}
.pos-zindexbehind {z-index: -1;
  position: absolute;
  left: 10cm
}
</style>
</head>
<body>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
<pre><div class=whitebg>****Positioning Properties*****
1. clip                   - set shape of box
2. overflow               - set behaviour if size of content is
large than the shape
3. position               - place box using a scheme
4. top, right, bottom, left - place box in one of four directions
5. vertical-align         - arrange text vertically
6. z-index                - place text perpendicular to the screen
at or below the top level
Not demonstrated: 1, 2
</div></pre>
<!-- DEMONSTRATIONS START HERE -->
<hr color=CornflowerBlue size=5px />
<p class=pos-positionstatic>p: position-static (normal) one two
three four five six seven eight nine ten eleven twelve thirteen
fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=pos-positionrelative1>p: position-relative (left -1cm)
one two three four five six seven eight nine ten eleven twelve
thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
```

```
<p class=pos-positionrelative2>p: position-relative (left 1cm)
one two three four five six seven eight nine ten eleven twelve
thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=pos-positionabsolute1>p: position-absolute (left 15cm
top 5cm) one two three four five six seven eight nine ten eleven
twelve thirteen fourteen
fifteen sixteen seventeen eighteen nineteen twenty twenty-one
twenty-two twenty-three twenty-four twenty-five</p>
<p class=pos-positionfixed1>p: position-fixed (left 15cm top 3cm)
one two three four five</p>
<hr color=CornflowerBlue size=5px />
span: vertical-align (sub) Water is H<span
class=pos-verticalalignsub>2</span>O<br />
span: vertical-align (super) One kilobyte is 2<span
class=pos-verticalalignsuper>10</span> bytes<br />
img: vertical-align (top) This image <img
class=pos-verticalaligntop src='logo.jpg' /> is a logo.<br />
img: vertical-align (middle) This image <img
class=pos-verticalalignmid src='logo.jpg' /> is a logo.<br />
img: vertical-align (bottom) This image <img
class=pos-verticalalignbot src='logo.jpg' /> is a logo.
<hr color=CornflowerBlue size=5px />
<div class=whitebg>
img: z-index (infront) This image <img class=pos-zindexfront
src='logo.jpg' /> is a logo.
one two three four five six seven eight nine ten</div>
<div class=whitebg>
img: z-index (behind) This image <img class=pos-zindexbehind
src='logo.jpg' /> is a logo.
one two three four five six seven eight nine ten</div>
<hr color=CornflowerBlue size=5px />
</body>
</html>
```
(End of the web page)

Click the link, html, for Positioning in the menu. Figure 8.10 shows the display of the web page, positioning.htm.

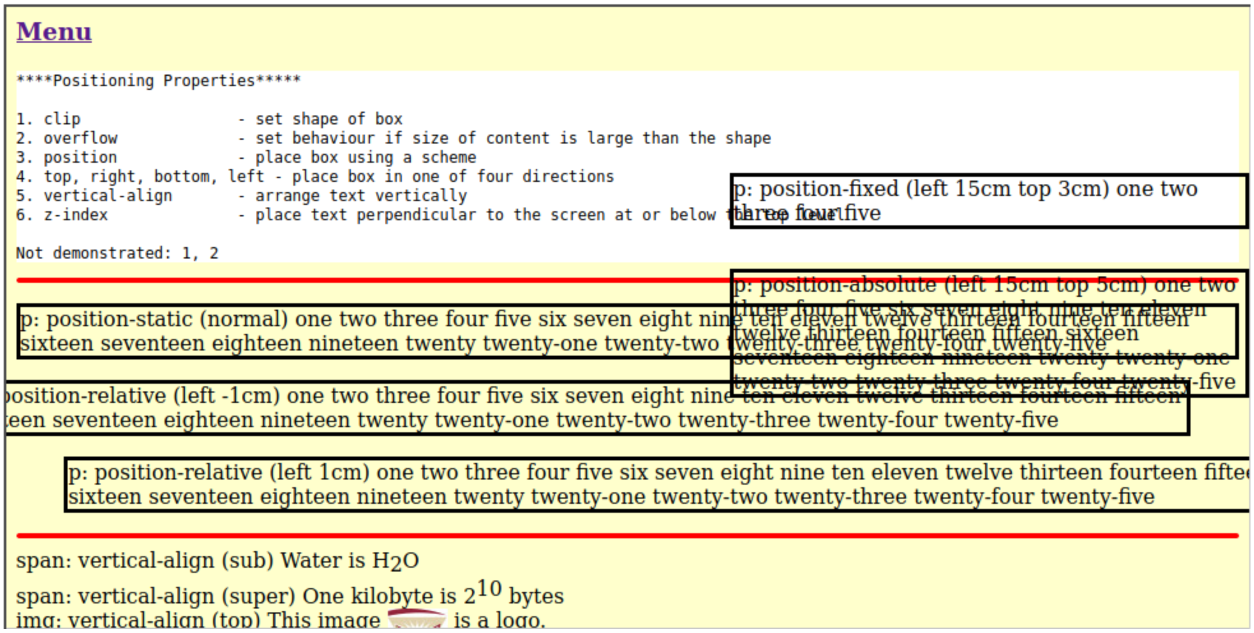
8.8.9 Other

The other properties are listed in Figure 8.1. They are demonstrated in the next web page.

Web Page 8.9 other.htm

It is assumed that the user has remotely logged-in to the server and is using the Linux operating system.

Type the following file in an editor and save it in directory, c08css, using the filename, other.htm. Then change the permissions of the file to 604.



Menu

```
****Positioning Properties*****

1. clip                  - set shape of box
2. overflow              - set behaviour if size of content is large than the shape
3. position              - place box using a scheme
4. top, right, bottom, left - place box in one of four directions
5. vertical-align        - arrange text vertically
6. z-index               - place text perpendicular to the screen at or below the top level

Not demonstrated: 1, 2
```

p: position-fixed (left 15cm top 3cm) one two three four five

p: position-absolute (left 15cm top 5cm) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

p: position-static (normal) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

osition-relative (left -1cm) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen een seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

p: position-relative (left 1cm) one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifte sixteen seventeen eighteen nineteen twenty twenty-one twenty-two twenty-three twenty-four twenty-five

span: vertical-align (sub) Water is H_2O

span: vertical-align (super) One kilobyte is 2^{10} bytes

img: vertical-align (top) This image is a logo.

Figure 8.10 Positioning web page: positioning.htm

(Type the following web page)

```
<html>
<head>
<title>Other properties</title>
<style type="text/css">
/********** background-color **********/
body  {
   background-color: #ffffcc;    /* light grey */
}
span.whitebg {
   background-color:white
}
div.whitebg {
   background-color:white
}
/**********margin **********/
p.margin1 {margin: 0 5cm 0 5cm;
   border-style: solid
}
/*********** 1. ***********
********** clear **********
***************************
values: none | left | right | both
default value: none
```

```
**************************/
/*********** 2. ************
********** cursor **********
****************************
values: auto | crosshair | default | pointer | move | e-resize |
ne-resize | nw-resize | n-resize |
   se-resize | sw-resize | s-resize | w-resize | text | wait |
progress | help
default value: auto
***************************/
.cursorauto {cursor: auto
}
.cursorcrosshair {cursor: crosshair
}
.cursordefault {cursor: default
}
.cursorpointer {cursor: pointer
}
.cursormove {cursor: move
}
.cursore-resize {cursor: e-resize
}
.cursorne-resize {cursor: ne-resize
}
.cursornw-resize {cursor: nw-resize
}
.cursorn-resize {cursor: n-resize
}
.cursorse-resize {cursor: se-resize
}
.cursorsw-resize {cursor: sw-resize
}
.cursors-resize {cursor: s-resize
}
.cursorw-resize {cursor: w-resize
}
.cursortext {cursor: text
}
.cursorwait {cursor: wait
}
.cursorprogress {cursor: progress
}
.cursorhelp {cursor: help
}
/************ 3. *********
********** display *******
**************************
```

```
values: inline | block | list-item | run-in | inline-block |
table | inline-table | table-row-group |
  table-header-group | table-footer-group | table-row |
table-column-group | table-column |
  table-cell | table-caption | none
default value: inline
*************************/
.displayinline {display: inline
}
.displayblock {display: block
}
.displaynone{display: none
}
/*********** 4. ***********
********** float **********
***************************
values: left | right | none
default value:  none
*************************/
.floatright1 {float:right;
  border-style: dotted;
  border-width: 2px
}
.floatright2 {float:right;
  border-style: dotted;
  border-width: 2px;
  margin: 0 0 0 4cm
}
.floatleft1 {float:left;
  border-style: dotted;
  border-width: 2px
}
.floatleft2 {float:left;
  border-style: dotted;
  border-width: 2px;
  margin: 0 4cm 0 0
}
.floatright3{float:right;
  border:2px dotted black;
  margin: 0 2cm 0 4cm;
  text-align:center
}
/************* 5. ***************
********** pseudoclass **********
*********************************
values: :active, :focus, :hover, :link, :visited
default value: none
```

```
********************************/
a:link {color: blue}
a:visited {color: red}
a:hover {color: yellow}
a:active {color: green}
a.link2:link {font-size: 125%}
a.link2:visited {font-size: 150%;
  text-decoration: none}
a.link2:hover {font-size: 175%}
a.link2:active {font-size: 200%}
.link3:link {background: yellow}
.link3:visited {background: lime;
  text-decoration: none}
.link3:hover {background: blue}
.link3:active {background: magenta}
/************* 6. **************
********** visibility **********
********************************
values: visible | hidden | collapse
default value: visible
*******************************/
.invis {visibility: hidden
}
</style>
</head>
<body>
<h3><a href= "menucss.htm" target=_top>Menu</a></h3>
<pre><div class=whitebg>*****Other Properties*****
1. clear        - prevents overlap of boxes placed by float
2. cursor       - type of cursor displayed when the mouse passes
over the box (image or text)
3. display      - box is displayed as block or inline
4. float        - box (image or text) is placed to the right or
left
5. pseudoclass  - links are displayed according to how the user
interacts with the links
6. visibility   - box is visible or invisible
Not demonstrated: 1
</div></pre>
<a name="begin"></a>
<!-- DEMONSTRATIONS START HERE -->
<hr color=CornflowerBlue size=5px />
<strong>span: cursor</strong><br />
<em>Move the mouse over the text to display the different
cursors.</em><br />
<span class=cursorauto>Auto</span><br />
<span class=cursorcrosshair>Crosshair</span><br />
```

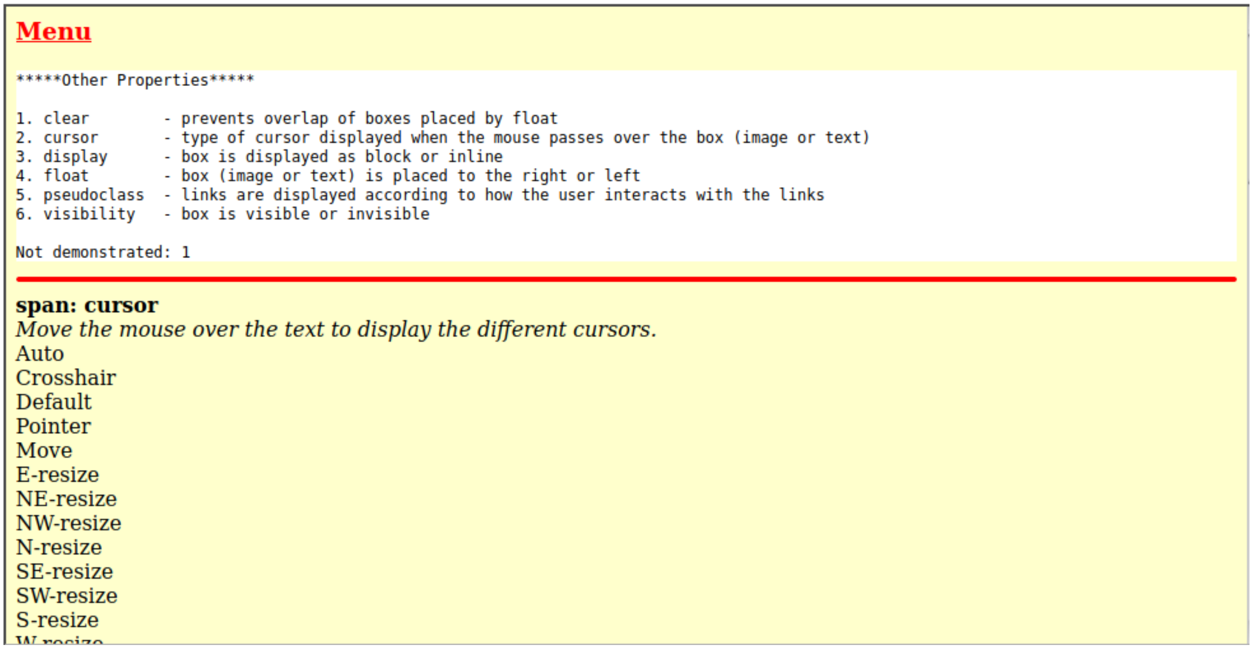
```
<span class=cursordefault>Default</span><br />
<span class=cursorpointer>Pointer</span><br />
<span class=cursormove>Move</span><br />
<span class=cursore-resize>E-resize</span><br />
<span class=cursorne-resize>NE-resize</span><br />
<span class=cursornw-resize>NW-resize</span><br />
<span class=cursorn-resize>N-resize</span><br />
<span class=cursorse-resize>SE-resize</span><br />
<span class=cursorsw-resize>SW-resize</span><br />
<span class=cursors-resize>S-resize</span><br />
<span class=cursorw-resize>W-resize</span><br />
<span class=cursortext>Text</span><br />
<span class=cursorwait>Wait</span><br />
<span class=cursorprogress>Progress</span><br />
<span class=cursorhelp>Help</span>
<hr color=CornflowerBlue size=5px />
<p class=displayinline>p: display (inline) one two three four
five</p>
<p class=displayinline>p: display (inline) one two three four
five</p>
<p class=displayblock>p: display (block) one two three four
five</p>
<p class=displayblock>p: display (block) one two three four
five</p>
<p><em>The line in the code, <strong>&lt;p
class=displaynone&gt;p: display (none)
one two three four five&lt;/p&gt;</strong>, is not displayed and
there is no space for the line.</em></p>
<p class=displaynone>p: display (none) one two three four
five</p>
<hr color=CornflowerBlue size=5px />
<p class=margin1><img class=floatright1 src='logo.jpg' />img:
float (right)
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
</p>
<br />
<p class=margin1><img class=floatright2 src='logo.jpg' />img:
float (right with left margin)
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
</p>
<br />
```

```
<p class=margin1><img class=floatleft1 src='logo.jpg' />img:
float (left)
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
</p>
<br />
<p class=margin1><img class=floatleft2 src='logo.jpg' />img:
float (left with right margin)
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
</p>
<br />
<p class=margin1><span class=floatright3><img src='logo.jpg'
/><br />Logo</span>
img: float (right with left and right margins, with text)
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
a b c d e f g h i j k l m n o p q r s t u v w x y z
</p>
<hr color=CornflowerBlue size=5px />
<strong>a:link, a:visited, a:hover, a:active</strong>
<br />
<em>Move the mouse over the links and click on the links.</em><br
/>
<a href="#begin">Link1</a><br />
<a class=link2 href="#end">Link2</a>
<br />
<a class=link3 href="#end">Link3</a>
<br />
<hr color=CornflowerBlue size=5px />
<p>6: visibility<p>
<p><em>The line in the code, <strong>&lt;p class=invis&gt;p:
visibility&lt;/p&gt;</strong>,
is not displayed, but there is space for the line.</em></p>
<p class=invis>p: visibility<p>
<p>p: visibility<p>
<hr color=CornflowerBlue size=5px />
<a name="end"></a>
</body>
</html>
```

(End of the web page)

Click the link, html, for Other in the menu.

Figure 8.11 shows the display of the top of the web page, other.htm.



Menu

```
*****Other Properties*****

1. clear        - prevents overlap of boxes placed by float
2. cursor       - type of cursor displayed when the mouse passes over the box (image or text)
3. display      - box is displayed as block or inline
4. float        - box (image or text) is placed to the right or left
5. pseudoclass  - links are displayed according to how the user interacts with the links
6. visibility   - box is visible or invisible

Not demonstrated: 1
```

span: cursor
Move the mouse over the text to display the different cursors.
Auto
Crosshair
Default
Pointer
Move
E-resize
NE-resize
NW-resize
N-resize
SE-resize
SW-resize
S-resize
W-resize

Figure 8.11 Other web page, top part: other.htm

9 PHP

9.1 Introduction

PHP is an open-source (free) scripting language. The letters of the name, PHP, stand for, PHP: Hypertext Preprocessor, where the P stands for PHP. This is an example of a recursive acronym, which is a popular naming method in computer science.

PHP was created in 1994 by Rasmus Lerdorf and was originally called Personal Home Page Tools (PHP Tools). He announced Version 1.0 in 1995 and Version 2.0 in 1996. By 1997 other people began to help Lerdorf to develop PHP. In 1998 Version 3 was released by the PHP team and then in 2000, Version 4 was released. PHP 5 was released in 2004.

PHP is a scripting language and so the programs that are written in PHP are called scripts. A script is interpreted as it is executed in contrast to a compiled program, like C for example. A compiled program is first translated into machine language by a specialized program called a compiler and then executed.

PHP has sets of functions to access most major databases including Oracle and MySQL.

9.2 Installation of PHP

PHP can be installed on a home computer so that web pages with PHP code can be tested before transferring them to a web site on a server. However, it is not sufficient to download and install only PHP. A http (web) server must also be downloaded and installed.

The installation of PHP by the software package, EasyPHP, is described in Chapter 2. PHP, Apache (web server) and MySQL (database program) are installed by EasyPHP at the same time.

The need for the installation of PHP on a home computer can be demonstrated as follows. Display a web page which contains only HTML code (or HTML and JavaScript code) on the home computer. The web page is read from the hard disk of the home computer (the computer with the browser) and displayed on the screen by clicking File/Open File . . . in Firefox or File/Open . . . in Internet Explorer and selecting the file to be displayed. The display of the web page is the same as if it was retrieved on the internet from a web server.

Then display a web page which contains both HTML code and PHP code on the home computer in the same way. The output of the HTML code is displayed but only the PHP source code is displayed since a browser does not execute PHP code. View the entire source code of the web page by clicking View/Page Source in Firefox or View/Source in Internet Explorer.

9.3 Server-side Scripting Language

PHP is primarily used as a server-side scripting language which is embedded (contained) in HTML web pages. Also, it is possible to have a web page which contains only PHP instructions, that is, a web page without HTML instructions. The PHP instructions in a web page are executed in the server computer before the web page is sent to the client computer. This is the reason PHP is called a server-side scripting language.

There are two other uses of PHP: (a) command-line scripting language (like Linux/UNIX shells) where a script is executed from the command line, and (b) client-side scripting language. These uses are not common and will not be discussed in this chapter.

PHP instructions are executed by a program called the PHP interpreter. The PHP interpreter is a module of the web server, that is, it resides in main memory of the server computer together with the web server. So when a web page contains PHP instructions, the web server effectively calls the PHP interpreter to execute the instructions before the web page is sent to the browser.

A web page with PHP code is processed as follows. A request for the web page is sent by a browser to a web server. The web server retrieves the web page from the hard disk of the server computer. If the web server finds any PHP code in the web page, the server sends the code to the PHP interpreter which executes the PHP code. The output of PHP instructions can be HTML instructions. Any HTML instructions generated by the PHP code are sent back to the web server which incorporates the HTML instructions in the web page in place of the PHP code. In this way PHP modifies a web page and thus produces content in a web page. After all PHP code is executed, the server sends the modified web page, which now consists entirely of HTML code (or HTML and JavaScript code) to the browser. The browser interprets the HTML code and displays the output on the screen.

A web page with PHP instructions that change (add, modify or remove) HTML instructions in the web page is called a dynamic web page, the topic of the next section.

9.4 Dynamic web page

Web pages served by a web server, such as Apache, can be either static (unchanging) or dynamic (changing).

The contents of a static web page do not change and the HTML instructions sent by the web server to a browser produce the same display on the screen each time the same page is requested by a browser. A static web page does not include PHP instructions. A static web page can only be changed by editing the HTML instructions for the page and then storing the modified web page in place of the original web page on the server hard disk. When a browser requests a static web page from the web server, the server sends the file from the server hard disk directly (without any processing of the web page) to the browser.

A dynamic web page is a page that can change each time the web server sends the page to a browser. It contains PHP instructions that can change the content that is sent to the browser and therefore change the display on the screen. When a browser requests a dynamic web page from the web server, the server sends the web page to the PHP interpreter. The interpreter executes the PHP instructions and then replaces the PHP instructions in the web page with the output of the PHP instructions when the output is HTML instructions. In this way the HTML instructions are changed by the PHP instructions in the web page. Then the web server sends the modified web page to the browser.

Two simple examples of a dynamic web page are the following.
(1) A page that contains PHP instructions to insert the date and time of day into the page. There are PHP functions that output the date and time in various formats, which would be used.
(2) A page that selects at random one of several greetings (hello, world; hi there; how are you; . . .) which is displayed each time the web page is sent to the browser. There is a PHP function which generates a random number, which would be used.

With reference to the second example, the dynamic web page cannot send the different greetings in a particular order each time the web page is requested because the server has no memory of the last greeting that was sent. In the jargon of computer science one describes this lack of memory by saying that the protocol used to send web pages is stateless. Various methods have been developed to maintain state, which is the subject of a later chapter.

9.5 PHP tags

In general, there are three types of tags in a web page: PHP tags (or tags for some other server-side scripting language), JavaScript tags (or tags for some other client-side scripting language) and HTML tags.

The PHP instructions which are embedded in a web page must be enclosed within a pair of tags. Each pair of tags encloses one or more instructions which are sent to the PHP interpreter. The instructions outside the PHP tags are ignored by the PHP interpreter. The preferred PHP tags are the pair, <?php and ?>, because they are fully portable. If there is one set of PHP instructions embedded in HTML instructions, the file has the following form.

HTML instructions
<?php *PHP instructions* ?>
HTML instructions

There are also three other pairs of tags which are used to enclose PHP instructions. They are

<? ?>, <% %>, <script language="php"> </script>

All four pairs of tags are demonstrated later.

The other two types of tags enclose JavaScript and HTML and are used strictly by the browser on the client side (instead of on the server side in the case of PHP). JavaScript instructions are stored inside a pair of JavaScript tags which cause a JavaScript interpreter in the browser to execute the instructions. Instructions for other content (text, images, links) in a web page are stored inside a variety of HTML tags which are interpreted and rendered (displayed) by the browser.

9.6 Names of HTML and PHP files

The names of HTML and PHP files (web pages) are distinguished by their extension (suffix). The extensions are specified in the configuration file of the web server. By convention, the following extensions are used by most web servers.
(a) HTML file. A file with only HTML instructions is called a HTML file. It has the extension, .htm or .html. For example, the file with name, webpage1.htm, and the file with name, webpage2.html, are HTML files.

(b) PHP file. A file which contains any PHP instructions is called a PHP file. A PHP file can contain both HTML instructions and PHP instructions, or can contain only PHP instructions. It has the extension, .php. Also, older versions of configuration files may include by default the extensions, .php3 and .php4, to indicate the version (3 or 4) of the PHP code in the PHP files . For example, the file with name, webpage3.php, is a PHP file.
HTML files (files with extensions, .html and .htm) are sent by the web server directly to the browser. Any PHP source code in a HTML file would be displayed, since it was not executed (and so not removed) by the PHP interpreter. PHP files (files with the extension, .php, .php3 and .php4) are sent by the web server first to the PHP interpreter, where the PHP instructions are executed. Then the processed file is sent to the browser.

A server can be configured to recognize extensions in addition to those in (a) and (b) above. For example, a file with extension, .html, could be specified as a PHP file and therefore could contain PHP instructions. However, it is good programming practice to use only the extension, .php, for a file when PHP code is included in the file.

9.7 Syntax of PHP

The syntax of PHP is modeled on the syntax of the C language. Also the syntax of C++ and Java is based on the syntax of C. So, needless to say, a programmer who has a background in any of C, C++ or Java will readily understand PHP. For the benefit of programmers who have a knowledge of any C-like language, at the end of some subsections, the syntax of C and PHP is compared so that new knowledge of PHP can be built on the existing knowledge of C.

But do not despair if you are not familiar with C, C++ or Java, because PHP is simpler than those languages. It is easier to learn PHP first and then C (or C++ or Java), instead of C and then PHP. The treatment of PHP in this section is compact but a motivated beginner should be able to understand the demonstration programs later in the chapter. Any references to the C language in this section can be ignored by a novice programmer.

A good PHP tutorial for beginners is offered online by W3Schools. The web site is given in the references at the end of the chapter. A beginner should read this entire section to get an overview of PHP and then supplement each subsection, if necessary, by the corresponding topics presented by W3Schools.

The subsections are: 1. Comments, 2. Output, 3. Variables, 4. Constants, 5. Strings, 6. Arrays, 7. Objects, 8. Control Structures, 9. Functions. There are examples of instructions for all subsections except the first, Comments.

9.7.1 Comments

Comments are characters in a program which are ignored when the program is executed. Comments are used mainly to include documentation in a program, so as to make the program easier to understand when the programmer or someone else is reading the program. Comments are also used to temporarily remove instructions from a program without deleting the instructions during the development of a program.

There are three methods to insert comments into a PHP script.

9.7.1.1 Shell symbol for comments:

The number sign, #, can be inserted anywhere in a line and all characters after the # to the end of the line are considered by the PHP interpreter to be a comment. The comment that starts with the symbol, #, is terminated by the newline character. This is a single-line comment.

9.7.1.2 C++ method: //

The double slash, //, has the same function as the number sign, #. This is a single-line comment.

9.7.1.3 C method: /* */

The slash-asterisk pair of characters, /*, is used to indicate the start of a comment as in C and it can be inserted anywhere in a line. The comment is terminated by the asterisk-slash pair, */, which can be inserted anywhere after /* on the same line or on any line after the same line. All characters between the /* and the */ are considered part of the comment and are ignored by the PHP interpreter. This method applies to one or more lines, and so inserts multiline comments. In contrast, the other two methods apply only to one line, the line where the comment symbol appears.

PHP comments and HTML comments are different. The PHP comments appear inside the PHP tags and are inserted by the methods above. The HTML comments appear in the HTML instructions, outside the PHP tags. There is only one method to insert comments in HTML instructions: inside the pair of tags, <!-- and -->. The HTML tags insert multiline comments in HTML code just as the characters, /* */, insert multiline comments in PHP code.

9.7.2 Output

There are three ways to send output from PHP to the web page, which is then sent to the client browser: echo, print() and printf(). In addition, the function, sprintf(), is used to store a string in a variable so that the string can be output later by one of the three methods. The examples use variables, $name and $greeting. Variables are discussed in the next subsection. Quotation of strings is covered in the subsection, Strings.

9.7.2.1 echo

Echo in PHP is the same as echo in a shell (tcsh shell in Linux or UNIX, for example). It outputs a string to the browser. It does not return a value because it is not a function.

Example 2.1: echo

```
$name = "Bill";
echo "Hello, $name";          //Output: Hello, Bill
```

9.7.2.2 print()

print() is a function. It has only one argument which it outputs to the browser. It returns true if the string is displayed, false otherwise. The main difference between echo and print() is that print() returns a value and echo does not, and so print() can be used in an expression.

Example 2.2: print(), which produces the same output as above

```
$name = "Bill";
```

```
     print("Hello, $name");        //Output: Hello, Bill
```

9.7.2.3 printf()

printf() is a function which is the same as the printf() function in C. It formats a string which it outputs to the browser.

Example 2.3: printf(), which produces the same output as above

```
     $name = "Bill";
     printf("Hello, %s", $name);   //Output: Hello, Bill
```

9.7.2.4 sprintf()

The function, sprintf(), corresponds to the same function in C and is used to store a formatted string in a variable instead of displaying it, and so the string can be displayed elsewhere in the program. However, in C the string variable, in which the formatted string is stored, is in the first argument of the function. In PHP, the string is returned by the function and assigned to a variable.

Example 2.4: sprintf(), which produces the same output as above

```
     $name = "Bill";
     $greeting = sprintf("Hello, %s", $name);
     echo "$greeting";             //Output: Hello, Bill
```

Figure 9.1 shows both the PHP instructions and the display by the browser using echo, print(), printf() and sprintf(). The display by the browser is the same in all four cases: Hello, Bill.

9.7.3 Data and Variables

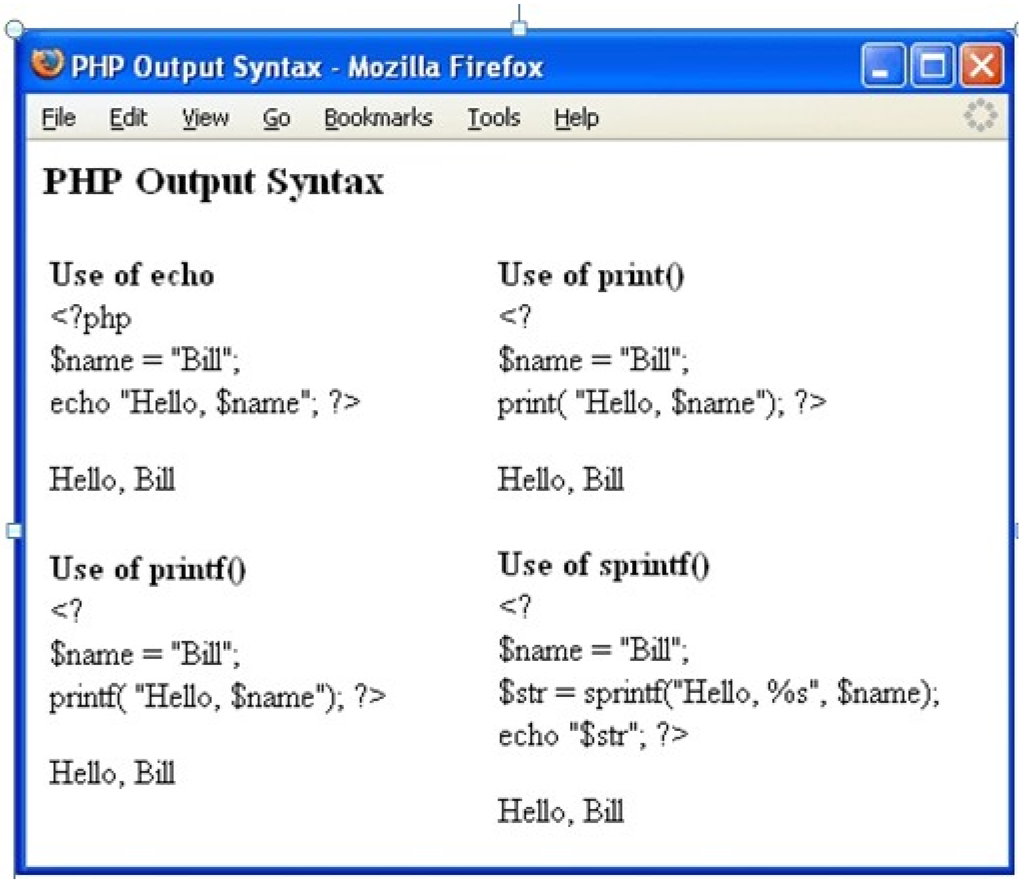
There are four basic (also called scalar) types of data in PHP: integer, floating point, string and boolean. There are two derived (also called compound) types of data in PHP: array and object (instance of a class). The number of bytes of main memory occupied by the data depend on the type of data which is stored.

A variable is a storage location in main memory which holds data. So a variable is characterized by two properties: an address (where it is in storage) and a value (what data is stored at the address). In a program the variable (the memory storage location) has a name which is used to refer to either the variable address or its value, depending on the context.

The name of variable in a PHP script refers only to the value of the variable. The name is prefixed by a dollar sign, $. The name of a variable in PHP (after the $) can consist of letters (a-z, A-Z), digits (0-9) and underscore (_) with the restriction that the name cannot start with a digit. The length of a variable name can be one or more characters. Some examples of valid names (including the mandatory prefix, $) are $var, $Var, $var2, $var_2, $_var, $_123. Examples of invalid names are $2var, $var-2, $var 2 (a space is not allowed in a name).

The name of a variable is case-sensitive. For example, $var and $Var are two different variables.

A variable is not declared in PHP. It is effectively declared the first time it is used. A variable can be used for the first time anywhere in a PHP script


PHP Output Syntax - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

PHP Output Syntax

Use of echo

```
<?php
$name = "Bill";
echo "Hello, $name"; ?>

Hello, Bill
```

Use of print()

```
<?
$name = "Bill";
print( "Hello, $name"); ?>

Hello, Bill
```

Use of printf()

```
<?
$name = "Bill";
printf( "Hello, $name"); ?>

Hello, Bill
```

Use of sprintf()

```
<?
$name = "Bill";
$str = sprintf("Hello, %s", $name);
echo "$str"; ?>

Hello, Bill
```

Figure 9.1 Output methods in a PHP script

.

A variable is not typed in PHP, that is, the programmer does not specify the data type of the variable. However, internally PHP does associate a type with a variable. A value of any type can be assigned to a variable and then later in the same script a value of a different type can be assigned to the same variable. For example, an integer value can be assigned to a variable: $var = 2, and later in the script a string can be assigned to the same variable: $var = "hello".

A variable is deleted by the function, unset().

There are functions to test for each type of variable. The function is passed the value of the variable and returns either TRUE or FALSE. The functions are classified as, Variable Handling Functions, in the online PHP Manual, given in the last section, References.

(a) Functions to test for basic data types
    is_int($var) - Finds whether the type of a variable is integer
    is_float($var) - Finds whether the type of a variable is float
    is_string($var) - Finds whether the type of a variable is string
    is_bool($var) - Finds whether the type of a variable is a boolean
(b) Functions to test for derived data types
    is_array($var) - Finds whether the type of a variable is an array

is_object($var) - Finds whether the type of a variable is an object
For example, if $var= 2, is_int($var) returns TRUE and all of the other functions return FALSE.
For example, if $var=”hello”, is_string($var) returns TRUE and all of the other functions return FALSE.

C and PHP
(1) Name: The name of a variable in C is not prefixed by a special character, unlike in PHP where the name of a variable is prefixed by the symbol, $.
(2) Value and address: The name of a variable in C refers to the value of the variable, like in PHP. The address of a variable in C is the name of the variable prefixed with the symbol, &. In C for example, &var is the address of the variable which has value, var. The address of a variable in PHP is not accessible.
(3) Declaration and type: A variable in C must be declared by specifying the data type, either before or at the same time that it is assigned a value. In C for example, an integer variable, var, is declared by the statement, int var, which does not assign a value to the variable or, int var = 1, which both declares var and assigns a value to var. Also, a variable in C must be declared in a function before the first instruction in the function.
In PHP a variable is not declared before it is used, a variable does not have a type specified by the programmer and a variable can be used for the first time anywhere in a function.
(4) Deletion: A variable in C cannot be deleted after it is declared, unlike in PHP.

Example 3.1: variable names are case sensitive

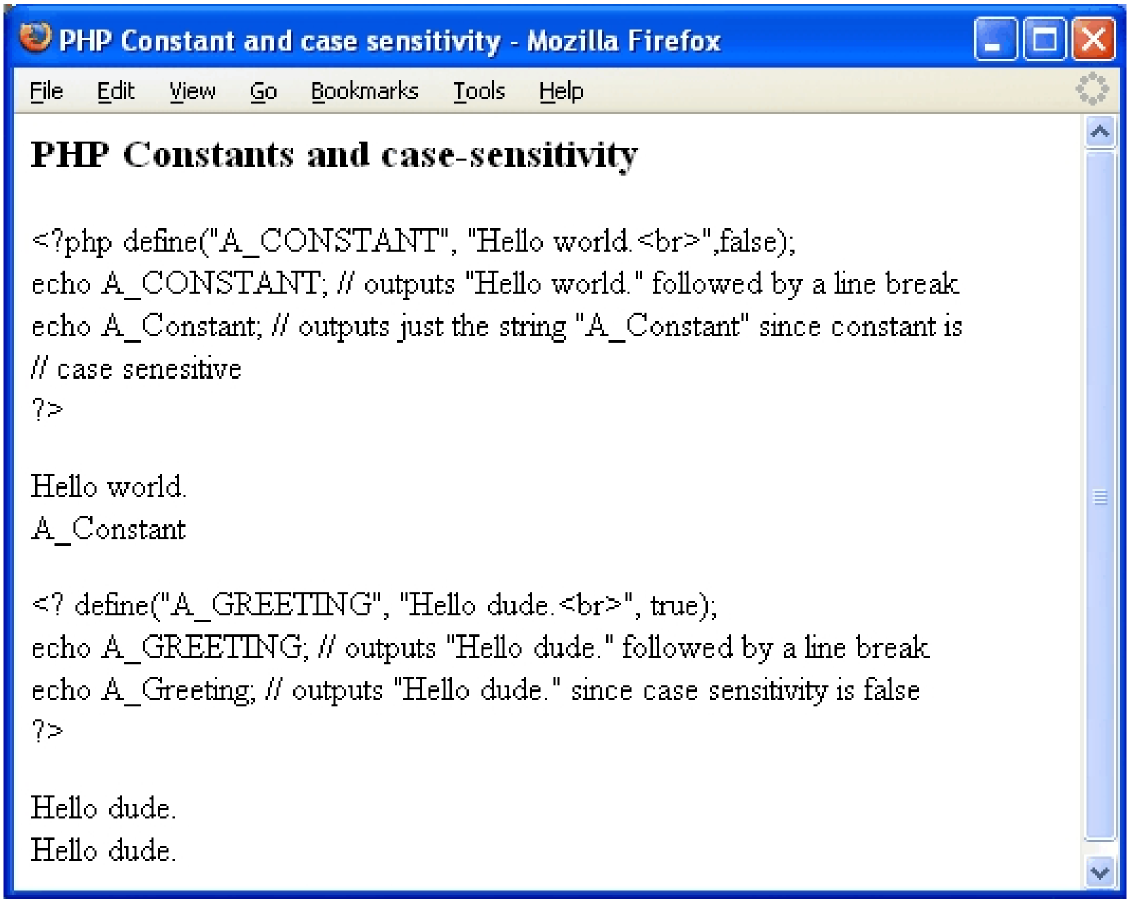
```
   $var = "Bill";
   $Var = 2;
   $VAR = "cats";
   echo "$var has $Var $VAR.";      //Output: Bill has 2 cats.
```

9.7.4 Constants

A constant is the value of one of the four basic data types with the restriction that the value cannot change. A constant is created by the function, define(), anywhere in a script. Once a constant is defined, it cannot be deleted and it cannot be changed, even with the define() function. After the definition, the constant exists until the end of the script. The name of a constant is not prefixed by the dollar sign, $, as for a variable. Constants have global scope, that is, they are accessible in any function in a script, no matter where they are defined. By convention the names of constants are written in uppercase so that they are easy to notice in the script.
The name of a constant is case sensitive by default. However, the name of a constant can be made case insensitive by setting the third argument of define() to true in the definition of the constant.

C and PHP
A constant in C is defined by the directive, define, which must be placed outside of all functions. A constant in PHP is defined by the function, define(), which can be called anywhere inside a function.

Figure 9.2 demonstrates both a constant which is case sensitive, third argument of define() is false, and case insensitive, third argument is true. The third argument can be omitted and in this case the default value of the third argument is false, that is, the constant is case sensitive. In Figure 9.2, both the PHP code and the output of the code after it is executed by the browser are shown.


PHP Constant and case sensitivity - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

PHP Constants and case-sensitivity

```
<?php define("A_CONSTANT", "Hello world.<br>",false);
echo A_CONSTANT; // outputs "Hello world." followed by a line break
echo A_Constant; // outputs just the string "A_Constant" since constant is
// case senesitive
?>

Hello world.
A_Constant

<? define("A_GREETING", "Hello dude.<br>", true);
echo A_GREETING; // outputs "Hello dude." followed by a line break
echo A_Greeting; // outputs "Hello dude." since case sensitivity is false
?>

Hello dude.
Hello dude.
```

Figure 9.2 Constant and case sensitivity of the name

In the following example, the third argument of define() is omitted and so the name of the constant is case sensitive.
The concatenation operator, (.), is covered in the next subsection.
Example 4.1: float constant

```
   define('PI', '3.14159');
   echo "<br>The reciprocal of PI = " . PI . " is 1/PI = " .
1/PI;
     //Output: The reciprocal of PI = 3.14159 is 1/PI =
0.31831015504888
   printf("<br>The square of PI = %f is PI*PI = %f", PI, PI*PI);
     //Output: The square of PI = 3.14159 is PI*PI = 9.869588
```

9.7.5 Strings

A string is a sequence of 0 or more characters. In PHP a string is a basic data type.

There are three ways to write a string in a PHP program: enclose the string in double quotes ("), single quotes (') and here document, usually called heredoc. A good practice is to quote a string by single quotes unless the string includes variables or escape sequences, in which case use double quotes. This practice is not always followed in this chapter.

9.7.5.1 Double-quoted strings

A string enclosed in double quotes, "*string*", allows variable interpolation and escape sequences. Variable interpolation means replacing a variable name by its value. An escape sequence is a backslash, (\), followed by a character without any intervening space, which together has a special meaning. Three common examples are: \n which means line feed (newline), \r means carriage return and \t means horizontal tab. Also, escape sequences are used to remove the special meaning of characters. Two examples are \", which permits the use of the double quote, ", in a string that is enclosed by double quotes (otherwise " in the string would terminate the string) and \$, which permits the use of $ in a string (otherwise $ is the prefix of the name of a variable).

9.7.5.2 Single-quoted strings

A string enclosed in single quotes, '*string*', does not allow variable interpolation and expands only two escape sequences: \', which puts a single quote in a single-quoted string, and \\, which puts one backslash in a single-quoted string.

9.7.5.3 Here document (heredoc) strings

A heredoc is used to put multiline strings in a program. It not necessary to insert newlines at the end of each line. Also, single and double quotes need not be escaped with \. Whitespace (spaces, tabs, newlines) are preserved in a heredoc.

Syntax: *stringname* <<< *identifier*
string
identifier;

The identifier, *identifier*, in the syntax can be any sequence of characters. The end of the string is terminated by the identifier which must be on a new line without any other characters before or after the identifier except the semicolon after the identifier.

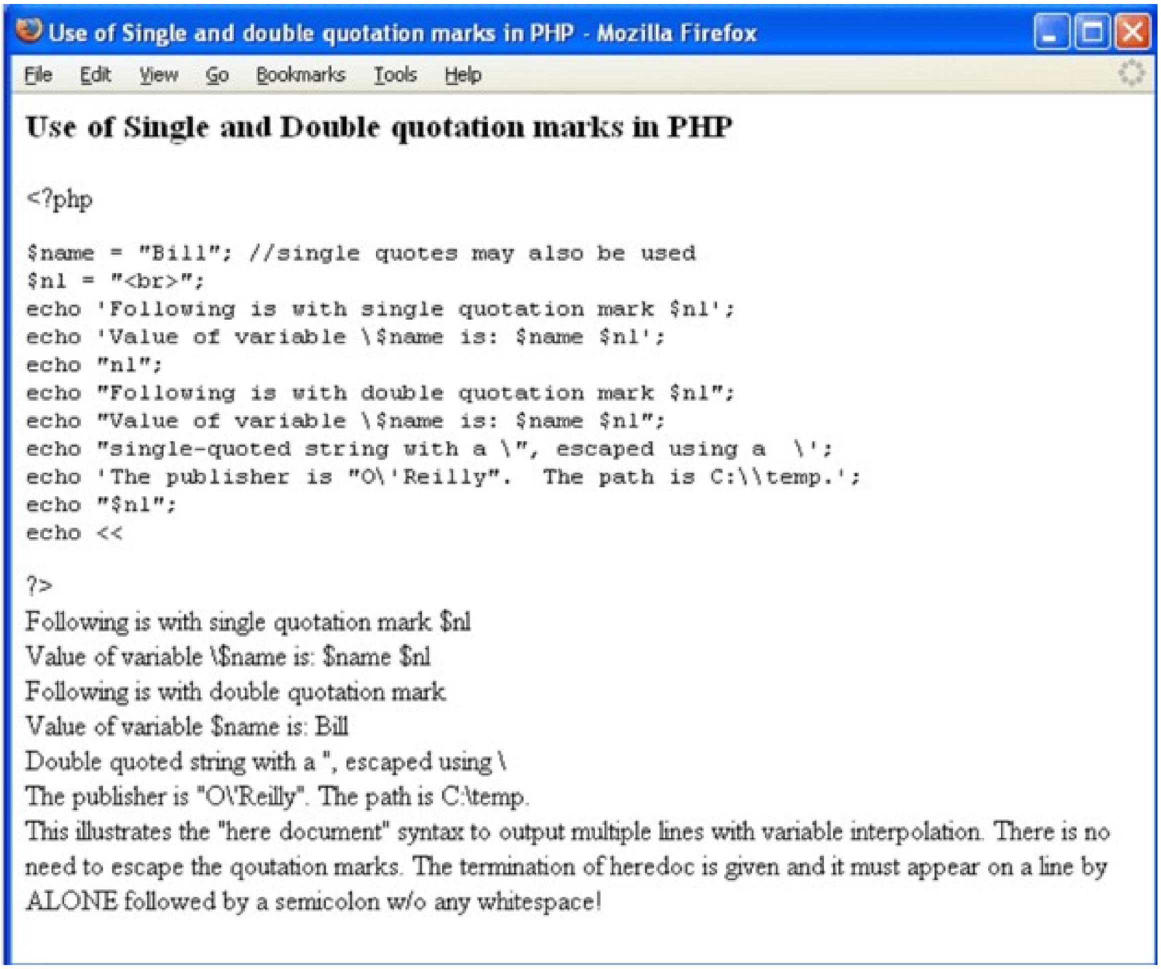
C and PHP

A string in C is a derived data type: an array of characters terminated by the null character, unlike in PHP where a string is a basic data type: an array of characters not terminated by the null character.

Figure 9.3 illustrates the use of single quotes, double quotes and heredoc. Both the PHP code and the output of the code after it is displayed by the browser are shown.

9.7.5.4 Concatenation of strings

Concatenation of strings is done by the dot (.) operator in PHP with syntax: *string1 . string2*. Also in PHP, a function which returns a string can be concatenated to another string. The verbs, concatenate and catenate, mean the same thing: to link together.


Use of Single and double quotation marks in PHP - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

Use of Single and Double quotation marks in PHP

```
<?php

$name = "Bill"; //single quotes may also be used
$nl = "<br>";
echo 'Following is with single quotation mark $nl';
echo 'Value of variable \$name is: $name $nl';
echo "nl";
echo "Following is with double quotation mark $nl";
echo "Value of variable \$name is: $name $nl";
echo "single-quoted string with a \", escaped using a  \';
echo 'The publisher is "O\'Reilly".  The path is C:\\temp.';
echo "$nl";
echo <<

?>
```

```
Following is with single quotation mark $nl
Value of variable \$name is: $name $nl
Following is with double quotation mark
Value of variable $name is: Bill
Double quoted string with a ", escaped using \
The publisher is "O\'Reilly". The path is C:\temp.
This illustrates the "here document" syntax to output multiple lines with variable interpolation. There is no
need to escape the qoutation marks. The termination of heredoc is given and it must appear on a line by
ALONE followed by a semicolon w/o any whitespace!
```

Figure 9.3 Single and double quotes and variables replacement

C and PHP
In C the library function, strcat(*string1*, *string2*), is used to concatenate strings, whereas in PHP strings are concatenated by the operator, dot (.).

Example 5.1: concatenation

```
$name = 'Bill';
echo 'Hello ' . $name . '.';       Output: Hello, Bill.
echo 'The time is ' . time() . ' seconds after the beginning
     of 1970.';
     //Output: The time is 1106607559 seconds after the
     beginning
     //    of 1970.
```

9.7.6 Arrays

An array is a group of zero or more variables. An array is a derived data type in PHP. There are two kinds of arrays in PHP: indexed arrays and associative arrays. The variables in an indexed array are organized in index-value pairs, where the index is an integer that is associated with the value of the variable in the array. The variables in an associative array consist of key-value pairs, where the key is a string that is associated with the value of the variable.

The index of an indexed array is an integer which can have any value, and in particular the index of the first array element does not need to be 0 (as in C) and the index of the next array element need not increase by 1 (as in C). So the index does not, in general, identify the position of the variable in the array (as in C).

The key of an associative array is a string and can have any value for each element (value) in the array. An associative array can be visualized as a table with two columns where the first column is the key and the second column is the value (of the variable). PHP stores both types of arrays internally as associative arrays; the only difference being whether an integer or string is used for the key.

An array can be created by assigning values to the array or by using the construct, array(). Since data types are not declared in PHP, different elements (cells) of an array in PHP can have different data types.

C and PHP:
An array in C is a derived data type, like an array in PHP. There are only indexed arrays in C, where the first value of the array has index 0, the second value has index 1 and so on. The array in PHP is a generalization of the array in C. In PHP there are both indexed arrays, like in C, and associate arrays as explained above.

An array in C is created by declaring an array with a particular data type. In C for example, char a1[10], declares an array of characters with name, a1, which can contain up to 10 characters. All elements in an array in C must have the same data type, char in the example above. In PHP an array is not declared and elements in the same array can have different data types.

Example of indexed array, created by assigning values:

```
   $weekdays[1] = 'Sunday';
   $weekdays[2] = 'Monday';
   . . .
   $weekdays[7] = 'Saturday';
```

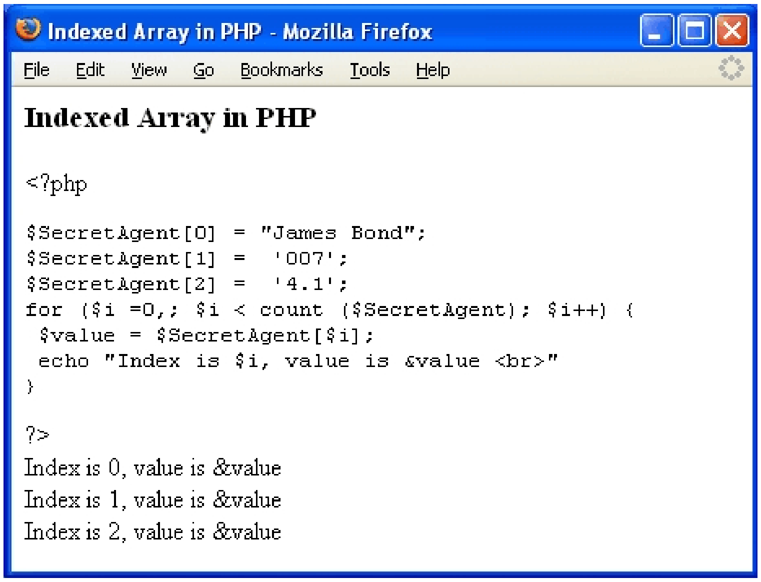
Example of associative array, created by assigning values:

```
   $SecretAgent['name'] = 'James Bond';
   $SecretAgent['id'] = '007';
   $SecretAgent['gpa'] = '4.1';
```

Example of the same associative array, created by using array():

```
   $SecretAgent = array('name' => 'James Bond', 'id' => '007',
      'gpa' => '4.1');
```

Figure 9.4 demonstrates the creation of an indexed array and the display of its indices and values. Figure 9.5 demonstrates the creation of an associate array and the display of its keys and values.



Indexed Array in PHP

```
<?php

$SecretAgent[0] = "James Bond";
$SecretAgent[1] =  '007';
$SecretAgent[2] =  '4.1';
for ($i =0,; $i < count ($SecretAgent); $i++) {
 $value = $SecretAgent[$i];
 echo "Index is $i, value is &value <br>"
}

?>
Index is 0, value is &value
Index is 1, value is &value
Index is 2, value is &value
```

Figure 9.4 Indexed array in PHP

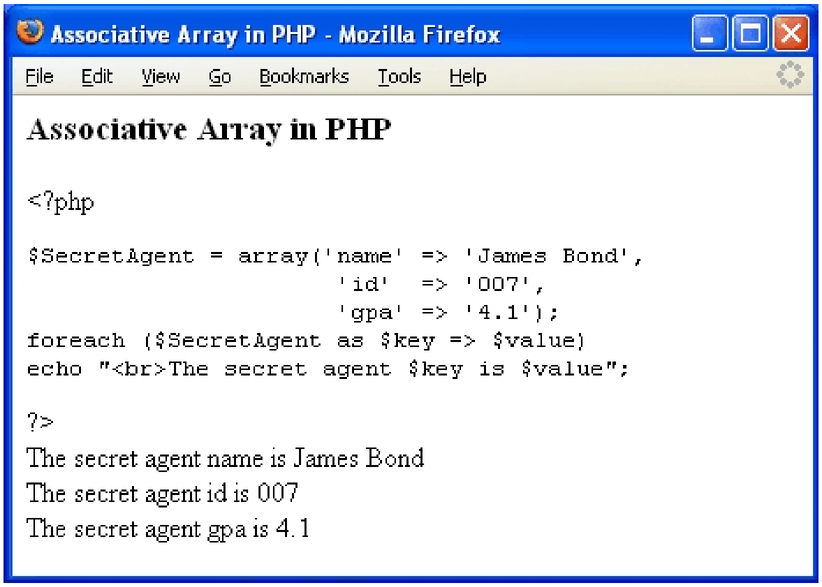
The function, is_array(), tests if a variable is an array. For example, if $aday = 'Monday', then $aday is not an array and so is_array($aday) returns 0 (or false). $SecretAgent is an array and so is_array($SecretAgent) returns 1 (or true).

The functions, sizeof() and count(), are identical; both return the number of elements in the array. In C the function, sizeof(), performs the same operation.
The function, unset(), deletes an array, or any other variable. It would be necessary to delete the array, $SecretAgent, if a secret agent retires or expires.

Example 6.1: array

```
$SecretAgent = array('name' => 'James Bond', 'id' => '007',
   'gpa' => '4.1');
echo $SecretAgent;      //displays, Array, since the array
                        exists
echo $Secretagent;      //displays nothing since the array does
                        not exist
$size = sizeof($SecretAgent);
echo "<br>size = $size array elements";
echo "<br>$SecretAgent[name] is $SecretAgent[id]";   //keys are
                                                     not quoted
if (isset($SecretAgent))
   echo "<br>\$SecretAgent exists";
else
   echo "<br>\$SecretAgent does not exist";
unset($SecretAgent);   //delete array
if (isset($SecretAgent))
   echo "<br>\$SecretAgent exists";
```

```
else
  echo "<br>\$SecretAgent does not exist";
```


Associative Array in PHP - Mozilla Firefox
File Edit View Go Bookmarks Tools Help

Associative Array in PHP

```
<?php

$SecretAgent = array('name' => 'James Bond',
                     'id'   => '007',
                     'gpa'  => '4.1');
foreach ($SecretAgent as $key => $value)
echo "<br>The secret agent $key is $value";

?>
The secret agent name is James Bond
The secret agent id is 007
The secret agent gpa is 4.1
```

Figure 9.5 Associative array in PHP

9.7.7 Objects

PHP is an object-oriented programming (OOP) language. The class is the basic entity of an OOP language. The class is a specification for a container of variables (properties) and functions (methods), collectively called the members of the class. An object is an instance of a class, that is, the object is the actual container that is specified by the class. A good analogy that illustrates the relationship between class and object is cookie cutter and cookie. One cookie cutter (class) is the template or pattern that is used to cut the cookie dough to make many cookies (objects).

C++ is an OOP language and the members of a class can be private, protected or public. In PHP there is no member access control, that is, all members of a class are public. By convention the methods in a class act only on the properties in the class, but this is not enforced by PHP since the members are public. A class name is not case sensitive, unlike a variable name which is case sensitive. So for example, the class names, SecretAgent and secretagent, refer to the same class.

The details of the syntax for class definition, object declaration and member access are different in PHP then in C++. The syntax in PHP is illustrated in the following example. Suppose there are several secret agents, each with a name, id and gpa, and the data about them is stored in a computer program. The data can be stored either in arrays, one for each secret agent, or in objects, also one for each secret agent but specified by only one class. The example of a secret agent has been used to demonstrate arrays previously.

To demonstrate objects it is sufficient to consider only two secret agents: James Bond and George Costanza. James has an id of 007 and a gpa (grade point average) of 4.1. George has an id of 001.5 and a gpa of 1.5; note that by coincidence the id and gpa of George are numerically the same.

Example 7.1: object
First the class is defined. In the example, there is one variable (property) in the class, the array which stores the data about a secret agent, and there are two functions (methods) in the array: one to get (retrieve) the data from the array and one to put (store) the data in the array.

After the class is defined, an object of the class is created for each secret agent. Then data is inserted into and extracted from the array for each secret agent by using the functions in the class.
A secret agent object (or any other variable) is deleted with the function, unset().

```
/////class definition/////
class SecretAgent {
   //class is the keyword to define a class
var $agent;
   //var is the keyword to define a variable (property)
function getagent () {
   //function is the keyword to define a function (method)
   //this: represents the object which calls the method
     return "name is " . $this->agent['name']
            . "<br>id is " . $this->agent['id']
            . "<br>gpa is " . $this->agent['gpa'];
  }
function putagent ($val1, $val2, $val3){
    $this->agent = array('name' => $val1, 'id' => $val2, 'gpa' =>
$val3);
  }
}

/////object declarations/////
$sa1 = new SecretAgent;
   //new is the keyword that creates an object of a class
$sa2 = new SECRETAgent;  //class name is not case sensitive

/////method calling/////
$sa1->putagent('James Bond', '007', '4.1');
   //call putagent() for object, sa1
echo "sa1:<br>" . $sa1->getagent();
   //call getagent() for object, sa1
$sa2->putagent('George Costanza', '001.5', '1.5');
echo "<br>sa2:<br>" . $sa2->getagent();
```

9.7.8 Control Structures

Control structures determine the order in which instructions in a program are executed. There are three kinds of control structures: sequential, selection and iteration. These structures can be used to achieve any flow of control without the use of unconditional jumps (the goto instruction in C, for example). Unconditional jumps should not be used in a program because the program becomes less organized and harder to understand.

9.7.8.1 Sequential

Sequential control structures are any instructions that do not alter the order of execution of instructions, that is, the instructions are executed in the order that they appear in the program.

9.7.8.2 Selection

Selection control structures produce a branch in a program, that is, a group of instructions is executed or not executed. There are three selection control structures in both PHP and C: if, if-else and switch. They are the same in both PHP and C although PHP allows both the C syntax and the shell syntax (tcsh shell of Linux or UNIX, for example).

9.7.8.3 Iteration

Iteration control structures produce a loop in a program, that is, a group of instructions are executed repeatedly zero or more times. There are three iteration control structures in C: while, do-while and for. In PHP there are the same iteration control structures as in C and in addition one more: foreach. The structure, foreach, is used most often to iterate over (traverse) the elements of an array.

For example, both the key and value of the array, SecretAgent, are displayed using the foreach control structure in the example which follows. Any other variables could be used for $key and $value; for example, $k and $v would save typing.

Example 9.1: foreach used to traverse an array

```
$SecretAgent = array('name' => 'James Bond', 'id' => '007',
   'gpa' => '4.1');
foreach ($SecretAgent as $key => $value)
   echo "<br>The secret agent $key is $value";
```

9.7.9 Functions

A function is a group of instructions which have a name. The definition of a function is the name of the function followed by the instructions which constitute the function. Arguments can be passed to a function and a value can be returned by a function. The arguments can be passed by value and passed by reference. The function definition starts with the keyword, function. The data type of the return value of a function is not specified since there are no data types in PHP.

A call to a function is an instruction which starts the execution of the instructions in the function. When the execution of the function is complete, the instruction following the call is executed next.

There are many built-in functions in PHP. Also, functions created by the programmer, called user-defined functions (instead of the expected, programmer-defined functions) can be written.

A function name can consist of letters (a-z, A-Z), digits (0-9) and underscore (_) with the restriction that the name cannot start with a digit. These are the same rules as for naming of variables, except there is no special mandatory prefix for a function name as there is for a variable name.

A function name is not case sensitive, unlike a variable name which is case sensitive. So for example, the function names, greeting and Greeting, refer to the same function.

C and PHP:

The syntax of a function definition in C is the same as in PHP. However, since variables do not have data types in PHP, the declaration a function in PHP is simpler than in C. For example, if a function in C is declared by, int function1(char arg1), the same function would be declared in PHP by, function1(arg1).

The following example demonstrates a function which gives someone a big welcome.
Two arguments, one integer and one string, are passed by value to the function and the function returns a string.

Example 9.1: user-defined function

```
   //function definition
   function greeting($n, $name) {
      for ($i = 0; $i < $n; $i++)
           echo "Welcome $name! ";
      return "<br><font color=red>*****$name was welcomed $n
           times***** </font>";
   }
   //variable definitions
   $num = 100;
   $firstname = 'Bill';
   //function call
   $returnvalue = GrEeTiNg($num, $firstname);      //function
                                                    names are
   //not case sensitive
   echo $returnvalue;
```

The following example demonstrates two built-in functions: strlen() and die(). strlen() is the same function as in C; it returns the number of characters in the string which is passed to it. If the string does not exist, strlen() returns -1. Most functions are designed to return -1 if they fail.

The function die() is called if a function returns -1, using the syntax demonstrated in the example below. die() displays the string which is passed to it and then calls exit() which terminates the script, and so any instructions that follow die() are not executed.
Example 9.2: built-in function

```
   $str = "PHP is wonderful";
   //$str exists
   $result = strlen($str) or die("<br>strlen(\$str) returned -
1");
   echo "Length of \"$str\" is $result characters.";
   //$string does not exist and so strlen() returns -1
   $result = strlen($string) or die("<br>strlen(\$string)
returned -1");
   echo "Length of \"$string\" is $result characters.";    //not
     executed
        //because die() terminates the script
```

9.8 Web Page Demonstrations

The web pages in this section must be stored on a server and then displayed by a browser. It is assumed that the programmer remotely logs in to the server. Also it is assumed that the server is running Linux. Some familiarity with Linux shell commands is assumed, although the required commands are given.

The programmer types the web page using a text editor, emacs for example, and stores the web page in the subdirectory, c09php, of the web home directory. The directory must have permissions, 707 (write permission is required because the interactive web page writes files on the server), and each web page must have permissions, 604. The web pages can be viewed by a browser on any computer connected to the Internet by typing the URL of the web page in address bar of the browser.

The following Linux commands create the directory and then change its permissions. First change the working directory to the web home directory, if necessary.

```
        pwd                          (display working directory)
        mkdir c09php                 (create directory, c09php)
        chmod 707 c09php             (change permissions of directory, c09php)
```

Change the working directory to c09php to prepare to store the web pages.

```
        cd c09php                    (change working directory to c09php)
```

9.8.1 Menu

For convenience, a web page with a menu is written that has links to all of the other web pages in this chapter. Each of the web pages in the menu has a link back to the menu web page. The links in the web pages assume that all web pages are stored in the same directory as the menu. The menu web page is given the name, menuphp.htm.

Web Page 9.1 menuphp.htm

Start a text editor in Linux, for example, emacs, for the file, menuphp.htm.

```
        emacs menuphp.htm
```

Type the following file, or copy and paste it, and then save it using the filename, menuphp.htm, in directory, c09php. After, exit emacs.

```
        ^x^s   (save file: press and hold Ctrl, then type xs)
        ^x^c   (exit emacs: press and hold Ctrl, then type xc)
```

In Linux it is necessary to set the permissions of all web pages to 604. The third digit of the permissions, 4, allows anyone to read (display) the web page. Set the permissions for the menu web page.

```
        chmod 604 menuphp.htm
```

```
<html>
<head>
<title>Menu for PHP</title>
</head>
<body>
<h2>
<a href="../index.html">Main Menu</a>
</h2>
<h1>Menu for PHP</h1>
<h3>
9.1 First PHP Page<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
<a href="hello.php">hello.php</a> or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="hello-withcode.php">hello-withcode.php</a> or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="interactive.php?fn=hello.php">interactive.php</a><br
/><br />
9.2 PHP Configuration<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="phpinfo.php">phpinfo.php</a> or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="phpinfo-withcode.php">phpinfo-withcode.php</a> or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="interactive.php?fn=phpinfo.php">interactive.php</a><br /
><br />
9.3 PHP Syntax<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="syntax.php">syntax.php</a> or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="syntax-withcode.php">syntax-withcode.php</a> or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="interactive.php?fn=syntax.php">interactive.php</a><br
/><br />
9.4 HTML form and PHP script in separate files<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a href="cal.htm">cal.htm,
cal.php</a> or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="cal-withcode.htm">cal-withcode.htm, cal-withcode.php</a>
or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="interactive.php?fn=cal.htm">interactive.php</a><br /><br
/>
9.5 HTML form and PHP script in one file<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="cal_onefile.php">cal_onefile.php</a> or<br />
```

```
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp
<a href="cal_onefile-withcode.php">cal_onefile-withcode.php</a>
or<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
<a
href="interactive.php?fn=cal_onefile.php">interactive.php</a><br
/>
</h3>
</body>
</html>
```

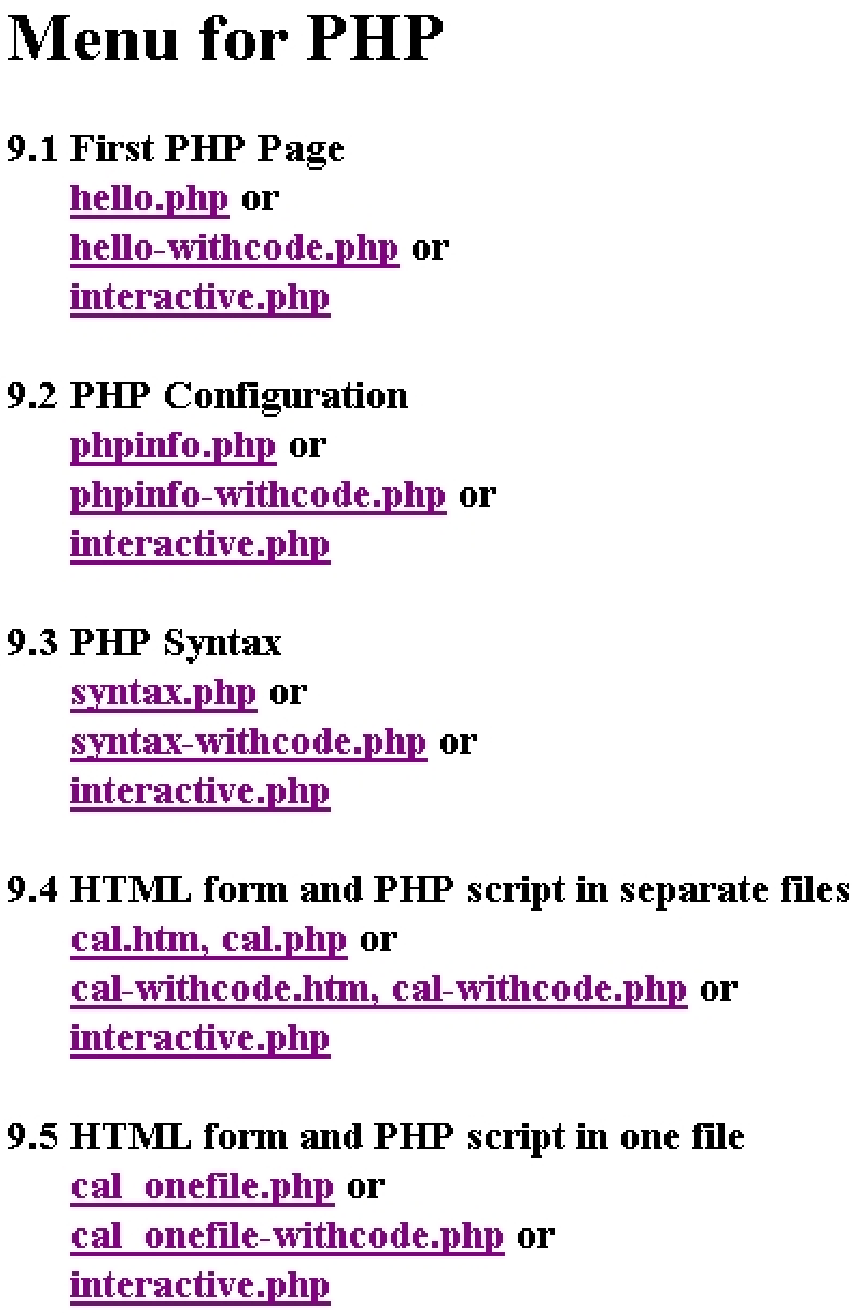
Figure 9.6 shows the menu as displayed by the browser. There are three web pages for each demonstration. For example, the demonstration of headers, has the three web pages, hello.php, hello-withcode.php and interactive.php.
(a) The web page, hello.php, shows only the output of the HTML source code.
(b) The web page, hello-withcode.php, shows both the source code itself, given in hello.php, followed by the output of the source code. This web page should be used to associate the lines of source code with the output of the corresponding source code, to learn PHP.
(c) The web page, interactive.php, shows the source code on the left side of the page and the output of the source code on the right side. The source code can be changed by typing on the left side and then Submit can be clicked to see the output of the changed source code on the right side. The original source code and its output can be displayed by clicking, original page.

The source code itself is inserted into a web page, as in hello-withcode.php, by simply replacing the two characters which are the tag delimiters, < and > , by the escape sequences for the characters, &lt; for < and &gt; for >, and enclosing the entire code in the tags, <pre> and </pre>.

9.8.2 First PHP web page

By tradition the first program in any programming language displays the message, hello, world. (This is the output of the first program in the classic book, The C Programming Language, by Kernighan and Ritchie, first published in 1978.)

There are four different pairs of tags that can be used to enclose PHP instructions in a web page. All tags are used in the web page below to output, hello, world, using echo. The preferred pair of tags is, <?php and ?>, because they are fully portable.



Menu for PHP

9.1 First PHP Page
hello.php or
hello-withcode.php or
interactive.php

9.2 PHP Configuration
phpinfo.php or
phpinfo-withcode.php or
interactive.php

9.3 PHP Syntax
syntax.php or
syntax-withcode.php or
interactive.php

9.4 HTML form and PHP script in separate files
cal.htm, cal.php or
cal-withcode.htm, cal-withcode.php or
interactive.php

9.5 HTML form and PHP script in one file
cal_onefile.php or
cal_onefile-withcode.php or
interactive.php

Figure 9.6 Menu for web page: menuphp.htm

Web Page 9.2 hello.php
Start a text editor, emacs for example, and type the following file, or copy and paste it. Save it in directory, c09php, using the filename, hello.php. Then change the permissions of the file to 604.

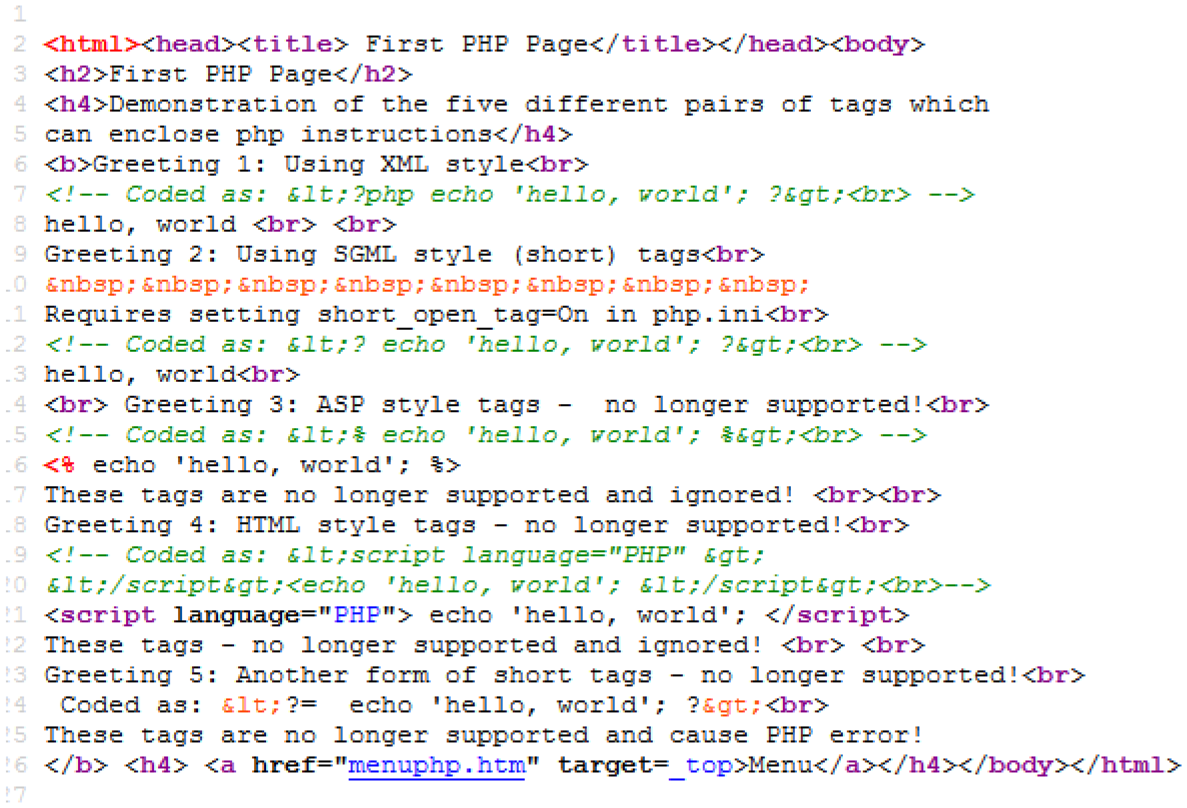
```
<html>
<head>
<title> First PHP Page</title>
</head>
```

```
<body>
<h2>First PHP Page</h2>
<h4>Demonstration of the five different pairs of tags which
can enclose php instructions</h4>
<b>Greeting 1: Using XML style<br>
<!-- Coded as: &lt;?php echo 'hello, world'; ?&gt;<br> -->
<?php echo 'hello, world'; ?> <br> <br>
Greeting 2: Using SGML style (short) tags<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
Requires setting short_open_tag=On in php.ini<br>
<!-- Coded as: &lt;? echo 'hello, world'; ?&gt;<br> -->
<? echo 'hello, world'; ?><br>
<br> Greeting 3: ASP style tags -  no longer supported!<br>
<!-- Coded as: &lt;% echo 'hello, world'; %&gt;<br> -->
<% echo 'hello, world'; %>
<?php echo 'These tags are no longer supported and ignored!'; ?>
<br><br>
Greeting 4: HTML style tags - no longer supported!<br>
<!-- Coded as: &lt;script language="PHP" &gt;
&lt;/script&gt;<echo 'hello, world'; &lt;/script&gt;<br>-->
<script language="PHP"> echo 'hello, world'; </script>
<?php echo 'These tags - no longer supported and ignored!'; ?>
<br> <br>
Greeting 5: Another form of short tags - no longer supported!<br>
 Coded as: &lt;?=  echo 'hello, world'; ?&gt;<br>
<?php echo 'These tags are no longer supported and cause PHP
error!'; ?>
</b>
<h4><a href="menuphp.htm" target=_top>Menu</a></h4>
```

Click the link, hello.php, in the menu.

When the request by the browser is received by the web server, the web server finds the file on the hard disk and transfers it into main memory. Then it sends the file to the PHP interpreter (which resides in main memory with the web server) since there are PHP instructions in the file, as indicated by the file extension, .php. Each time the interpreter reads a PHP tag, it executes the instructions within the PHP tags and replaces those instructions with the output of the instructions. So in the file above, for example, <?php echo >hello, world=?>, is replaced by, hello, world. The PHP interpreter sends the file back to the web server after the PHP instructions are executed. Then the web server sends the file to the web browser and the browser displays it on the screen.

Figure 9.7 shows source code of a web page, similar to Web Page 9.2, which uses the five different PHP tags, including the ones no longer supported. Figure 9.8 shows the display by a browser of the web page shown in Figure 9.7.



```

<html><head><title> First PHP Page</title></head><body>
<h2>First PHP Page</h2>
<h4>Demonstration of the five different pairs of tags which
can enclose php instructions</h4>
<b>Greeting 1: Using XML style<br>
<!-- Coded as: &lt;?php echo 'hello, world'; ?&gt;<br> -->
hello, world <br> <br>
Greeting 2: Using SGML style (short) tags<br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
Requires setting short_open_tag=On in php.ini<br>
<!-- Coded as: &lt;? echo 'hello, world'; ?&gt;<br> -->
hello, world<br>
<br> Greeting 3: ASP style tags -  no longer supported!<br>
<!-- Coded as: &lt;% echo 'hello, world'; %&gt;<br> -->
<% echo 'hello, world'; %>
These tags are no longer supported and ignored! <br><br>
Greeting 4: HTML style tags - no longer supported!<br>
<!-- Coded as: &lt;script language="PHP" &gt;
&lt;/script&gt;<echo 'hello, world'; &lt;/script&gt;<br>-->
<script language="PHP"> echo 'hello, world'; </script>
These tags - no longer supported and ignored! <br> <br>
Greeting 5: Another form of short tags - no longer supported!<br>
 Coded as: &lt;?=  echo 'hello, world'; ?&gt;<br>
These tags are no longer supported and cause PHP error!
</b> <h4> <a href="menuphp.htm" target=_top>Menu</a></h4></body></html>

```

Figure 9.7 Web page with PHP script using five types of tags

9.8.3 PHP configuration

The PHP function, phpinfo(), displays the configuration of the PHP interpreter, that is, how PHP was installed on the server and other information.

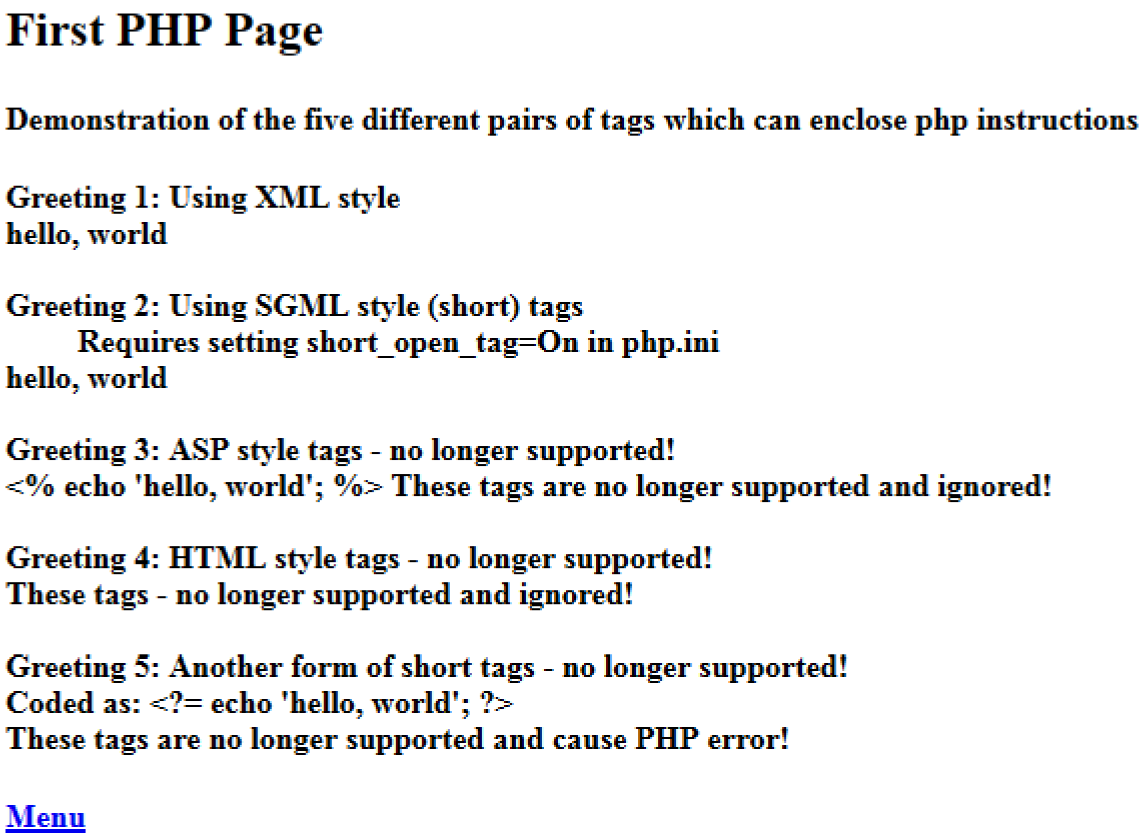
Web Page 9.3 phpinfo.php

Start a text editor, emacs for example. Type the following file, or copy and paste it, and save it in directory, c09php, using the filename, phpinfo.php. Then change the permissions of the file to 604.

```
<html>
     <head>
          <title> PHP Configuration
          </title>
     </head>
<body>
     <h1>PHP Configuration</h1>
     <h2>
     <a href="menuphp.htm" target=_top>Menu</a>
     <?php phpinfo(); ?>
     <a href="menuphp.htm" target=_top>Menu</a>
     </h2>
```

```
</body>
</html>
```

Click the link, phpinfo.php, in the menu.
The function, phpinfo(), is called by PHP which generates HTML instructions that the browser displays on the screen.



First PHP Page

Demonstration of the five different pairs of tags which can enclose php instructions

Greeting 1: Using XML style
hello, world

Greeting 2: Using SGML style (short) tags
Requires setting short_open_tag=On in php.ini
hello, world

Greeting 3: ASP style tags - no longer supported!
<% echo 'hello, world'; %> These tags are no longer supported and ignored!

Greeting 4: HTML style tags - no longer supported!
These tags - no longer supported and ignored!

Greeting 5: Another form of short tags - no longer supported!
Coded as: <?= echo 'hello, world'; ?>
These tags are no longer supported and cause PHP error!

Menu

Figure 9.8 The Web page displayed by the code of the last figure

9.8.4 PHP Syntax

The PHP syntax is covered in nine subsections of section 9.7, Syntax of PHP. There are demonstrations of the syntax in all subsections except the first, Comments. All programmers, including novices, will learn a number of programming techniques as well as syntax from the following file, syntax.php.

Web Page 9.4 syntax.php
Start a text editor and type the following file, or copy and paste it. Save it in directory, c09php, using the filename, syntax.php, and then change the permissions of the file to 604. If the file is typed, type the examples, one at a time. After typing each example, save the file and then run the PHP instructions by requesting the file in the browser the first time by typing the URL and then by refreshing the browser (press F5 in Internet Explorer, for example) each time thereafter.

```
<html>
<head>
<title> PHP Syntax
</title>
</head>
<body>
<h1>PHP Syntax</h1>
<h2>
<a href=”menuphp.htm” target=_top>Menu</a>
</h2>
<!--
*******************
*    1. Comments  *
*******************
There is no output for this subsection.
-->
<!--
*******************
*    2. Output    *
*******************
-->
<h2>2. Demonstration of different methods of output</h2>
<h3>
Example 2.1: echo
<?php
$name = "Bill";
echo "Hello, $name";
?>
<br>Example 2.2: print()
<?php
$name = "Bill";
print( "Hello, $name");
?>
<br>Example 2.3: printf()
<?php
$name = "Bill";
printf( "Hello, %s", $name);
?>
<br>Example 2.4: sprintf()
<?php
$name = "Bill";
$greeting = sprintf( "Hello, %s", $name);
echo "$greeting";
?>
</h3>
<!--
*******************
```

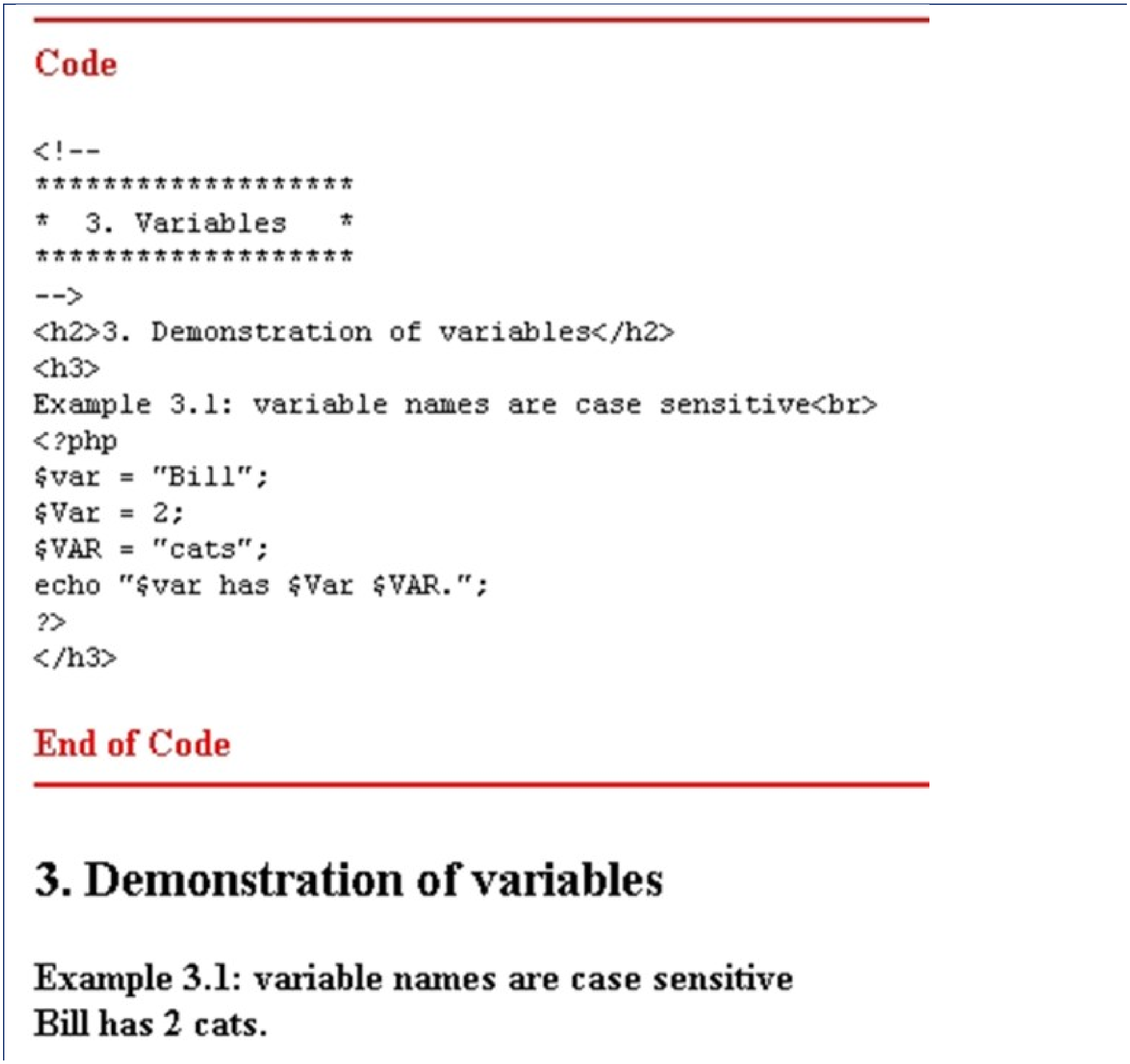
```
*  3. Variables   *
*******************
-->
<h2>3. Demonstration of variables</h2>
<h3>
Example 3.1: variable names are case sensitive
<?php
$var = "Bill";
$Var = 2;
$VAR = "cats";
echo "$var has $Var $VAR.";
?>
</h3>
<!--
*******************
*  4. Constants   *
*******************
-->
<h2>4. Demonstration of constants</h2>
<h3>
Example 4.1: string constant
<?php
define('COMPLIMENT', 'You look marvelous');
echo COMPLIMENT;
?>
<br>Example 4.2: float constant
<?php
define('PI', '3.14159');
echo "<br>The reciprocal of PI = " . PI . " is 1/PI = " .  1/PI;
printf("<br>The square of PI = %f is PI*PI = %f", PI, PI*PI);
?>
</h3>
<!--
*******************
*   5. Strings    *
*******************
-->
<h2>5. Demonstration of quotation of strings</h2>
<h3>
Example 5.1: double-quoted string
<?php
$name = "Bill";
echo "\$name = $name\n";
echo "Hello, $name\n";
?>
<br>Example 5.2: double-quoted string with " and '
<?php
```

```
$name = "Bill";
echo "It's \"$name\"\n";
?>
<br>Example 5.3: single-quoted string
<?php
$name = "Bill";
echo 'Hello, $name\n';
?>
<br>Example 5.4: single-quoted string with ", ' and \
<!-- &amp;#39; is the escape sequence for '  -->
<?php
echo 'The publisher is "O\'Reilly".  The path is C:\\temp.';
?>
<br>Example 5.5: heredoc<br>
<?php
$longstring = <<< endofstring
It is not necessary to escape
(put a \ in front of) a
single quote, ', and
double quote, ",
in a heredoc.
endofstring;
echo $longstring;
?>
<br>Example 5.6: concatenation
<?php
$name = 'Bill';
echo '<br>Hello ' . $name . '. ';
echo 'The time is ' . time() . ' seconds after the beginning of
1970.';
?>
</h3>
<!--
*******************
*    6. Arrays    *
*******************
-->
<h2>6. Demonstration of arrays</h2>
<h3>
Example 6.1: array<br>
<?php
$SecretAgent = array('name' => 'James Bond', 'id' => '007', 'gpa'
=> '4.1');
echo $SecretAgent;
echo $Secretagent;
$size = sizeof($SecretAgent);
echo "<br>size = $size array elements";
echo "<br>$SecretAgent[name] is $SecretAgent[id]";
if (isset($SecretAgent))
    echo "<br>\$SecretAgent exists";
else
    echo "<br>\$SecretAgent does not exist";
unset($SecretAgent);
if (isset($SecretAgent))
    echo "<br>\$SecretAgent exists";
else
    echo "<br>\$SecretAgent does not exist";
?>
</h3>
<!--
*******************
*   7. Objects    *
*******************
-->
<h2>7. Demonstration of objects</h2>
<h3>
Example 7.1: object<br>
<?php
//class definition
class SecretAgent {
var $agent;
function getagent () {
    return "name is " . $this->agent['name']
           . "<br>id is " . $this->agent['id']
           . "<br>gpa is " . $this->agent['gpa'];
}
function putagent ($val1, $val2, $val3){
    $this->agent = array('name' => $val1, 'id' => $val2, 'gpa' =>
$val3);
}
}
//object declarations
$sa1 = new SecretAgent;
$sa2 = new SECRETAgent;
//method calling
$sa1->putagent('James Bond', '007', '4.1');
echo "sa1:<br>" .  $sa1->getagent();
$sa2->putagent('George Costanza','001.5', 1.5);
echo "<br>sa2:<br>" . $sa2->getagent();
?>
</h3>
<!--
***************************
*  8. Control Structures  *
```

```
***************************
-->
<h2>8. Demonstration of control structures</h2>
<h3>
Example 8.1: foreach used to traverse an array
<?php
$SecretAgent = array('name' => 'James Bond', 'id' => '007', 'gpa'
=> '4.1');
foreach ($SecretAgent as $key => $value)
    echo "<br>The secret agent $key is $value";
?>
</h3>
<!--
*******************
*  9. Functions   *
*******************
-->
<h2>9. Demonstration of functions</h2>
<h3>
Example 9.1: user-defined function<br>
<?php
//function definition
function greeting($n, $name) {
  for ($i = 0; $i < $n; $i++)
    echo "Welcome $name! ";
  return "<br><font color=red>*****$name was welcomed $n
times*****</font>";
}
$num = 100;
$firstname = 'Bill';
//function call
$returnvalue = GrEeTiNg($num, $firstname);
echo $returnvalue;
?>
<br>Example 9.2: built-in function<br>
<?php
$str = "PHP is wonderful";
$result = strlen($str) or die("strlen() returned -1");
echo "Length of \"$str\" is $result characters.";
//$string does not exist
$result = strlen($string) or die("<br>strlen(\$string) returned
-1");
echo "Length of \"$string\" is $result characters.";
?>
</h3>
<h2>
<a href="menuphp.htm" target=_top>Menu</a>
```

```
</h2>
</body>
</html>
```

Click the link, syntax.php, in the menu. Return to the menu, by clicking Menu. Then click the link, syntax-withcode.php, in the menu which displays the source code and its output. The series of figures, Figure 9.9 to 9.18, shows various parts of the source code and output of the source code of web page, syntax-withcode.php.



Code

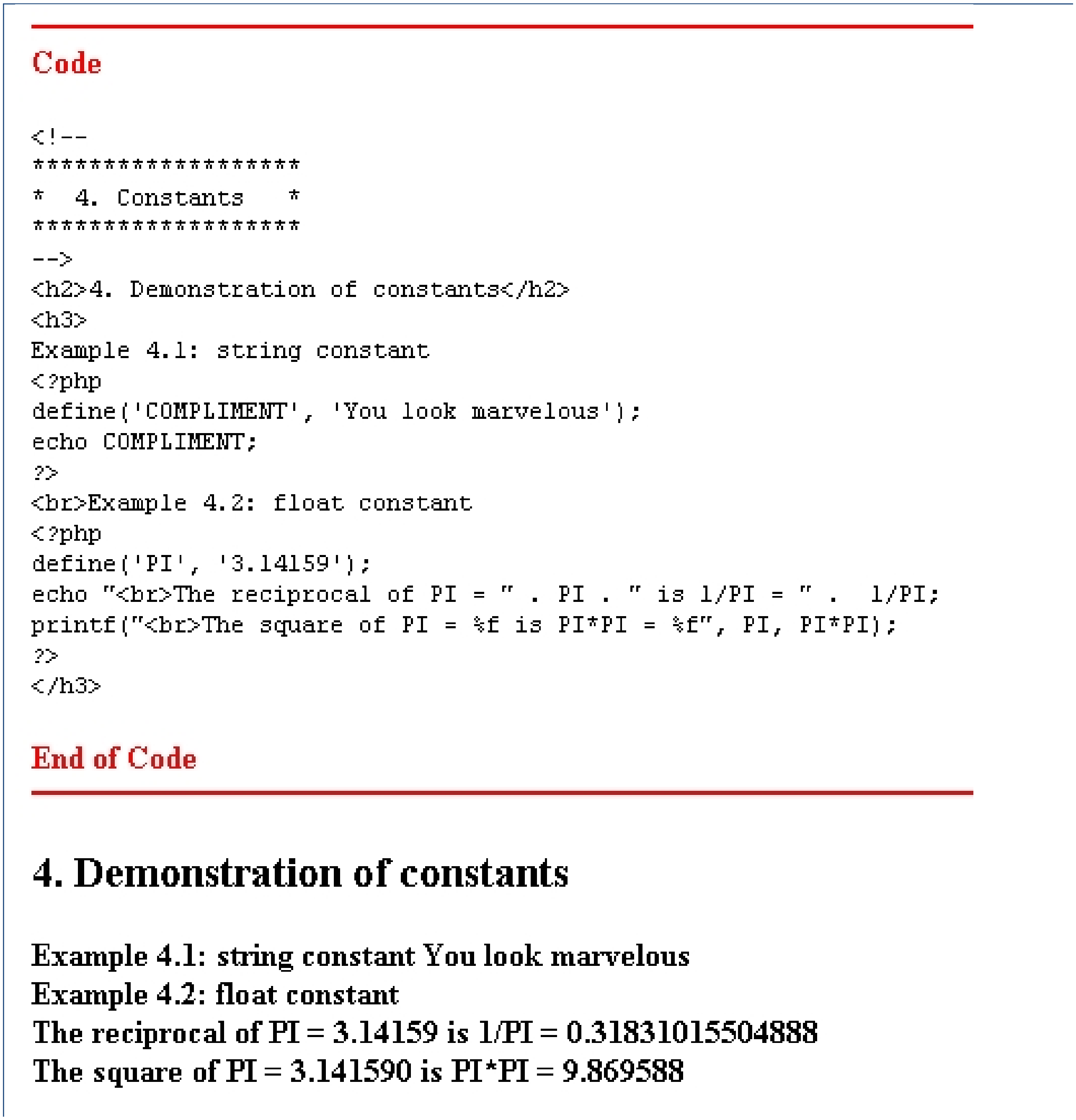
```
<!--
*******************
*  3. Variables   *
*******************
-->
<h2>3. Demonstration of variables</h2>
<h3>
Example 3.1: variable names are case sensitive<br>
<?php
$var = "Bill";
$Var = 2;
$VAR = "cats";
echo "$var has $Var $VAR.";
?>
</h3>
```

End of Code

3. Demonstration of variables

Example 3.1: variable names are case sensitive
Bill has 2 cats.

Figure 9.9 Variables: part of web page, syntax-withcode.php



Code

```
<!--
*******************
*  4. Constants   *
*******************
-->
<h2>4. Demonstration of constants</h2>
<h3>
Example 4.1: string constant
<?php
define('COMPLIMENT', 'You look marvelous');
echo COMPLIMENT;
?>
<br>Example 4.2: float constant
<?php
define('PI', '3.14159');
echo "<br>The reciprocal of PI = " . PI . " is 1/PI = " .  1/PI;
printf("<br>The square of PI = %f is PI*PI = %f", PI, PI*PI);
?>
</h3>
```

End of Code

4. Demonstration of constants

Example 4.1: string constant You look marvelous
Example 4.2: float constant
The reciprocal of PI = 3.14159 is 1/PI = 0.318310155048888
The square of PI = 3.141590 is PI*PI = 9.869588

Figure 9.10 Constants: part of web page, syntax-withcode.php

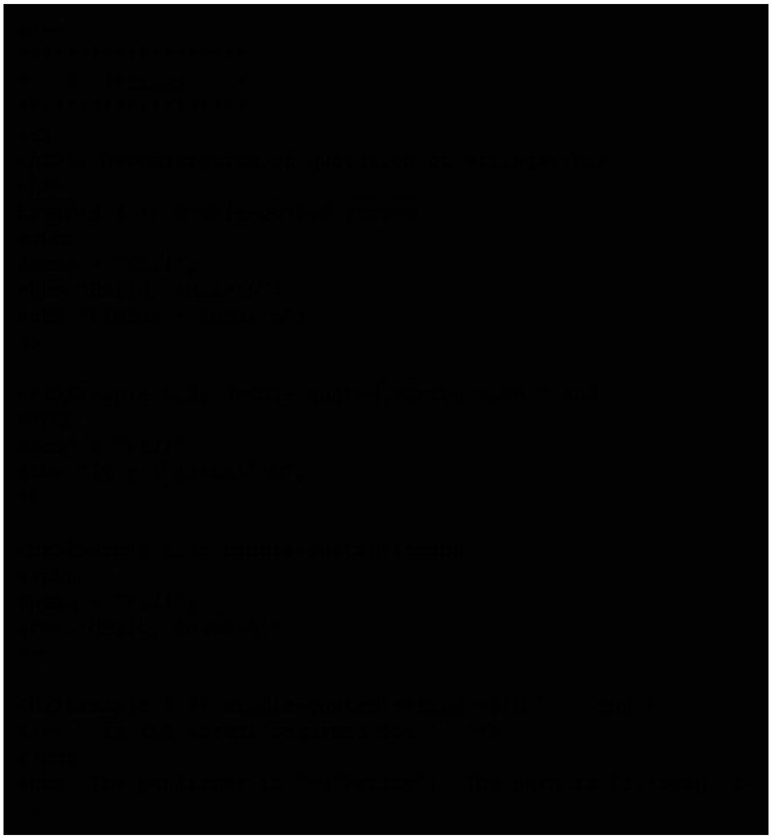


Figure 9.11 Strings: first part of web page, syntax-withcode.php

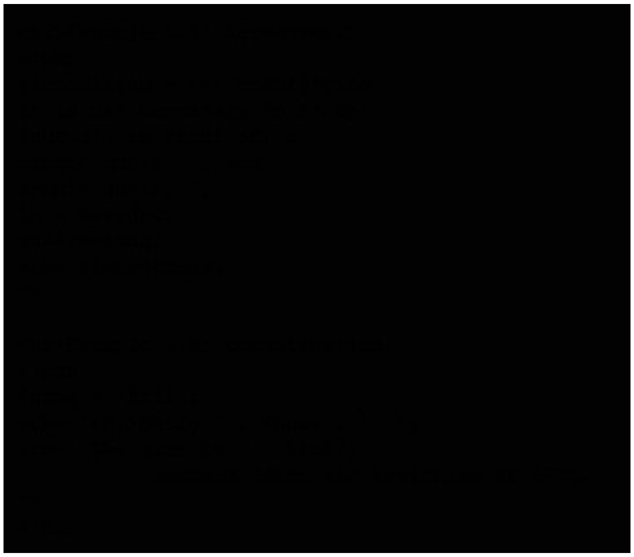
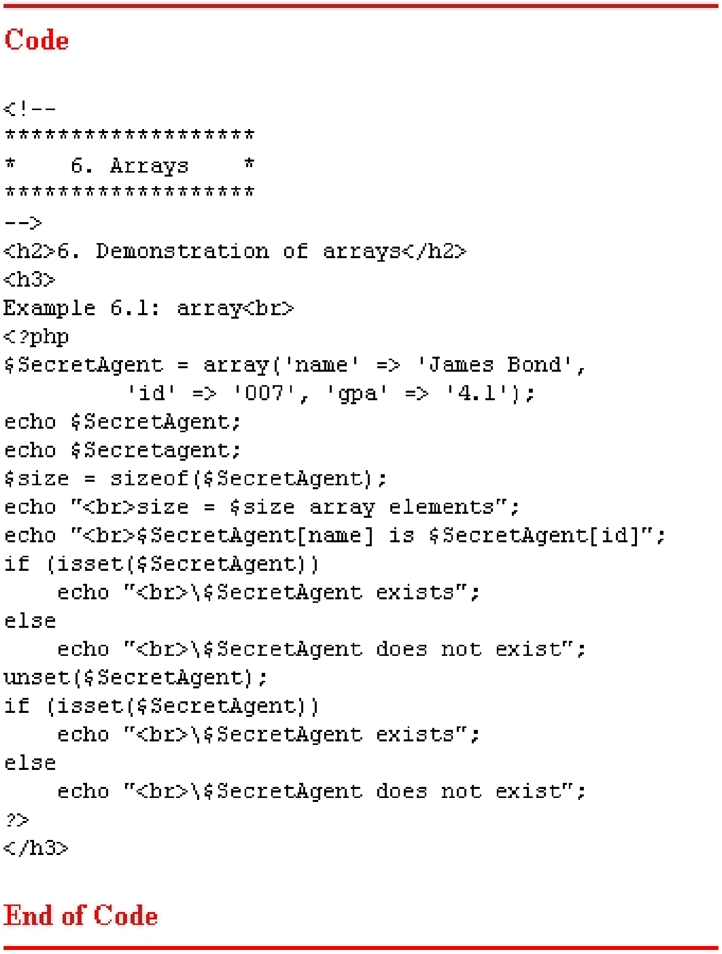


Figure 9.12 Strings: second part of web page, syntax-withcode.php

[redacted]

Figure 9.13 Strings: third part of web page, syntax-withcode.php



Code

```
<!--
*******************
*    6. Arrays    *
*******************
-->
<h2>6. Demonstration of arrays</h2>
<h3>
Example 6.1: array<br>
<?php
$SecretAgent = array('name' => 'James Bond',
         'id' => '007', 'gpa' => '4.1');
echo $SecretAgent;
echo $Secretagent;
$size = sizeof($SecretAgent);
echo "<br>size = $size array elements";
echo "<br>$SecretAgent[name] is $SecretAgent[id]";
if (isset($SecretAgent))
    echo "<br>\$SecretAgent exists";
else
    echo "<br>\$SecretAgent does not exist";
unset($SecretAgent);
if (isset($SecretAgent))
    echo "<br>\$SecretAgent exists";
else
    echo "<br>\$SecretAgent does not exist";
?>
</h3>
```

End of Code

Figure 9.14 Arrays: part of web page, syntax-withcode.php

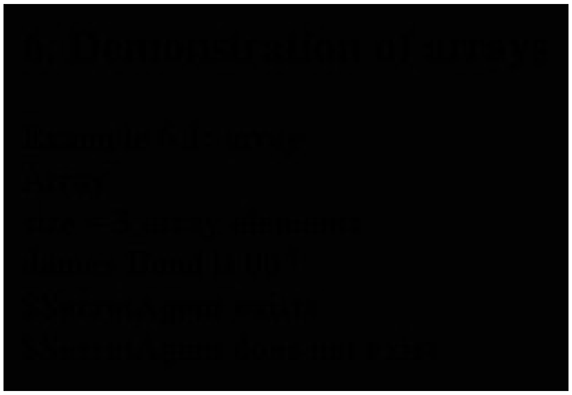


Figure 9.15 Arrays: part of web page, syntax-withcode.php

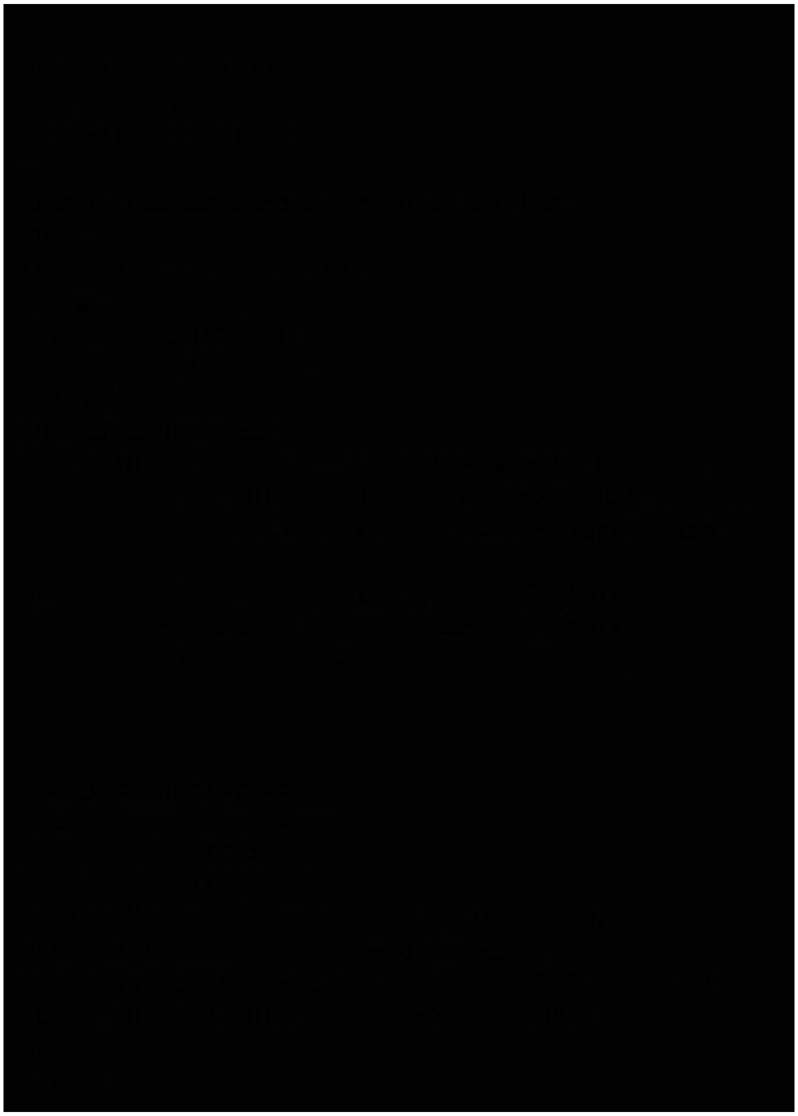
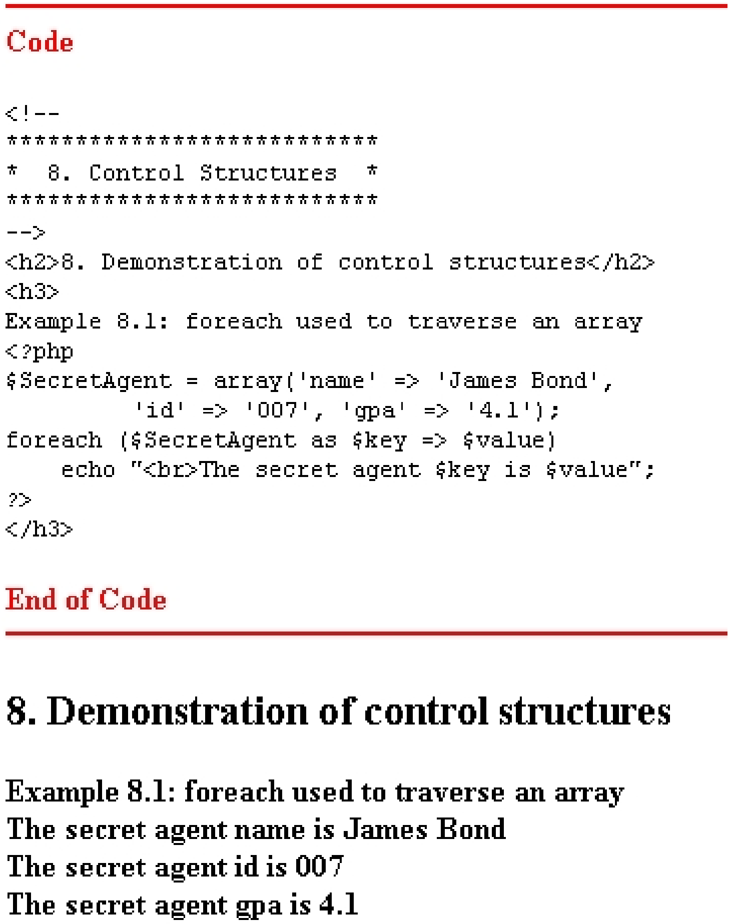


Figure 9.16 Objects: part of web page, syntax-withcode.php

Figure 9.17 Objects: part of web page, syntax-withcode.php



Code

```
<!--
****************************
*  8. Control Structures  *
****************************
-->
<h2>8. Demonstration of control structures</h2>
<h3>
Example 8.1: foreach used to traverse an array
<?php
$SecretAgent = array('name' => 'James Bond',
         'id' => '007', 'gpa' => '4.1');
foreach ($SecretAgent as $key => $value)
    echo "<br>The secret agent $key is $value";
?>
</h3>
```

End of Code

```
8. Demonstration of control structures

Example 8.1: foreach used to traverse an array
The secret agent name is James Bond
The secret agent id is 007
The secret agent gpa is 4.1
```

Figure 9.18 Control structures: part of web page, syntax-withcode.php

9.8.5 HTML form and PHP script in separate files

The HTML form is sometimes called the front end and the PHP script is sometimes called the back end.

The front end refers to the browser; the form is displayed by the browser. The back end refers to the web server; the PHP script is executed by the web server. The software resides on two computers, client and server, which are connected by the internet. The browser runs on the client computer and the web server runs on the server computer.

The user by means of the browser requests the web page with the form. The browser displays the form on the screen. The form receives information from the user, who uses the keyboard to type data into text boxes and the mouse to click radio buttons, for example. When all information is entered, the user clicks the submit button. The name of the file (web page) requested by the browser is the value of the action attribute in the form tag. The browser sends both the information in the form and the request for the file to the web server. The web server stores the information from the form in an array. The requested file is loaded into main memory from the hard disk of the server computer. Since the file contains a PHP script the web server sends it to the PHP interpreter. The script is executed by the interpreter and in the case that follows accesses the information in the array, processes the information and then sends the output back to the web server. Then the web server sends the web page to the browser which displays the web page on the screen.

The form is the input (or interface) to a four-function calculator. On the form two numbers are typed in two text elements and then one of the four radio buttons, add, subtract, multiply or divide, is clicked. When the submit button is clicked, the data is sent to the PHP script which validates the data. If there is no data, the case when the form is requested for the first time, the script displays the form. If the data is incomplete or invalid, the script displays an error message. If the data is valid, the script does the calculation and outputs the result.

Two files are used; one contains the HTML form and the other contains the PHP script. The web page, cal.htm, contains only the form. The value of the action attribute, cal.php, is the other web page which contains the PHP script. The web page, cal.php, is requested when submit is clicked on the web page, cal.htm.

9.8.5.1 HTML form

This web page contains only HTML instructions. The form starts with the tag, <form . . . >. The value of the attribute, action, in this tag is the web page to which the browser will send the form data when the submit button, Calculate, is clicked.

Web Page 9.5.1 cal.htm

Using a text editor, type the following file, or copy and paste it. Save it in directory, c09php, using the name, cal.htm, and then change the permissions of the file to 604. This file contains only HTML instructions. The form consists of two text boxes, one group of four radio buttons, a submit button and a reset button. The form is the interface between the user and the calculator, the calculator being the other web page, cal.php.

```
<html>
```

```
     <head>
          <title> HTML form and PHP script in separate files
          </title>
     </head>
<body>
     <h1>HTML form and PHP script in separate files</h1>
     <h2>Four-Function Calculator</h2>
     <form method="post" action="cal.php">
     x = <input type="text" name="x" size=10>
     y = <input type="text" name="y" size=10><p>
     <input type="radio" name="calc" value="add"> x + y
<b>add</b><br>
     <input type="radio" name="calc" value="subtract"> x - y
          <b>subtract</b><br>
     <input type="radio" name="calc" value="multiply"> x * y
          <b>multiply</b><br>
     <input type="radio" name="calc" value="divide"> x / y
          <b>divide</b><p>
     <input type="submit" name="submit" value="Calculate">
     <input type="reset" name="reset" value="Clear">
     </form>
     <h2>
     <a href="menuphp.htm" target=_top>Menu</a>
     </h2>
</body>
</html>
```

9.8.5.2 PHP script

The web page with the PHP script, cal.php, is requested by the web page with the HTML form, cal.htm, when the submit button, Calculate, is clicked.

The PHP script validates the data and, if it is correct, makes the calculation and then generates the HTML instructions to display the result. If there is no data, the PHP instructions generate HTML instructions for the form which will be displayed by the browser. If the data is incomplete or invalid, the PHP instructions produce HTML instructions to display an error message. Then the server sends the web page, cal.php, to the browser which will display the HTML output: either the result or the form or an error message.

Web Page 9.5.2 cal.php

Using a text editor, type the following file, or copy and paste it. Save it in directory, c09php, using the name, cal.php, and then change the permissions of the file to 604. This file contains mostly PHP instructions which are embedded in HTML instructions. The PHP script first validates the data by making three tests: all data submitted or not, data is numerical or not, divide by zero or not. If the data is not valid, an error message is displayed. If the data is valid, the result is calculated and then displayed.

The link, Next Calculation, requests the web page, cal.htm, which displays the HTML form. After this link is clicked, another calculation can be made.

Note that since both files, cal.htm and cal.php, are stored in the same directory on the server, each one refers to the other by only the name of the file.

```
<html>
     <head>
          <title> Calculation Script </title>
     </head>
<body>
     <h1>HTML form and PHP script in separate files</h1>
     <h2>
     <?php
     //validate data
     if (($_POST[x] == "") || ($_POST[y] == "")|| ($_POST[calc]
== ""))
     {
            echo "<font color=red>The form is
incomplete.</font><br>\n";
     }
     elseif ((!is_numeric($_POST[x])) || (!
is_numeric($_POST[y])))
     {
            echo "<font color=red>Both x and y must be
                    numbers.</font><br>\n";
     }
     elseif (($_POST[y] == 0) && ($_POST[calc]=="divide"))
     {
            echo "<font color=red>Division by 0 is not
                    allowed.</font><br>\n";
     }
     else
     //calculate result
     {
          echo "The result is:<br>";
          if ($_POST[calc]=="add"){
               $result = $_POST[x] + $_POST[y];
               $op = "+";
          }
          else if ($_POST[calc]=="subtract"){
               $result = $_POST[x] - $_POST[y];
               $op = "-";
          }
          else if ($_POST[calc]=="multiply"){
               $result = $_POST[x] * $_POST[y];
               $op = "*";
          }
          else if ($_POST[calc]=="divide"){
               $result = $_POST[x] / $_POST[y];
               $op = "/";
          }
          echo "x $op y<br>= $_POST[x] $op $_POST[y]<br>=
$result";
     }   //end of else
     ?>
     <p><a href="cal.htm"> Next Calculation </a>
     </h2>
</body>
</html>
```

Click the link, cal.htm, cal.php, in the menu.
Make several calculations using all four functions, with positive and negative numbers. Also, test the data validation done by the script by entering incomplete data, entering nonnumeric data and dividing by zero.

9.8.6 HTML form and PHP script in one file

The program is the same as in the last subsection: a four-function calculator. The two files of the last subsection are combined into one file in this subsection. The details of the functioning of the browser in the client computer and the web server in the server computer are the same as for the previous demonstration, that is, when the HTML form and PHP script are in separate files.

Both the HTML form and PHP script can be one file by using a self-processing (also called self-referencing) web page. The value of the action attribute in the form is the same file as the file in which the form is contained, that is, if the name of the file with the form is webpage1.php, then in the form tag, action=webpage1.php. But it is not necessary to explicitly use the name of the file in the form tag. It is better to use the server array variable, $_SERVER['PHP_SELF'], whose value is the name of the file. Then there is no need to know the name of the file, which may not be chosen when the file is first typed or which may be changed later.

After the data is entered in the form and the submit button is clicked, the same page is requested. The PHP instructions in the web page will generate HTML instructions that will either display the form, preceded by an error message in some cases, or make the calculation and display the result. Then the server will send the web page to the browser which will execute the HTML instructions and thereby either display the form with or without an error message, or the result.

The form is a string with name, $form, with the value of the action attribute, cal_onefile.php. Also in the web page there is a copy of same form with name, $form_same. This form demonstrates the use of the variable, $_SERVER['PHP_SELF'], as the value of the action attribute. The form is displayed in the script by the command, echo "$form", and so the second form can be used by changing its name from $form_same to $form and changing the name of the first form from $form to, for example, $form1.

Description of PHP script in web page

(1) Outline

There are three parts to the script.

String - entire form
Function - calculate and display result
Program - validate data, display form, call function if data is valid

(2) Location

All of the script is in the body of the web page. The string and form could be in the head.

(3) Action

The value of the action attribute of the form is the same web page as the web page which contains the form. So when the form is submitted, the server sends back the same web page.

(4) Method

The value of the method attribute of the form is POST. This is the method by which the browser sends data to the server program. The data is stored in the global array, $_POST. There are two methods: POST and GET. Generally, the method, POST, should be used to send data from forms.

(5) Elements

There are five elements in the form: two text, one radio, submit and reset.
The names of the two text elements are x and y.
The name of the radio element is calc which is a group of four buttons. The four buttons are distinguished by the attribute, value. The four values are add, subtract, multiply and divide.
The submit element is given the value, Calculate, which appears as a name on the button.
The reset element is given the value, Clear, which appears as a name on the button.

(6) Tests

The program makes four tests.

Test 1: web page requested - form
Test 2: one or more values missing - error message, form
Test 3: x or y is not a number - error message, form
Test 4: divide by zero - error message, form

Web Page 9.6 cal_onefile.php

Using a text editor, type the following file, or copy and paste it. Save it in directory, c09php, using the name, cal_onefile.php, and then change the permissions of the file to 604. This file contains both the HTML form and the PHP script. One string is used to store the entire form, and one function is used to make the calculation and display the result. When the page is requested for the first time, only the form is displayed. When the data is incomplete or the data is not numerical or division by zero occurs, first an error message is displayed and after the error message the form is displayed. When the data is valid, the result is displayed without displaying the form afterward.

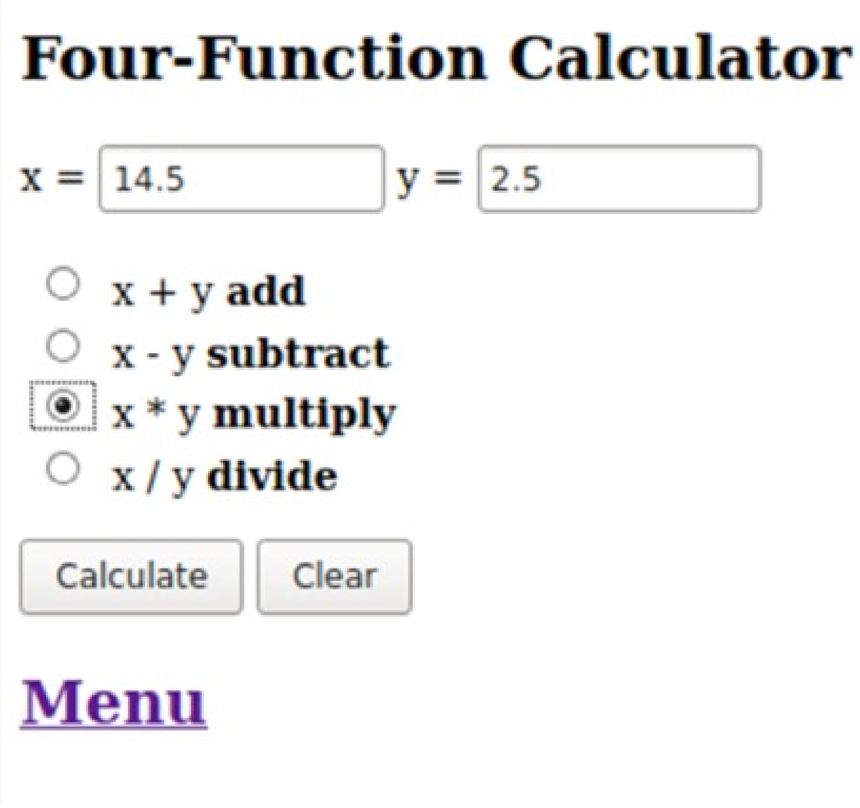
```
<html>
     <head>
          <title> Calculation form and script in one file</title>
     </head>
<body>
```

```
     <h1>HTML form and PHP script in one file</h1>
     <?php
//create the form using a string
     $form = "
          <h2>Four-Function Calculator</h2>
          <form method=post action=cal_onefile.php>
          x = <input type=text name=x size=10>
          y = <input type=text name=y size=10><p>
          <input type=radio name=calc value=add> x + y
<b>add</b><br>
          <input type=radio name=calc value=subtract> x - y
               <b>subtract</b><br>
          <input type=radio name=calc value=multiply> x * y
               <b>multiply</b><br>
          <input type=radio name=calc value=divide> x / y
               <b>divide</b><p>
          <input type=submit name=submit value=Calculate>
          <input type=reset name=reset value=Clear>
          </form>
          ";      //end of string

// same form using variable which stores name of web page
// value of action attribute is $_SERVER[PHP_SELF]
// three strings are concatenated with the operator, .
// to use this form change its name to $form and change the name
// of the form above to another name, $form1, for example
     $form_same = "
          <h2>Four-Function Calculator</h2>
          <form method=post action=" . $_SERVER[PHP_SELF] . ">
          x = <input type=text name=x size=10>
          y = <input type=text name=y size=10><p>
          <input type=radio name=calc value=add> x + y
<b>add</b><br>
          <input type=radio name=calc value=subtract> x - y
               <b>subtract</b><br>
          <input type=radio name=calc value=multiply> x * y
               <b>multiply</b><br>
          <input type=radio name=calc value=divide> x / y
               <b>divide</b><p>
          <input type=submit name=submit value=Calculate>
          <input type=reset name=reset value=Clear>
          </form>
          ";      //end of string

     //make the calculation using a function
     function calc(){
     echo "<h2>";
     if ($_POST[calc]==add){
          $result = $_POST[x] + $_POST[y];
          $op = "+";
     }
     else if ($_POST[calc]==subtract){
          $result = $_POST[x] - $_POST[y];
          $op = "-";
     }
     else if ($_POST[calc]==multiply){
          $result = $_POST[x] * $_POST[y];
          $op = "*";
     }
     else if ($_POST[calc]==divide){
          $result = $_POST[x] / $_POST[y];
          $op = "/";
     }
     echo "The result is:<br>";
     echo "x $op y<br>= $_POST[x] $op $_POST[y]<br>= $result";
     echo "</h2>";
     }       //end of function
     //display the form or the result
     echo "<h2>";
     if (($_POST[x] == "") && ($_POST[y] == "")&& ($_POST[calc]
== ""))
          echo "$form";   //display string
     //validate data
     elseif (($_POST[x] == "") || ($_POST[y] == "")||
              ($_POST[calc] == "")){
          echo "<font color=red>The form is
incomplete.</font><br>\n";
          echo "$form";   //display string
     }
     elseif ((!is_numeric($_POST[x])) || (!
is_numeric($_POST[y]))){
          echo "<font color=red>Both x and y must be
               numbers.</font><br>\n";
          echo "$form";   //display string
     }
     elseif (($_POST[y] == 0) && ($_POST[calc]=="divide")){
          echo "<font color=red>Division by 0 is not
               allowed.</font><br>\n";
          echo "$form";   //display string
     }
     else {
     //calculate result
          calc();     //call function
          echo "<p><h3><a href=\"cal_onefile.php\"> Next
               Calculation </a></h3>";
     }
     echo "</h2>";
     ?>
     <h2>
     <a href="menuphp.htm" target=_top>Menu</a>
     </h2>
</body>
</html>
```

Click the link, cal-onefile.php, in the menu.

Make several calculations using all four functions, with positive and negative numbers. Also, test the data validation which is done by the script by entering incomplete data, entering nonnumeric data and dividing by zero.

Figure 9.19 gives the form displayed to the user. When the user enters the data shown in Figure 9.19, and then clicks, Calculate, the result is shown in Figure 9.20.

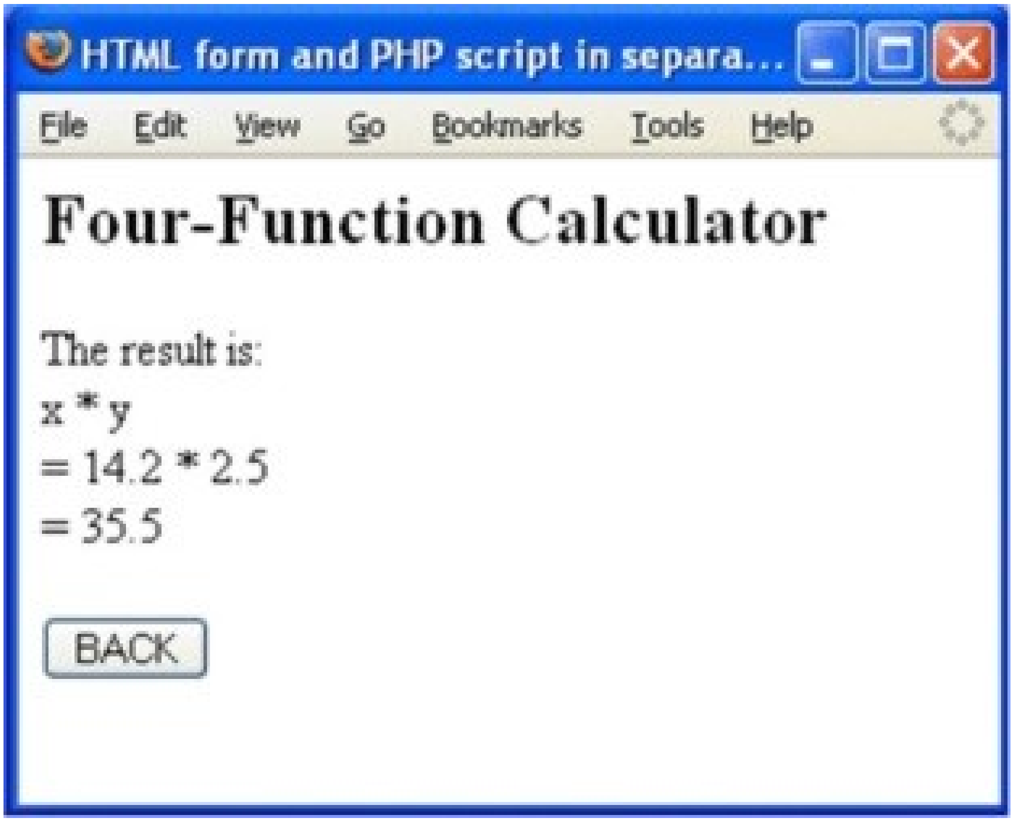


Four-Function Calculator

x = 14.5 y = 2.5

○ x + y add
○ x - y subtract
● x * y multiply
○ x / y divide

Figure 9.19 Four-function calculator interface

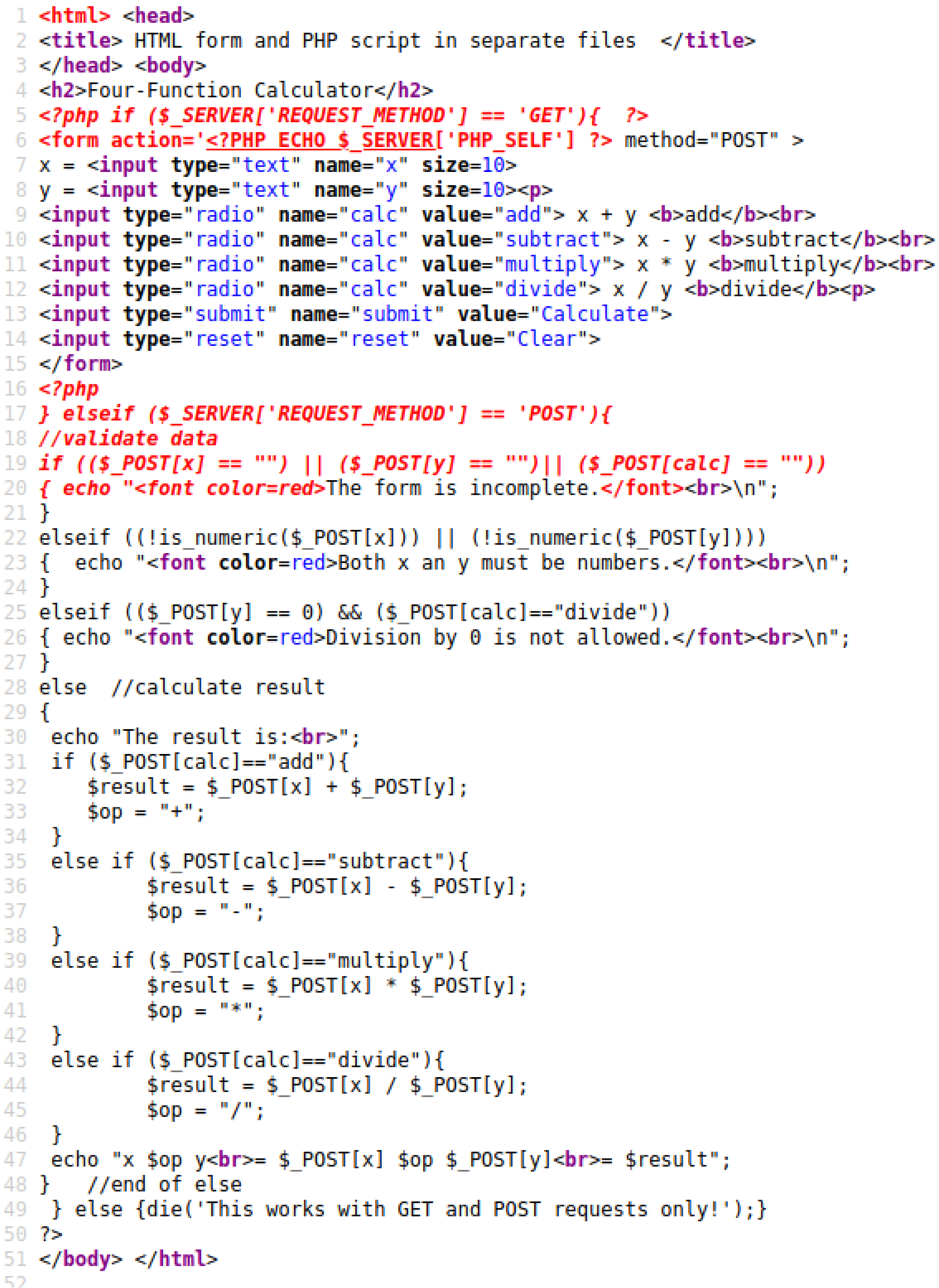


Four-Function Calculator

The result is:
x * y
= 14.2 * 2.5
= 35.5

Figure 9.20 Result display of the calculator

The following is a demonstration of a simple calculator implemented using an HTML form and a PHP script, both in one web page, but implemented differently than the previous web page, cal-onefile.php. The web page for this calculator is given in Figure 9.21. This web page is a variation of Web Page 9.6 which does not use a string to store the form or a function to calculate the result, and also tests for the method. The PHP script is in the same web page and hence the action attribute is given as, $_SERVER['PHP_SELF']. The form is displayed when the web page is first loaded. On pressing the “Calculate” button, the POST method is used to invoke the PHP script for the result. When the “BACK” button is clicked, the calculator form is displayed again for another calculation.



```
<html> <head>
<title> HTML form and PHP script in separate files  </title>
</head> <body>
<h2>Four-Function Calculator</h2>
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET'){   ?>
<form action='<?PHP ECHO $_SERVER['PHP_SELF'] ?> method="POST" >
x = <input type="text" name="x" size=10>
y = <input type="text" name="y" size=10><p>
<input type="radio" name="calc" value="add"> x + y <b>add</b><br>
<input type="radio" name="calc" value="subtract"> x - y <b>subtract</b><br>
<input type="radio" name="calc" value="multiply"> x * y <b>multiply</b><br>
<input type="radio" name="calc" value="divide"> x / y <b>divide</b><p>
<input type="submit" name="submit" value="Calculate">
<input type="reset" name="reset" value="Clear">
</form>
<?php
} elseif ($_SERVER['REQUEST_METHOD'] == 'POST'){
//validate data
if (($_POST[x] == "") || ($_POST[y] == "")|| ($_POST[calc] == ""))
{ echo "<font color=red>The form is incomplete.</font><br>\n";
}
elseif ((!is_numeric($_POST[x])) || (!is_numeric($_POST[y])))
{  echo "<font color=red>Both x an y must be numbers.</font><br>\n";
}
elseif (($_POST[y] == 0) && ($_POST[calc]=="divide"))
{ echo "<font color=red>Division by 0 is not allowed.</font><br>\n";
}
else  //calculate result
{
 echo "The result is:<br>";
 if ($_POST[calc]=="add"){
    $result = $_POST[x] + $_POST[y];
    $op = "+";
 }
 else if ($_POST[calc]=="subtract"){
         $result = $_POST[x] - $_POST[y];
         $op = "-";
 }
 else if ($_POST[calc]=="multiply"){
         $result = $_POST[x] * $_POST[y];
         $op = "*";
 }
 else if ($_POST[calc]=="divide"){
         $result = $_POST[x] / $_POST[y];
         $op = "/";
 }
 echo "x $op y<br>= $_POST[x] $op $_POST[y]<br>= $result";
}   //end of else
 } else {die('This works with GET and POST requests only!');}
?>
</body> </html>
```

Figure 9.21 Source code for the four-function calculator

10 JavaScript

10.1 Introduction

JavaScript is a client-side scripting language. It is embedded in HTML, just like PHP. However, JavaScript is executed on a client computer by a browser, whereas PHP is executed on a server computer by the server program.

One of the main uses of JavaScript is to validate data that the user enters in a form on a web page before the form is submitted to the server. This is the use of JavaScript that is mainly demonstrated in this chapter.

JavaScript is not the same as Java. JavaScript was developed by Netscape and is a scripting language. The program in a scripting language is called a script. A script is interpreted which means that it is executed without first being compiled. Browsers which execute JavaScript include a JavaScript interpreter just as servers which run PHP include a PHP interpreter. All browsers support JavaScript. JavaScript was introduced in 1995 September and first given the name, LiveScript. But at the end of 1995 it was renamed, JavaScript, probably primarily for marketing reasons since Java was well-known by that time.

Java was developed by Sun Microsystems and is a compiled language. A program called a compiler translates a human-generated program (source code) into a form that could be executed on a real machine (machine code). Most languages such as C and C++ need to be compiled on the platform that is used to execute it. On the other hand Java is a platform-independent programming language and hence requires a two-step 'compilation' process. The first step is to convert the human-produced program code into a platform-independent intermediate virtual machine (VM) code. In the second step, this VM code is executed by the VM execution system installed on each real machine.

Browsers are standardized to run Javascript and any changes in the language would have to be handled by a new version of the browser.

10.2 Web Page

A web page which contains both JavaScript and HTML instructions has the extension, .htm or .html. The extension is required so that the browser can recognize the file as a web page.

A JavaScript script is embedded in a web page in either the head section or body section or both. Scripts, as well as all other content, in the head and body sections are loaded (transferred into main memory) when the page is received from the server. After the page is loaded the head section is executed first and then the body section is executed. Both the head and the body can contain both HTML instructions and JavaScript instructions. The instructions are executed in the order that they appear in the page, unless the JavaScript instructions are in a function in which case the instructions are executed when the function is called or an event occurs which calls the function.

In the head section it is common practice to put the definitions of JavaScript functions which are executed in the body section. This ensures that the function definitions exist before the functions are executed, since the head section is processed first.

Tags are used to identify JavaScript instructions in a web page. A script written in JavaScript is the group of instructions between the pair of tags:

```
   <script type="text/javascript"> and  </script>.
```

The structure of a web page that contains JavaScript is the following. One set of HTML instructions and one JavaScript script is shown below in each of the head and body sections, but there can be any number of combinations of HTML instructions and JavaScript scripts in each section. A comment in HTML is enclosed in the pair of tags, <!-- and -->. A single-line comment in JavaScript starts with, //.

```
<html>
<head>
   <!--HTML instructions-->
   <script type="text/javascript">
      //JavaScript instructions
   </script>
</head>
<body>
   <!--HTML instructions-->
   <script type="text/javascript">
      //JavaScript instructions
   </script>
</body>
</html>
```

If the same script is executed on more than one web page, a copy of the script can be put in each web page. Alternatively, one copy of the script can be stored in a file, which has the extension, .js, and then loaded into each web page that executes the script. If the file with the script has the name, script1.js, it is loaded into a web page as follows.

```
   <script src="script1.js"></script>
```

The file, script1.js, contains only JavaScript code without <script> tags or any HTML code. When the browser reads the value of the src attribute, the browser downloads the JavaScript file and the processing of the web page stops until the download is complete. This may cause a noticeable delay in the loading of the web page.

10.3 Three Parts of JavaScript

JavaScript is divided into three parts: core, client-side and server-side.

10.3.1 Core

Core JavaScript is the basic part of the language which makes no reference to objects in web pages. It has all of the components of a modern compiled programming language, C++ for example.

10.3.2 Client-side

Client-side JavaScript is the part of the language that refers to the objects in the programming environment of a browser. The JavaScript in a web page generally includes both core and client-side instructions.

10.3.3 Server-side

Server-side JavaScript is embedded in a web page and executed by the server. However, PHP is now the most popular server-side scripting language and is used in this book. This chapter does not cover server-side JavaScript.

10.4 Syntax of Core JavaScript

Core JavaScript is the basic part of the language which makes no reference to objects in web pages. Client-side JavaScript is the part of the language that is implemented only in web browsers. Client-side JavaScript includes both core JavaScript and components that only refer to a web page, so core JavaScript must be learned first. This section is concerned with core JavaScript. The components of client-side JavaScript are covered in the next section.

The syntax of JavaScript is based on the syntax of Java which is, in turn, based on the syntax of C++. The syntax of C++ is the same as the syntax of C, since C++ is essentially object-oriented C (C with classes). It is assumed that the reader is familiar with the syntax of Java or C++ or C. This section mainly covers differences between the syntax of JavaScript and Java or C or C++. Also references are made to PHP, which can be ignored by readers who are not familiar with PHP.

The subsections are: 1. Comments, 2. Instructions, 3. Output, 4. Variables, 5. Constants, 6. Strings, 7. Arrays, 8. Objects, 9. Control Structures, 10. Functions.

10.4.1 Comments

Comments are characters in a program which are ignored when the program is executed. Comments are used mainly to include documentation in the program, so as to make the program easier to understand when the programmer or someone else is reading the program. Comments are also used to temporarily remove instructions from a program without deleting the instructions during the development of a program.

There are two methods to insert comments into a JavaScript program.

(1) // - single-line comment (C++ method)

The double slash, //, can be inserted anywhere in a line and all characters after the // to the end of the line are a comment. The // applies to only one line.

(2) /* */ - multi-line comment (C method)

The slash-asterisk pair of characters, /*, can be inserted anywhere in a line and the asterisk-slash pair, */, can be inserted anywhere after the /* on the same line or on any line after the same line. All characters between the /* and the */ are a comment. This method applies to one or more lines, in contrast to the first method which applies to only one line.

JavaScript comments and HTML comments are different. The JavaScript comments appear inside the JavaScript tags and are inserted by the methods above. The HTML comments appear in the HTML instructions, outside the JavaScript tags, and are inserted by a method like the C method, except the comments appear inside the pair of tags, <!-- and -->, instead of inside /* and */.

10.4.2 Instructions

Instructions on separate lines can be terminated by a newline (by pressing the key, Enter) or by a semicolon (;). If more than one instruction is on the same line, all the instructions except the last must be terminated by a semicolon. It is good programming practice to terminate all instructions by a semicolon so that the end of an instruction is visible.

10.4.3 Output

There is basically only one way to produce output by JavaScript which is displayed on the screen by the browser: write method of the document object. The syntax is

document.write(*msg*)

where *msg* is a string, for example, *msg* = "hello, world".

A variation of the write method is the writeln method which outputs a string followed by one newline character. However, instead of using the writeln method the write method can be used by adding the new line character, \n, to the end of string.

The output method in JavaScript, write(), corresponds to the function, printf(), in C and the object, cout, in C++. There is no input method in JavaScript that corresponds to input from the keyboard by scanf() in C and by cin in C++. However, there is a dialog box, prompt(), which receives input from the keyboard, which is discussed later.

The function, write(), is actually part of client-side JavaScript since it is a method of the document object which is the web page inside the browser window. But it is included here since demonstrations of core JavaScript cannot be made without using write().

10.4.4 Variables

A variable is a storage location in main memory which holds data.

Name: The name (identifier) of a variable does not need a prefix, like in PHP (where the prefix is $). However, the rules for variable names are the same as in PHP. Variable names can include characters which are letters (a-z,A-Z), digits (0-9) and the underscore(_), except that the first character of the name cannot be a digit. A digit is not allowed as the first character so that JavaScript can easily distinguish a name from a number.

Case: Names of variables are case-sensitive, like in PHP.

Type: Variables are untyped, as in PHP.

Declaration: Variables can be declared or not declared. The keyword, var, is used to declare variables. Always declare variables using the keyword, var. This is called explicit declaration of a variable. If a variable is used which is not declared with var, it is still declared internally in JavaScript; this is called an implicit declaration. However, the difference and the big disadvantage of declaring variables implicitly is that they are created as global variables, whereas, variables which are declared explicitly are local variables. It is important that variables declared inside functions are local variables, so that variables with the same name as variables outside the function (global variables) or with the same name as variables in other functions can be used.

Variables can be declared anywhere in a function like in C++ and not only at the beginning of a function as is mandatory in C. However, unlike in C++ variables in a function are defined throughout the function from the beginning of the function even if they are not declared at the beginning. For this reason it is advisable to declare all variables which are used in a function at the beginning of a function because the function will be easier to understand and it will be easy to avoid the declaration of different variables with the same name.

Examples:

```
var i;                (i declared only)
var j = 0;            (j declared and initialized to integer,
                0)
var j = "zero";        (same variable, j, declared and
                       initialized to string, "zero", after
                       being declared as an integer)
var name = "Bill";     (name declared and initialized to
                       string, "Bill")
var greeting = "Hello " + name;   (greeting declared and
                                  initialized to string, "Hello
                                  Bill")
                       (+ is the concatenation operator)
var fnv = document.form1.firstname.value;  (fnv (short for
                                           first name value) is
                       initialized to the value of the text
                       element, firstname, in the form, form1,
                       in the current HTML document)
```

10.4.5 Constants

A constant is a storage location in main memory which holds data with a value that cannot change.

A constant is declared with the keyword, const, just as in C++. The following are examples of declarations of a constant and a variable.

Examples

```
const pi = 3.14;              //declaration of constant, pi
var i = 0;                    //declaration of variable, i
```

JavaScript has some special numeric values which are defined in the language and are constants. The most important numeric constant is, NaN, which is a not-a-number value. For example, 0/0 and $(-2)^{1/2}$ both give an undefined result which is NaN. The function, isNaN(), tests for the value, NaN.

Another numeric constant is, Infinity, which represents a number that is larger than the largest number that can be stored in a variable. JavaScript uses the 64-bit floating-point format (the double type in Java, C and C++) which can represent numbers approximately as large as about 1.8 x 10^{308}. The function, isFinite(), tests if a number is not NaN and not plus or minus Infinity.

10.4.6 Strings

A string is sequence of 0 or more characters. In C and C++ a string is a sequence of characters that must end with the null character (a byte with all 8 bits equal to zero). In JavaScript a string does not need to end with the null character.

A string is enclosed by either single quotes (′) or double quotes (″), which are equivalent. The double quote character (″) can be contained in a string enclosed by single quotes, and the single quote character (′) can be contained in a string enclosed by double quotes. HTML as well as JavaScript uses both single quotes and double quotes to delimit strings. So when HTML and JavaScript are combined it is good practice to use one type of quote for JavaScript and the other type of quote for HTML, for example, ′ for JavaScript and ″ for HTML.

The backslash (\) is used in strings to escape (change the meaning of) certain characters which follow it, for example, \n, which is a newline and \" which will not terminate a string. A string can be typed on more than one line by typing a backslash (\) at the end of the line.

Strings are concatenated by the operator, +, in contrast to the operator, . (dot), in PHP.

A regular expression is a special kind of string that is used for pattern matching. A regular expression is enclosed by slashes (/).

Examples

```
var str = "hello, world";   //declaration and initialization of
                            string, str
var str = 'hello, world';   //same as with double quotes
var str2 = "The length of \"" + str + "\" = " + str.length +
             " bytes<br>";
document.write(str2);
             //output: The length of "hello, world" = 12 bytes
```

10.4.7 Arrays

An array is a composite data type which is a group of values, called elements, each of which is identified by a number, called an index. The indices of the elements are numbered from 0 to *length*-1, where *length* is the number of elements in the array. Arrays in JavaScript are indexed arrays, the same as the arrays in C and C++. But in JavaScript an element of an array can have any data type and so different elements of the same array can have different data types, unlike in C and C++, since JavaScript is an untyped language.

An array is treated as a distinct data type but actually an array is an object, covered in the next subsection. Since an array is an object it is created by an array constructor, Array(), covered in the next section. However, there is also literal syntax for creating arrays. For example, if a is an array, the declaration of a as a literal is

```
   var a = [1.23, "Bill", true];
```

The array, a, is chosen so that each element of the array has a different data type: 1.23 is a floating-point number, "Bill" is a string and true is a Boolean literal.

An element of an array is accessed using the [ ] operator, the same operator as used to create an array literal. The first element of array, a is a[0] which is 1.23, the second element is a[1] which is "Bill" and the third element is a[2] which is true. For example, the value of the first element can be changed to 3.21 by

```
   a[0] = 3.21;
```

There is no associative array as in PHP. But, in fact objects, which are discussed next, are stored internally in JavaScript as associative arrays.

Examples

```
   var days = new Array("Sunday", " Monday", "Tuesday",
           "Wednesday", "Thursday", "Friday", "Saturday");
           //creation of array by the constructor, Array()
   document.write("Length of the array is " + days.length +
           " elements<br>");
           //Output: Length of the array is 7 elements
   document.write(days[0] + " is a holiday<br>");
           //Output: Sunday is a holiday
   document.write(days[7] + " is not a day<br>");
           //Output: undefined is not a day
```

10.4.8 Objects

An object is a composite data type which is group of properties (variables) and methods (functions), each of which has a name. The values (properties and methods) of an object are associated with a name in contrast to an array where the values of the array are associated with a number, the index of the array.

A class in Java and C++ is an entity that defines the structure of an object. There is no entity, class, in JavaScript. A good discussion of the distinction between objects in JavaScript and in Java and C++ is in Chapter 8 of the Netscape core guide for Javascript 1.5, given in References.
There are a number of built-in objects in core JavaScript. They have global scope, meaning that they are accessible everywhere in a program, and so they are called global objects. The most common of the global objects are: 1. Array, 2. Boolean, 3. Date, 4. Math, 5. Number, 6. RegExp, and 7. String. There are other global objects as well as these.
An object, except for Math, is created by using the keyword, new, together with the object constructor, which is a function with the same name as the object.

10.4.8.1 Array object

An array is discussed in a previous subsection.

There are three ways to use the array constructor to create an array object. Note that the name of the constructor, Array, is case sensitive.

(1) Create an array object which is empty, that is, has no elements.

var *arrayobject* = new Array();

(2) Create an array object which has *length* elements, with indices from 0 to *length*-1, each element of which has the value, undefined.

var *arrayobject* = new Array(*length*);

(3) Create an array object with elements having explicit values given by the argument list, with indices from 0 to *length*-1.

var *arrayobject* = new Array(*e0*, *e1*, . . . , *e*(*length*-1));

An array object of the third form can also be created without a constructor using the syntax (called array literal syntax),

var *arrayobject* = [*e0*, *e1*, . . . , *e*(*length*-1)];

Elements of an array are accessed by using the operator, [ ]. The first element of array, *arrayname*, is *arrayname*[0], the second element is *arrayname*[1] and so on.

The array object has only one property, length, which is the number of elements in the array. The array object has twelve methods, which can be found, for example, in either book by Flanagan, given in References.

10.4.8.2 Boolean object

A boolean variable has only one of two values: true or false. The boolean object is described as a wrapper around a boolean value. This is computer science jargon (for the use of the word wrapper in this context) meaning that the constructor for a boolean object takes any value, converts it to true or false and stores it in the boolean object. Internally, JavaScript stores true as 1 and false as 0.

There is only one form of the constructor for a boolean object:

var *booleanobject* = new Boolean(*value*);

For *value* with values, 0, "" (empty string), null, NaN and undefined, *booleanobject* is false. For all other values, *booleanobject* is true. Note that *value* with value, "false", is also converted to true, since this is a string which is not empty.

10.4.8.3 Date object

The date is stored internally in JavaScript as the number of milliseconds from the beginning of 1970. There are four ways to use the date constructor to create a date object. The most common way is: var *datename* = new Date();
which creates a date object, *datename*, set to the current date and time.

10.4.8.4 Math object

Math is a library of functions and constants which are accessed by the syntax, Math.*function*() and Math.*constant*. There is no constructor function for Math.

10.4.8.5 Number object

A number is a basic data type in JavaScript. There is no distinction between integers (whole numbers) and floating-point numbers (fractional numbers) as in C, C++ and Java. All numbers are stored in JavaScript in the 64-bit floating-point format, which is type double in Java, C and C++.

There is one form of the constructor for a number object:

var *numberobject* = new Number(*value*);

where *value* is the value of a number to be converted to an object. It is rarely necessary to create a number object explicitly with the constructor because JavaScript automatically converts between the number value and number object as needed.

There are five numeric constants which are properties of the Number() constructor and not properties of individual number objects: largest and smallest numbers that can be stored, positive and negative infinity, and not-a-number value. An example of a not-a-number value is the result of taking the square root of a negative number. There are five methods, all of which convert a number to a string. The constants and methods can be found, for example, in either book by Flanagan, given in References.

10.4.8.6 RegExp object

A regular expression is a pattern of characters used to match character sequences in strings. There is one form of the constructor for a RegExp object:

var *reobject* = new RegExp(*pattern*, [*attributes*]);

where *pattern* is a string which is the pattern of characters and *attributes* is an optional string which can specify global, case-insensitive and multiline matches.

A regular expression can also be created with the literal syntax:

var *reobject* = /*pattern*/*attributes*;

The most common RegExp method is *reobject*.test(*string*) which returns true if *string* contains characters which match the regular expression, *reobject*, or false otherwise.

One internet reference which tabulates all of the special characters used in *pattern* is given by a web site in References.

Example: regular expression, re, that matches any string that ends with s

```
var re = new RegExp("s$");    //using constructor
var re = /s$/;                //using literal syntax
```

re matches the string, cats, but not the strings, cat or CATS.

10.4.8.7 String object

A string is discussed in a previous subsection.
There is one form of the constructor for a String object: var *strobject* = new String(*str*);
where *str* is a value of a string or represents a string.
The string object has only one property, length, which is the number of characters in the string.

The string object has sixteen methods, which can be found, for example, in either book by Flanagan, given in References.

10.4.9 Control Structures

Control structures determine the order in which instructions in a program are executed. The control structures in JavaScript are the same as in C and C++ with the addition of one more control structure: for/in.

The syntax of for/in is:

```
for (variable in object)
    statement
```

where *statement* is one or more instructions.

The for/in control structure is used to execute the instructions, *statement*, once for each property, *variable*, of an object, *object*. In contrast, the control structure, for, does the same thing for elements of an array.

10.4.10 Functions

A function is a group of instructions which have a name and can be executed by using the name of the function. Functions are defined and called (to call a function means to start the execution of the function) in the same way as in C and C++.

The names (identifiers) of functions and variables follow the same rules. Function names are case-sensitive and can include characters which are letters (a-z,A-Z), digits (0-9) and the underscore (_), except that the first character of the name cannot be a digit. A digit is not allowed as the first character so that JavaScript can easily distinguish a name from a number.

10.5 Client-side JavaScript

Client-side JavaScript is JavaScript used in an HTML document, commonly called a web page. Client-side JavaScript includes core JavaScript and additional objects that represent the contents of a web page.

The JavaScript in a web page is executed by the browser. The basic purpose of the browser is to display a web page in a window and JavaScript in a web page can make the display dynamic, that is, each time the browser displays the page, the display can be different.

The area on the screen in which the browser displays the web page is represented by a window object. The window object is the global object, meaning that all other objects of client-side JavaScript are part of the window object. The web page inside the window is represented by a document object. Inside the document there are forms, links and images and each of these are represented by objects. Finally, inside a form there are eleven elements (either text boxes or buttons) each of which is represented by an object. This is a hierarchy of objects with the window object as the root object and all other objects descending from the window object.

The hierarchy (ranking) of objects in client-side JavaScript is given in Figure 10.1 in the book by Flanagan, "JavaScript The Definitive Guide", given in References. Figure 10.1 below is a partial object hierarchy which shows the most common objects.

All of the objects which descend from the document object in Figure 10.1 are referred to as the document object model (DOM). The objects below Document in Figure 10.1 are supported by all browsers and is part of what is called the Level 0 DOM. The World Wide Web Consortium (W3C) has produced two standards of the DOM known as Level 1 and Level 2 and recent browsers implement most of these standards.

The type of organization in Figure 10.1 is called object-oriented programming (OOP). In OOP each object has properties (data) and methods (functions) associated with the object. All the properties and methods of an object, Object1, also belong to (are inherited by) all objects which descend from Object1.

The syntax to refer to properties or methods of an object is,

oName1.oName2. ... .oNameN.propertyOrMethodName

where *oName2* is an object which descends from object, *oName1*, and *propertyOrMethodName* is a property or method of object, *oNameN*.

For example, the document object is a property of the window object and the write() is a method of the document object. The syntax to call the write() method is:

window.document.write(*str*);

where *str* is a string variable. Since window is a global object, it is not necessary to use the prefix, window, when referring to properties of the window object. So, the call to write() can be written: document.write(*str*);

10.6 Dialog Boxes

The three most commonly used methods of the window object are methods which display dialog boxes: alert(), confirm(), prompt(). When calling these methods it is not necessary to use the prefix, window, since window is a global object. The last method, prompt(), is the means by which a user can input data to a script, which corresponds to the function, scanf(), in C and the object, cin, in C++.

A dialog box is a popup window which gives a message to the user. All three dialog boxes block which means that the execution of the script stops until the user acknowledges the dialog box. A brief description of the methods which display dialog boxes follows.

10.6.1 Alert()

Syntax: alert(*msg*)
Return value: no return
Description: display a message to the user, given by the string variable, *msg*.

10.6.2 Confirm()

Syntax: confirm(*qst*)

Return value: true if user clicks OK, false if user clicks Cancel

Description: ask the user a question, given by the string variable, *qst*, which the user answers by clicking OK or Cancel.

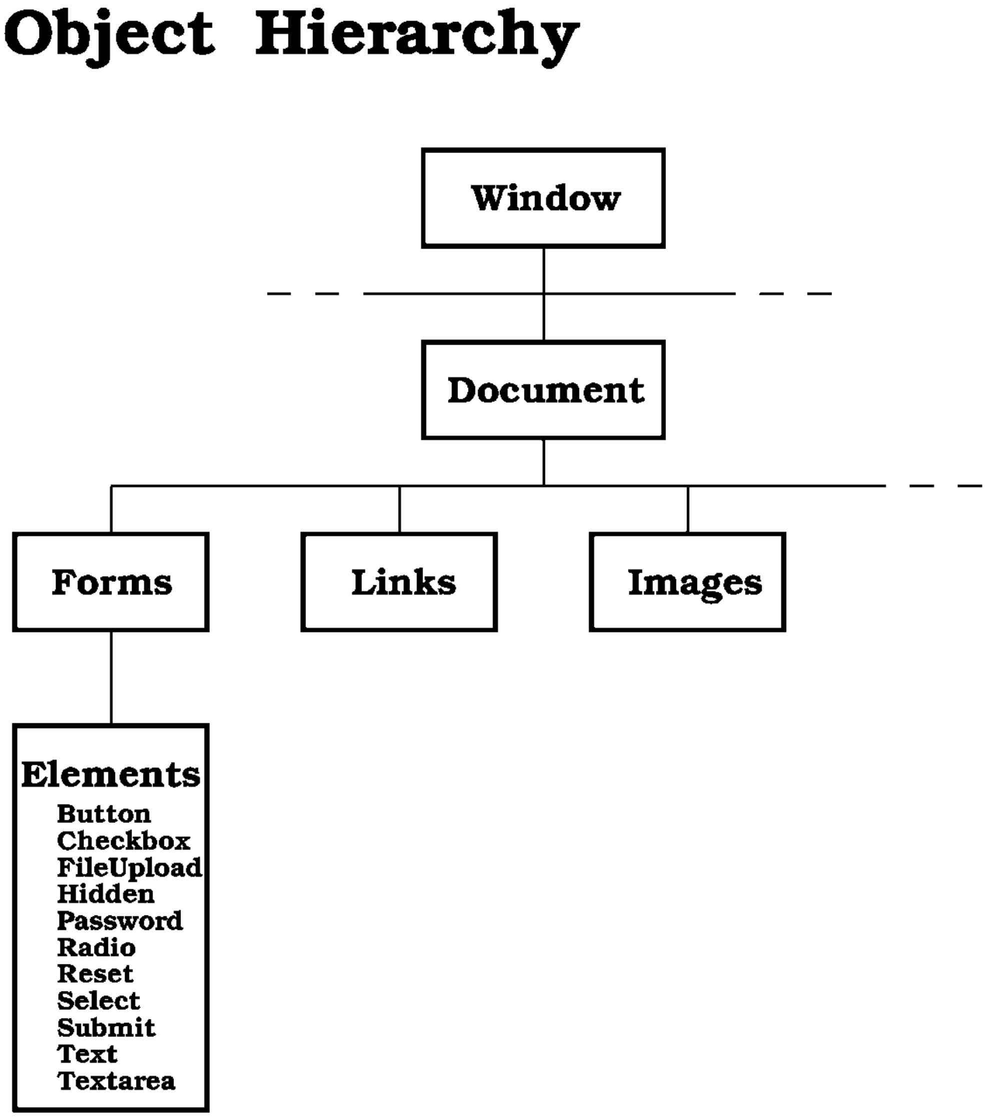


Figure 10.1 Partial client-side object hierarchy

10.6.3 Prompt()

Syntax: prompt(*msg*, *default*)

Return value: string typed by user after user clicks OK, null if user clicks Cancel

Description: display a message to the user, given by the string variable, *msg*, which asks the user to type a line of text; the string variable, *default*, is the initial value in the text input box, which is the empty string, "", for an initial empty input box.

10.7 Elements

An element is an object in a form. An element is also called a control. Most elements are used to receive input from the user. Elements are at the bottom of the hierarchy of the objects in client-side JavaScript. As shown in Figure 10.1, at the top of the hierarchy is the window object; the window object contains a document object; the document object contains a form object; the form object contains element objects. For example, if a form has the name, studentform, and a text element has the name, lastname, reference is made to the value property of the text element by the syntax,

```
      document.studentform.lastname.value
```

It is unnecessary to include, window, at the beginning (window.document . . .) because window is a global object.

A common technique to save typing is to create a variable for the name of the form, using a short variable name, f in this case,

```
      var f = document.studentform;
```

and then refer to the value of lastname (and any other element of the form) by

```
      f.lastname.value
```

There are eleven elements which can be on a HTML form, which are listed in Table 7.3 of Chapter 7 and shown in Figure 10.1. The elements are divided into two categories, as described in Chapter 7: text and buttons. Table 10.1 lists the categories, the eleven elements, as well as the events which are demonstrated for each element in this chapter. Events are covered in the next section.

Elements are objects and like all objects have properties and methods associated with them.

10.7.1 Properties of elements

There are several properties of elements. The most important properties are the following:

type: read-only string that identifies the element, given in the second column of Table 7.3

name: read-only string that is used for two purposes - to refer to the element in JavaScript using syntax, *form.name*, and to send to the server when the form is submitted in the format, *name=value*

value: read/write string that is sent to the server when the form is submitted in the format, *name=value*

10.7.2 Methods of elements

There are several methods of elements. Some important methods are listed below.
The user puts data in a form by using the keyboard or mouse. Only one element at a time can receive input from the user. The element which currently is accessible to the user by the keyboard is said to have the focus. The user can move the focus by the keyboard or mouse. The keyboard key, Tab, moves the focus from one element to another. The mouse can be used to move the focus to an element by clicking on the element.

Two methods in a program can be used to move the focus: blur() and focus(). These methods do not cause the onblur and onfocus functions to be executed, which are discussed later.
blur() - remove focus from an element using syntax

element.blur()

focus() - give focus to an element using syntax
element.focus()

Text in an element can be selected by the keyboard using cursor movement keys while depressing the Shift key or by the mouse by dragging the mouse across the text while depressing the left mouse button. All text can also be selected by using the method, select(), in a program.
select() - select all of the text in a text, textarea, password or fileupload element using syntax,
element.select()

An element can be clicked by the mouse. This can be simulated by the method, click(). But this is not a useful method because it does not cause the onclick function to be executed.

10.8 Events

An event is something that happens at a particular time and place. JavaScript can respond to events that the user causes in a web page at a time that the user chooses. The events are initiated by the user in two ways: by (1) typing on the keyboard or (2) clicking or moving the mouse.

There are about twenty events which are caused either directly or indirectly by using the keyboard or mouse. The most common keyboard event is keypress which occurs when a key is pressed and then released. The most common mouse event is click which occurs when the left mouse button is pressed and then released.

10.8.1 Event attribute

For each event there is an HTML attribute which associates the event with an element. The name of the attribute is the name of the event with the prefix, on. So, for example, the event attribute for event, keypress, is onkeypress and the event attribute for event, click, is onclick.

10.8.2 Event handler

The instructions which are executed when an event occurs is called an event handler. The instructions of the event handler are usually put in a function which is assigned to the event attribute, but the instructions can also be assigned directly to the event attribute.

So in JavaScript, a function can be executed in two different ways. The function can be called by an instruction which consists of the name of the function, just as functions are called in C or C++. Also a function can be executed when an event occurs (mouse click of a button, for example) if the event handler (a function) is assigned to the event attribute (onclick) for the element (button). The syntax is: *eventAttributeName=eventHandlerName*

For example, for the submit button, the HTML instruction which creates the button will enclose the event attribute and handler as follows

```
<input type="submit"value="Send" onclick="confirmSubmit()">
```

where the event handler, confirmSubmit(), is a function assigned to the event attribute, onclick. The function can do whatever the programmer desires, for example, display a dialog box informing the user that the form was sent to the server and asking the user not to click the button again. The function is executed immediately after the event occurs.

10.8.3 Events demonstrated

The following events are demonstrated in this chapter. The event attribute corresponding to the event is given with a brief description of the event.

onblur - element loses keyboard focus
onchange - value of element changes and then loses keyboard focus
onclick - mouse is clicked (pressed and then released) on an element
ondblclick - mouse is double-clicked (quickly pressed and then released, two times) on an element
onfocus - element receives keyboard focus
onkeypress - key is pressed and then released in an element

10.9 Demonstrations

There are eight different demonstrations which are web pages with both HTML instructions and JavaScript instructions. Also there is one web page, called a menu, with links to the eight demonstrations.

The web pages must be stored on a server computer and displayed on a client computer with a browser. It is assumed that programmer remotely logs in to the server. The instructions which follow are for the case when the operating system on the server is Linux. In the document home directory on the server, create the directory, c10js, and then set the permissions of the directory to 707 (write permission is required because the interactive web page writes files on the server). After, change the working directory to c10js. Do this using the following commands.

```
mkdir  c10js             (create directory,  c10js)
chmod 707  c10js         (change permissions of directory,  c10js)
cd  c10js                (change working directory to  c10js)
```

Table 10.1 lists all the elements and the event attributes which are demonstrated for each element in this chapter.

10.9.1 Menu

The web page, menujs.htm, is a menu which gives the links to the web pages which are the demonstration programs in this chapter. All demonstration web pages are in the same directory as menujs.htm.

If the web page is to be typed, start a text editor, emacs in Linux for example. The command to start emacs and open a file with name, menujs.htm, is:

```
emacs menujs.htm         (open file, menujs.htm, in emacs)
```

This is the name of the menu web page in the first subsection below.

After typing the file, save (keystrokes: Ctrl+xs) the file and exit (keystrokes: Ctrl+xc) emacs.
Set the permissions of the web page to 604. Display the permissions of the web page before and after.

```
ls -l m*                 (display permissions of all files starting with letter, m)
chmod 604 menujs.htm     (change permissions of file, menujs.htm)
ls -l m*                 (display permissions of all files starting with letter, m)
```

Table 10.1 Elements and Event Attributes

Category	Element Name	Event Attributes which are demonstrated
text	1. text	onchange, onkeypress
	2. password	onchange, onkeypress
	3. hidden	(not demonstrated)
	4. text area	ondblclick, onfocus, onblur
	5. select	onchange
	6. file upload	(not demonstrated)
button	7. radio	onclick
	8. checkbox	onclick
	9. submit	onclick
	10. reset	onclick
	11. button	onclick

Web page 10.1 menujs.htm

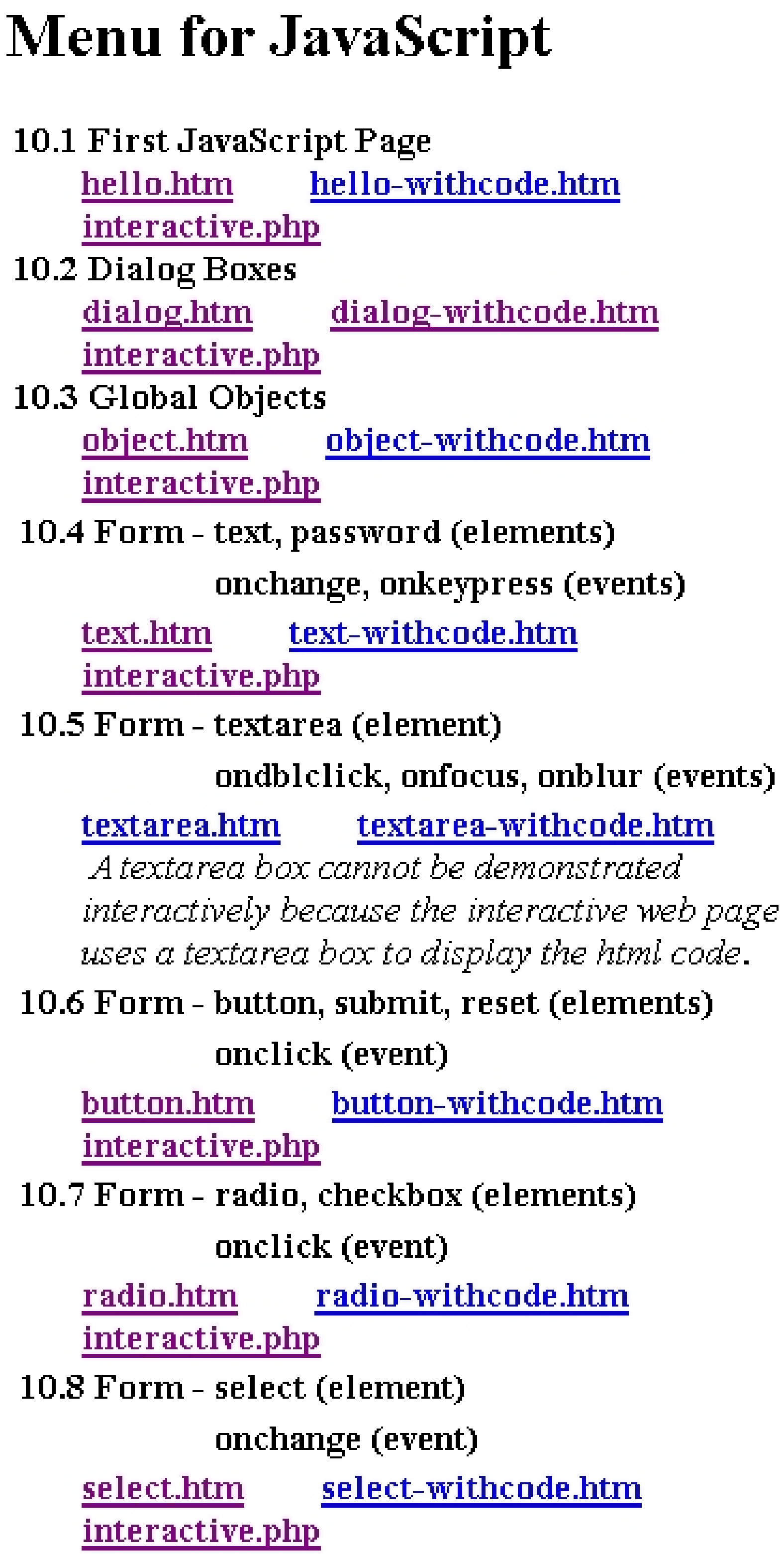
```
<html>
<head>
<title>Menu for JavaScript</title>
</head>
<body>
<h1>Menu for JavaScript</h1>
<table width=100% border=0><tr>
<td width=45%>
<h3> <!-- size of h3 is 14 points -->
10.1 First JavaScript Page
<br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="hello.htm">hello.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="hello-withcode.htm">hello-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="interactive.php?fn=hello.htm">interactive.php</a><br />
10.2 Dialog Boxes
<br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="dialog.htm">dialog.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="dialog-withcode.htm">dialog-withcode.htm</a><br>
```

```
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="interactive.php?fn=dialog.htm">interactive.php</a><br />
10.3 Global Objects
<br>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="object.htm">object.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="object-withcode.htm">object-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="interactive.php?fn=object.htm">interactive.php</a><br />
<table><tr><td><font point-size=14><b>
10.4 Form - </b></font></td><td><font point-size=14><b>text,
password (elements)</b></font></td></tr>
<tr><td>&nbsp;</td><td><font point-size=14><b>onchange,
onkeypress (events)</b></font></td></tr></table>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="text.htm">text.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="text-withcode.htm">text-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="interactive.php?fn=text.htm">interactive.php</a><br />
<table><tr><td><font point-size=14><b>
10.5 Form - </b></font></td><td><font point-size=14><b>textarea
(element)</b></font></td></tr>
<tr><td>&nbsp;</td><td><font point-size=14><b>ondblclick,
onfocus, onblur (events)</b></font></td></tr></table>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="textarea.htm">textarea.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="textarea-withcode.htm">textarea-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<span
style="font-style:italic;font-weight:normal">
A textarea box cannot be demonstrated<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;interactively
because the interactive web page<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;uses a textarea
box to display the html code.</span>
<br /><table><tr><td><font point-size=14><b>
10.6 Form - </b></font></td><td><font point-size=14><b>button,
submit, reset (elements)</b></font></td></tr>
<tr><td>&nbsp;</td><td><font point-size=14><b>onclick
(event)</b></font></td></tr></table>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="button.htm">button.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="button-withcode.htm">button-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="interactive.php?fn=button.htm">interactive.php</a><br />
```

```
<table><tr><td><font point-size=14><b>
10.7 Form - </b></font></td><td><font point-size=14><b>radio,
checkbox (elements)</b></font></td></tr>
<tr><td>&nbsp;</td><td><font point-size=14><b>onclick
(event)</b></font></td></tr></table>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="radio.htm">radio.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="radio-withcode.htm">radio-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="interactive.php?fn=radio.htm">interactive.php</a><br />
<table><tr><td><font point-size=14><b>
10.8 Form - </b></font></td><td><font point-size=14><b>select
(element)</b></font></td></tr>
<tr><td>&nbsp;</td><td><font point-size=14><b>onchange
(event)</b></font></td></tr></table>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="select.htm">select.htm</a>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="select-withcode.htm">select-withcode.htm</a><br>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="interactive.php?fn=select.htm">interactive.php</a><br />
</h3>
</td>
<td width=55% height=100% valign=top>
<iframe src=attributesdemo.htm width=100% height=100%
frameborder=0></iframe>
</td>
</tr></table>
</body>
</html>
```

Figure 10.2 shows the menu as displayed by the browser. The web page with the menu also includes Table 10.1 Elements and Event Attributes which is not shown in the figure.

There are two versions of each web page, except the menu: with and without the source code displayed. The name of the web page with the source code displayed is the name of the web page without the source code followed by a hyphen and the letters, withcode. Also, the web page with the source code is displayed in an interactive web page, which is explained below. So, each demonstration is has three links in the menu. For the first demonstration, First JavaScript Page, the links are hello.htm, hello-withcode.htm and interactive.php. Note that the names of the links are the same as the names of the web pages.



Menu for JavaScript

10.1 First JavaScript Page

hello.htm hello-withcode.htm
interactive.php

10.2 Dialog Boxes

dialog.htm dialog-withcode.htm
interactive.php

10.3 Global Objects

object.htm object-withcode.htm
interactive.php

10.4 Form - text, password (elements)
onchange, onkeypress (events)

text.htm text-withcode.htm
interactive.php

10.5 Form - textarea (element)
ondblclick, onfocus, onblur (events)

textarea.htm textarea-withcode.htm

A textarea box cannot be demonstrated interactively because the interactive web page uses a textarea box to display the html code.

10.6 Form - button, submit, reset (elements)
onclick (event)

button.htm button-withcode.htm
interactive.php

10.7 Form - radio, checkbox (elements)
onclick (event)

radio.htm radio-withcode.htm
interactive.php

10.8 Form - select (element)
onchange (event)

select.htm select-withcode.htm
interactive.php

Figure 10.2 Menu for web pages: menujs.htm

The web page, interactive.php, for each demonstration shows the source code on the left side of the page and the output of the source code on the right side. The source code can be changed by typing on the left side and then Submit can be clicked to see the output of the changed source code on the right side. The original source code and its output can be displayed by clicking, original page.

10.9.2 First JavaScript Page

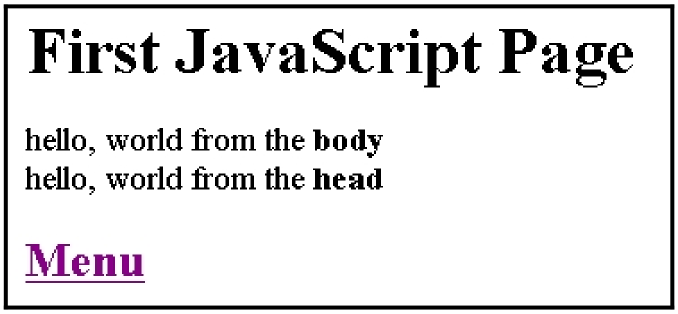
By tradition the first program in any programming language displays the message, hello, world. JavaScript instructions are identified by tags and can be embedded in the head section or the body section or both of the web page. In the web page, hello.htm, a JavaScript instruction in the body section displays a message and then calls a function in the head section which also displays a message. The messages are displayed by the method, write(), of the document object.

Web page 10.2 hello.htm

```
<html>
<head>
<title> First JavaScript Page </title>
<script type="text/javascript">
//function definition
function firstfunction()
{
    document.write("<br>hello, world from the <b>head</b>");
}
</script>
</head>
<body>
<h1>First JavaScript Page</h1>
<script type="text/javascript">
    document.write("hello, world from the function in the head
called from the <b>body</b>");
    //function call
    firstfunction();
</script>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

Click the link, hello.htm, in the menu. The web server sends the file to the browser. Then the browser executes the file starting with the head section and then the body section. When the browser reads a JavaScript tag, it stops execution and the JavaScript interpreter executes the instructions within the JavaScript tags and replaces those instructions with the output (if any) of the instructions. Then the browser continues to execute the file starting with any output that was produced by the JavaScript interpreter. This process continues until the browser has finished executing the file and the page is displayed.

Figure 10.3 shows the display of the web page, hello.htm.



First JavaScript Page

hello, world from the **body**
hello, world from the **head**

Menu

Figure 10.3 First JavaScript web page: hello.htm

10.9.3 Dialog Boxes

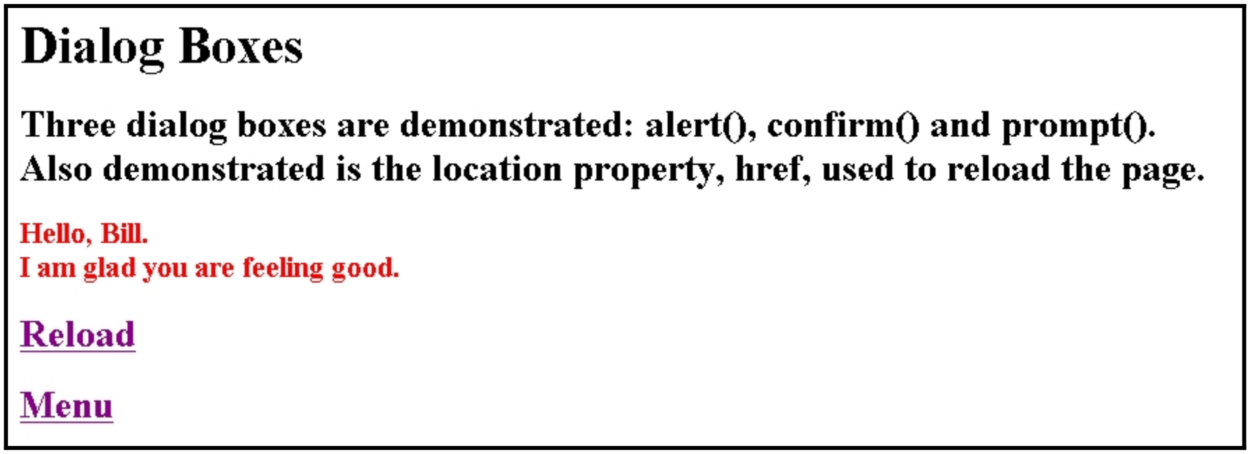
There are three dialog boxes which are displayed by methods of the window object: alert(), confirm() and prompt(). In the web page, dialog.htm, the dialog boxes are demonstrated.

Web page 10.3 dialog.htm

```
<html>
<head>
<title> Dialog Boxes </title>
</head>
<body>
<h1>Dialog Boxes</h1>
<h2>Three dialog boxes are demonstrated: alert(), confirm() and
prompt().<br>
Also demonstrated is the location property, href, used to reload
the page.</h2>
<script type="text/javascript">
//alert()
alert("This dialog box is produced by the method, alert().\n" +
      "This box is usually used to give instructions or report
errors to the user.\n\n" +
      "Instructions\nIn the next dialog box, answer the question
by clicking OK or Cancel.\n" +
      "Then in the next dialog box, type your name in the text
box and after click OK.\n" +
      "Click Reload to start the script again.  This time leave
the text box blank\n" +
      "and click OK, or type your name and then click Cancel.  "
+
      "An alert dialog box appears\nwith the instruction to type
your name.");
//confirm()
var reply = confirm("This dialog box is produced by the method,
confirm(),\n\n" +
      "How are you feeling?  Click OK for good or Cancel for 
bad.");
//prompt and alert
var yourname = "";
while (yourname == null || yourname == "")
{
   var yourname = prompt("This dialog box is produced by the 
method, prompt().\n" +
      "What is your name?", "");
   if (yourname == null || yourname == "")
   {
   //prompt returns null if Cancel is clicked
   //prompt returns "" if OK is clicked when the box is empty
   //alert("yourname = " + yourname);
      alert("Please type your name in the prompt dialog box and 
then click OK.");
   }
}
//message which uses the return of both confirm and prompt
if (reply == true)
      document.write("<font color=red><h3>Hello, " + yourname + 
".<br>" +
                     "I am glad you are feeling 
good.</h3></font>");
else
      document.write("<font color=blue><h3>Hello, " + yourname + 
".<br>" +
                     "I am sorry you are feeling 
bad.</h3></font>");

//link to current page to execute the page again
//location.href is the current URL
document.write("<h2><a href=" + location.href + 
">Reload</a></h2>");
</script>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

Click the link, dialog.htm, in the menu. Execute the script a few times by clicking Reload.

Figure 10.4 shows the display of the web page, dialog.htm, after clicking OK in the first two dialog boxes and typing, Bill, and clicking OK in the third dialog box.



Dialog Boxes

Three dialog boxes are demonstrated: alert(), confirm() and prompt(). Also demonstrated is the location property, href, used to reload the page.

Hello, Bill.
I am glad you are feeling good.

Reload

Menu

Figure 10.4 Dialog box web page: dialog.htm

10.9.4 Global Objects

There are a number of built-in objects in core JavaScript. They have global scope, meaning that they are accessible everywhere in a program, and so they are called global objects.
The most common of the global objects are: 1. Array, 2. Boolean, 3. Date, 4. Math, 5. Number, 6. RegExp, and 7. String. They are demonstrated in the web page, object.htm.

Web page 10.4 object.htm

```
<html>
<head>
<title> Global Objects </title>
</head>
<body>
<h1>Global Objects</h1>
<script type="text/javascript">
/**************************/
/***** 1. Array object*****/
/**************************/
var i;
document.write("<h2>1. Array object</h2>");
var a1 = new Array();
document.write("The length of the array created by \"var a1 = new
Array()\" = " + a1.length);
document.write("<br>The array elements are: ");
for(i=0; i < a1.length; i++)
    document.write("<br>" + a1[i]);
var a2 = new Array(7);
document.write("<p>The length of the array created by \"var a2 =
new Array(7)\" = " + a2.length);
document.write("<br>The array elements are: ");
for(i=0; i < a2.length; i++)
{
   if (i ==0)
```

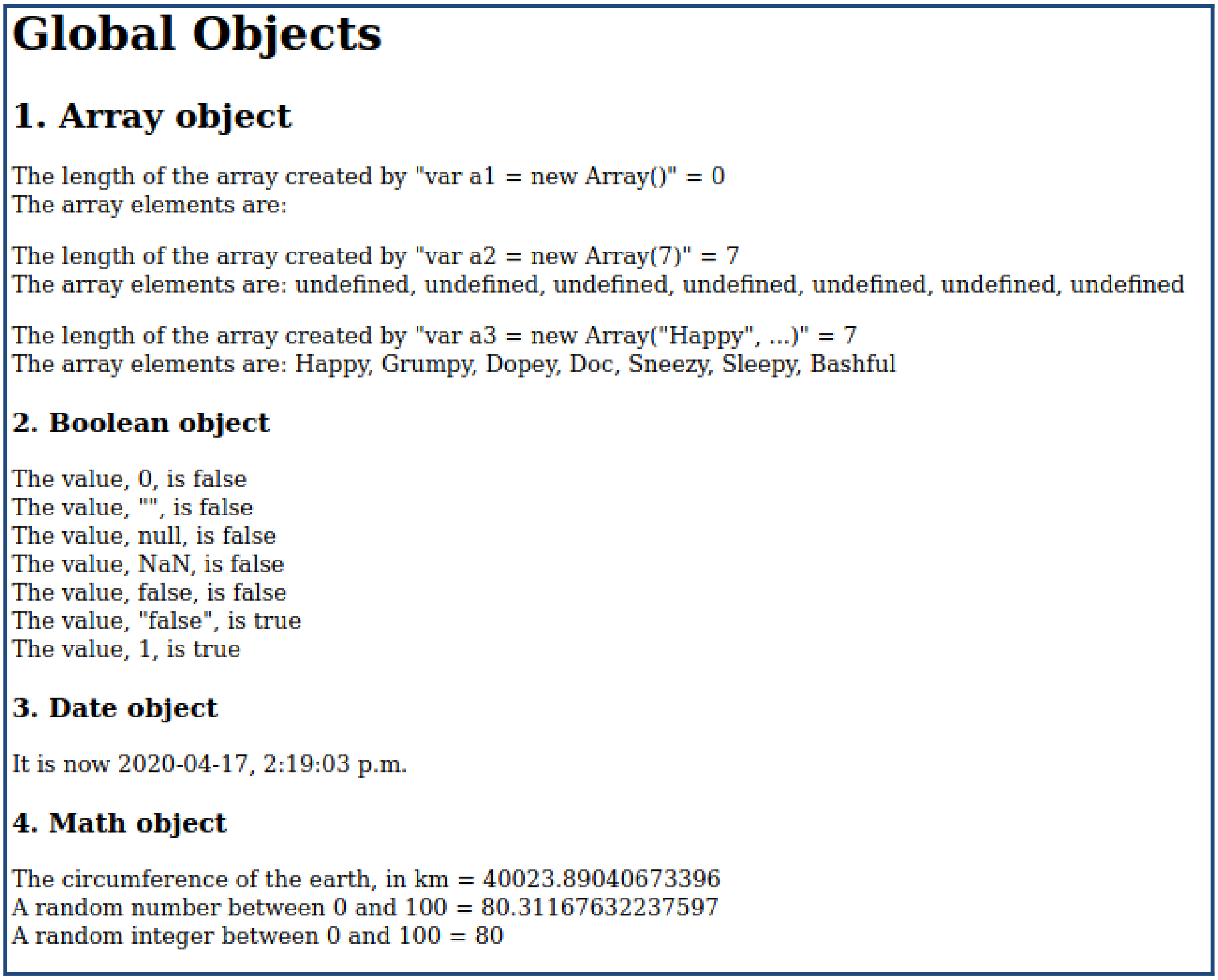
```
      document.write(a2[i]);
   else
      document.write(", " + a2[i]);
}
var a3 = new Array("Happy", "Grumpy", "Dopey", "Doc", "Sneezy",
"Sleepy", "Bashful");
document.write("<p>The length of the array created by \"var a3 =
new Array(\"Happy\", ...)\" = " +
               a3.length);
document.write("<br>The array elements are: ");
for(i=0; i < a3.length; i++)
{
   if (i ==0)
      document.write(a3[i]);
   else
      document.write(", " + a3[i]);
}
/***************************/
/*****2. Boolean object*****/
/***************************/
document.write("<h2>2. Boolean object</h2>");
var boolarray = new Array(0, "", null, NaN, false, "false", 1);
//boolarray2 is used for displaying the array elements
var boolarray2 = new Array(0, '""', null, NaN, false, '"false"',
1);
for(i=0; i < boolarray.length; i++)
{
   document.write("The value, " + boolarray2[i] + ", is " + new
Boolean(boolarray[i]) + "<br>");
}
/************************/
/*****3. Date object*****/
/************************/
document.write("<h2>3. Date object</h2>");
var d = new Date();       //create date object, d, set to current
date and time
var datetime = d.toLocaleString();   //convert d to a date and
time string, datetime
document.write("It is now " + datetime);  //display date and time
/************************/
/*****4. Math object*****/
/************************/
document.write("<h2>4. Math object</h2>");
//constant
var p = Math.PI;          //value of the constant, PI.
var radius = 6370;        //radius of earth in km
```

```
document.write("The circumference of the earth, in km = " +
2*p*radius);
//function
var r = Math.random();    //random number between 0 and 1
document.write("<br>A random number between 0 and 100 = " +
100*r);
document.write("<br>A random integer between 0 and 100 = " +
Math.round(100*r));
/*************************/
/*****5. Number object*****/
/*************************/
document.write("<h2>5. Number object</h2>");
//constants
document.write("<h3>Constants</h3>")
document.write("Number.MAX_VALUE = " + Number.MAX_VALUE +
"<br>");
document.write("Number.MIN_VALUE = " + Number.MIN_VALUE +
"<br>");
document.write("Number.NEGATIVE_INFINITY = " +
Number.NEGATIVE_INFINITY + "<br>");
document.write("Number.POSITIVE_INFINITY = " +
Number.POSITIVE_INFINITY + "<br>");
document.write("Number.NaN = " + Number.NaN + "<br>");
//some methods
document.write("<h3>Some methods</h3>");
var n1 = 1234.5678;
document.write("number.toExponential(digits): " + n1 + " = " +
n1.toExponential(4) + "<br>");
document.write("number.toFixed(digits): " + n1 + " = " +
n1.toFixed(2) + "<br>");
var n2 = 256;
document.write("Conversion of decimal to hexadecimal -
number.toString(radix): " +
n2 + " = " + n2.toString(16) + "<br>");
/*************************/
/*****6. RegExp object*****/
/*************************/
document.write("<h2>6. RegExp object</h2>");
var re = /s$/;
var stringarray = new Array("cat", "cats", "CATS");
for(i=0; i < stringarray.length; i++)
{
   document.write("The string, \"" + stringarray[i] + "\", ends
with s: " +
                  re.test(stringarray[i]) + "<br>");
}
/*************************/
```

```
/*****7. String object*****/
/***************************/
document.write("<h2>7. String object</h2>");
var str = "hello, world";
document.write("The length of \"hello, world\" = " + str.length +
"<br>");
document.write("The position of the first \"o\" in \"hello,
world\" = " +
               str.indexOf("o") + "<br>");
document.write("The position of the last \"o\" in \"hello,
world\" = " +
               str.lastIndexOf("o") + "<br>");
document.write("The position of the first \"x\" in \"hello,
world\" = " +
               str.indexOf("x") + "<br>");
document.write("The first five characters of \"hello, world\" = "
+
               str.substr(0, 5) + "<br>");
document.write("The second five characters of \"hello, world\" =
" +
               str.substr(5, 5) + "<br>");
document.write("The string \"hello, world\" converted to
uppercase = " +
               str.toUpperCase(str) + "<br>");
var u = 123;
var v = 456;
var w = u + v;   //numbers are added
var x = u.toString();  //convert number to string
var y = v.toString();  //convert number to string
var z = x + y;   //strings are concatenated
document.write("Numbers: " + u + " + " + v + " = " + w);
document.write("<br>Strings: " + x + " + " + y + " = " + z);
//link to current page to execute the page again
//location.href is the current URL
document.write("<h2><a href=" + location.href +
">Reload</a></h2>");
</script>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

Click the link, object.htm, in the menu. Execute the script a few times by clicking Reload and note that the time changes (date object) and the random number changes (math object) each time.

Figure 10.5 shows part of the display of the web page, object.htm.



Global Objects

1. Array object

The length of the array created by "var a1 = new Array()" = 0
The array elements are:

The length of the array created by "var a2 = new Array(7)" = 7
The array elements are: undefined, undefined, undefined, undefined, undefined, undefined, undefined

The length of the array created by "var a3 = new Array("Happy", ...)" = 7
The array elements are: Happy, Grumpy, Dopey, Doc, Sneezy, Sleepy, Bashful

2. Boolean object

The value, 0, is false
The value, "", is false
The value, null, is false
The value, NaN, is false
The value, false, is false
The value, "false", is true
The value, 1, is true

3. Date object

It is now 2020-04-17, 2:19:03 p.m.

4. Math object

The circumference of the earth, in km = 40023.89040673396
A random number between 0 and 100 = 80.31167632237597
A random integer between 0 and 100 = 80

Figure 10.5 Part of object web page: object.htm

Elements
text: The text element stores one line of text. The user can enter the text by typing on the keyboard, or the script can enter text by using the value property of the element. For example if the form has the name, formtext, and the text element has the name, firstname, then the contents of the text element is

document.formtext.firstname.value

and a string can be assigned to the text element.

password: The password element is the same as the text element except that each character typed is displayed as an asterisk (*) so that someone who is watching the user type the password cannot read the password on the screen.

Events
onchange: The change event occurs when the contents of a text or password element changes and then that element loses the keyboard focus, meaning that the cursor is no longer in the element. An element loses the focus when Tab is pressed or another element is clicked. A JavaScript function, called an event handler, can be registered with the browser so that the

function is executed when an event occurs. An event handler is registered for an element by an attribute of the element which is the name of the event with the prefix, on. So for the event, change, the event handler, *functionname*(), is registered by onchange, using the syntax,

onchange=*functionname*()

The event handlers (functions) which are executed after the change event in the demonstration web page validate the data in the demonstration web page. If the data is invalid, a dialog box gives the correct format of the text or password element and after the dialog box is closed, the focus returns to the same element and the value of the element is removed by setting the value of the element to the zero-length string (″″).

10.9.5 Form - text, password (elements); onchange, onkeypress (events)

onkeypress: The keypress event occurs when a key is pressed and then released. The event handler, *functionname*(), is registered by the attribute, onkeypress, with the syntax,

onkeypress=return *functionname*()

If the key which is pressed is space or Enter, in the demonstration web page the event handler returns false and the key stroke is cancelled, and so space and Enter cannot be part of the text in the element. Otherwise, the event handler returns true and the character is appended to the text already in the element.

The elements and events described above are demonstrated by the web page, text.htm.

Web page 10.5 text.htm

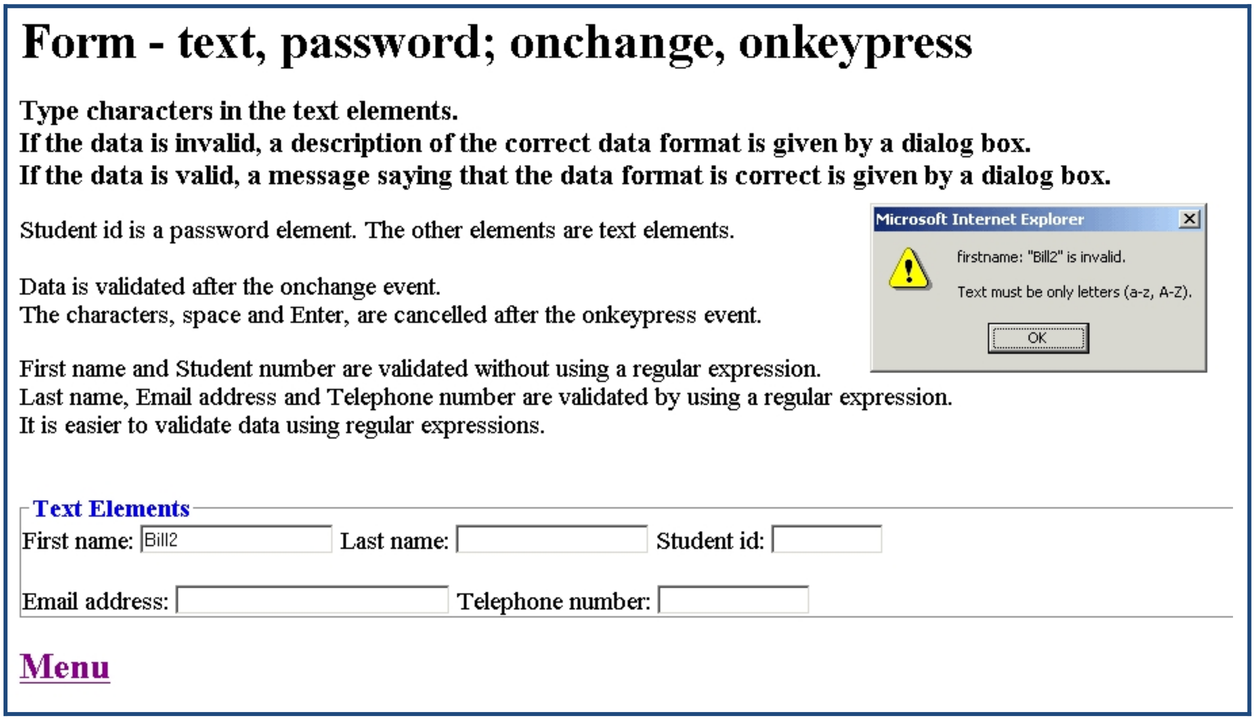
```
<html>
<head>
    <title>Forms and JavaScript</title>
<script type="text/javascript">
//global variable for form used in onchange event functions:
//     var f = document.formtext;
//This is defined after the form is defined
//so that the name of the form, formtext, is defined.
//If the name of the form is changed,
//the name needs to be changed only in the global variable
//and not in every function where the global variable is used.
/**********************************
*First name onchange event function*
*without regular expression        *
**********************************/

function vldt_firstname_without_regexp()
/*
* Validate first name without using a regular expression.
* First name is at least 2 characters long and at most 10
characters
* long, and contains only letters (a-z, A-Z).
* So a compound name, like Jean-Luc, is not allowed.
*/
{
//declare local variables
var e = f.firstname;
var fn = e.value;
var fnlower, len, errorflag, ch;
if (fn == "")
//no text
    return;
len = fn.length;
//validate length
if (len < 2 || len > 10)
{
  alert(e.name + ": \"" + fn + "\" is invalid.\n\n" +
        "Text must be between 2 and 10 characters in length." );
  f.lastname.focus();  //necessary to execute next instruction
  e.focus();
  e.value = "";
  }
else
{
  //validate characters
  fnlower = fn.toLowerCase();
  errorflag = false;
  for (var i = 0; i < len; i++)
  {
     ch = fnlower.charAt(i);
     if (ch >= 'a' && ch <= 'z')
   continue;
     else
      {
        alert(e.name + ": \"" + fn + "\" is invalid.\n\n" +
              "Text must be only letters (a-z, A-Z).");
        f.lastname.focus();  //necessary to execute next
instruction
        e.focus();
        e.value = "";
   errorflag = true;
   break;
      }
  }
  if (errorflag == false)
      alert("The data format is correct.");
}
}
/********************************
```

```
*Text onkeypress event function*
*******************************/
function cancelCharacter()
/*
* Cancel the characters, space and Enter
* ASCII codes:
*    space: 32 in decimal
*    Enter: 13 in decimal
*/
{
var kc = event.keyCode;
//test: remove //
//alert("keycode for " + String.fromCharCode(kc) +
// " = " + kc + ", in decimal");
if (kc == 32 || kc == 13)
    return false;
return true;
}
/**********************************
*Last name onchange event function*
**********************************/
function vldt_lastname()
/*
* Validate last name by using a regular expression.
* First name is at least 2 characters long and at most 10
characters long,
* and contains only letters (a-z, A-Z).
* So a compound name, like Jean-Luc, is not allowed.
*/
{
//declare local variables
var e = f.lastname;
var ln = e.value;
var re, ret, msg;
if( ln == "")
//no text
   return;
re = /^[a-z]{2,10}$/i;
ret = ln.search(re);
if (ret == -1)
  {
  msg = e.name + ": \"" + ln + "\" is invalid.\n\n";
  msg += "Text must be only letters (a-z, A-Z) and\n";
  msg += "must be between 2 and 10 characters in length.";
  alert(msg);
  f.firstname.focus();  //necessary to execute next instruction
  e.focus();
  e.value = "";
  }
else
  alert("The data format is correct.");
}
/***********************************
*Student id onchange event function*
***********************************/
function vldt_id()
/*
* Validate id without using a regular expression.
* The id is a 7-digit integer number that begins with 1 or 2 ...
or 9
* Leading zeros are ignored.  Digits after the decimal point are
ignored.
*/
{
//declare local variables
var e = f.id;
var ident = e.value;
if(ident == "")
//no text
   return;
if(isNaN(ident))
  {
    alert(e.name + ": \"" + ident + "\" is invalid.\n\nThe id is
a number.");
    f.lastname.focus();  //necessary to execute next instruction
    e.focus();
    e.value = "";
  }
else
  {
    ident = Math.floor(ident);   //truncates a number with a
decimal point
    if (ident < 1000000 || ident > 9999999)
    {
        alert(e.name + ": \"" + ident + "\" is invalid.\n\n" +
              "The id is a 7-digit integer number that does not
begin with 0.");
        f.lastname.focus();  //necessary to execute next
instruction
        e.focus();
        e.value = "";
    }
    else
```

```
        //alert("Used for testing\n" + e.name + " = " +
ident);  //for testing
        alert("The data format is correct.");
  }
}
/*****************************
*Email onchange event function*
*****************************/
function vldt_email()
/*
* Validate email address
* Format: x@y.z
*     x: one or more word characters at beginning: /^\w+
*        zero or one period (.) or hyphen (-)
*        followed by one or more word characters,
*        repeated zero or more times: ([\.-]?\w+)*
*     y: same as x
*     z: two or three letters (a-z) at end: [a-z]{2,3}$/
* Definition of word character: letter (A-Z, a-z), or digit (0-9)
* or underscore (_)
* Symbol for word character: \w
* Literal character, dot(.): \.
* Symbol for zero or more times: *
* Symbol for one or more times: +
* Symbol for zero or one time: ?
* Delimiters (beginning and end) of regular expression: /
* Beginning of string: ^
* End of string: $
*/
{
//declare local variables
var e = f.email;
var em = e.value;
var re, ret, msg;
if( em == "")
//no text
   return;
re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*\.[a-z]{2,3}$/
ret = em.search(re);
if (ret == -1)
  {
  msg = e.name + ": \"" + em + "\" is invalid.\n\n";
  msg += "Format: x@y.z\n";
  msg += "     x: one or more word characters at beginning,\n";
  msg += "         zero or one period (.) or hyphen (-)";
  msg += " followed by one or more word characters,\n";
  msg += "         repeated zero or more times\n";
  msg += "     y: same as x\n";
  msg += "     z: 2 or 3 lowercase letters\n";
  msg += "A word character is a letter (A-Z, a-z),";
  msg += " or digit (0-9) or underscore (_).\n";
  msg += "Example: myusername@cs.concordia.ca";
  alert(msg);
  f.firstname.focus();  //necessary to execute next instruction
  e.focus();
  e.value = "";
  }
else
  alert("The data format is correct.");
}
/**********************************
*Telephone onchange event function*
**********************************/
function vldt_tel()
/*
* Validate telephone number by using a regular expression.
* The telephone number begins with a 3-digit area code, then a
hyphen (-), then 3 digits,
* then a hyphen (-) and then 4 digits.
* The area code cannot start with 0.
*/
{
//declare local variables
var e = f.telephone;
var tel = e.value;
var re, ret, msg;
if( tel == "")
//no text
   return;
re = /^[1-9]\d{2}-\d{3}-\d{4}$/;
ret = tel.search(re);
if (ret == -1)
  {
  msg = e.name + ": \"" + tel + "\" is invalid.\n\n";
  msg += "An example of a valid telephone number is
514-848-2424,\n";
  msg += "where the first digit is not 0.";
  alert(msg);
  f.id.focus();  //necessary to execute next instruction
  e.focus();
  e.value = "";
  }
else
  alert("The data format is correct.");
```

```
}
</script>
</head>
<body>
<h1>Form - text, password; onchange, onkeypress</h1>
<h3>
Type characters in the text elements.<br>
If the data is invalid, a description of the correct data format
is given by a dialog box.<br>
If the data is valid, a message saying that the data format is
correct is given by a dialog box.<br></h3>
Student id is a password element.  The other elements are text
elements.
<br><br>
Data is validated after the onchange event.<br>
The characters, space and Enter, are cancelled after the
onkeypress event.
<p>
First name and Student number are validated without using a
regular expression.<br>
Last name, Email address and Telephone number are validated by
using a regular expression.<br>
It is easier to validate data using regular expressions.
<p><br>
<form name="formtext" method=post enctype="text/plain">
<fieldset>
<!-- Text Elements -->
<legend> <font color=blue><b>Text Elements</b></font> </legend>
First name: <input type="text" name="firstname"
onchange="vldt_firstname_without_regexp()"
    onkeypress="return cancelCharacter()">
Last name: <input type="text" name="lastname"
onchange="vldt_lastname()"
    onkeypress="return cancelCharacter()">
Student id: <input type="password" name="id" onchange="vldt_id()"
    onkeypress="return cancelCharacter()" size=10 maxlength=10>
<br><br>Email address: <input type="text" name="email"
onchange="vldt_email()"
    onkeypress="return cancelCharacter()" size=30>
Telephone number: <input type="text" name="telephone"
onchange="vldt_tel()"
    onkeypress="return cancelCharacter()" size=15>
</fieldset>
</form>
<script type="text/javascript">
//global variable for form used in onchange event functions
//defined here because form name, formtext, is defined
```

```
//only after the form is defined
var f = document.formtext;
</script>
<p>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```



Form - text, password; onchange, onkeypress

Type characters in the text elements.
If the data is invalid, a description of the correct data format is given by a dialog box.
If the data is valid, a message saying that the data format is correct is given by a dialog box.

Student id is a password element. The other elements are text elements.

Data is validated after the onchange event.
The characters, space and Enter, are cancelled after the onkeypress event.

First name and Student number are validated without using a regular expression.
Last name, Email address and Telephone number are validated by using a regular expression.
It is easier to validate data using regular expressions.

Text Elements
First name: Bill2 Last name: Student id:

Email address: Telephone number:

Menu

Microsoft Internet Explorer

firstname: "Bill2" is invalid.

Text must be only letters (a-z, A-Z).

Figure 10.6 Text web page: text.htm

Click the link, text.htm, in the menu. Type several different incorrect and correct values into each of the text elements. Terminate the entry of data in a text element by either pressing Tab or clicking on another text element. Note that when space or Enter is typed, the data in the text element does not change, that is, space and Enter are cancelled. Also note that if the text element is empty, focus can be moved to another text element without the appearance of a dialog box and therefore the data can be entered into text elements in any order.

Figure 10.6 shows the display of the web page, text.htm, after the name, Bill2, is typed in the text element, First name, and Tab is pressed.

10.9.6 Form - textarea (element); ondblclick, onfocus, onblur (events)

Element

textarea: The textarea element stores more than one line of text. The user can enter the text by typing on the keyboard, or the script can enter text by using the value property of the element, just as for the text and password elements in the last section.

The visible number of rows in the textarea element is given by the attribute, rows, and the visible number of columns is given by the attribute, cols. If more data is entered than is visible, scroll bars appear so that all data can be viewed by scrolling.

Browsers impose a limit on the total number of characters that can be entered in a text area element. The limit is likely either 32 kB (32 768 characters) or 64 kB (65 536 characters). There is no attribute, like maxlength for a text element, which limits the number of characters in a textarea element.

Events

ondblclick: The dblclick (double click) event occurs when the left mouse button is quickly double-clicked anywhere inside the textarea element. The event handler is specified by the attribute, ondblclick, and is executed after the dblclick event.

The dblclick event handler in the demonstration web page enters the date and time on a new line in the textarea element. After the user double clicks, a suggestion can be typed and the date and time of when the suggestion was made is already recorded.

onfocus: The focus event occurs when the textarea element is able to receive input from the keyboard, which is indicated by a flashing cursor in the element. The textarea element gets the focus when either the mouse is clicked on the element or when Tab is pressed and the next element in the tab sequence is the textarea element. The event handler is specified by the attribute, onfocus, for the event, focus. In the demonstration web page the event handler stores the contents of the textarea element in a global variable.

onblur: The blur event occurs when the keyboard focus is removed from the textarea element. The textarea element loses the focus when either the mouse is clicked on another element or Tab is pressed. The event handler is specified by the event attribute, onblur, for the event, blur. In the demonstration web page the event handler first tests if the contents of the textarea element is the zero-length string (there is no text in the element) and if so the event handler does nothing further. If the element is not empty and the contents of the element has changed since the element received the focus, a dialog box displays the number of characters which has been added to (or subtracted from) the element and gives the choice to either cancel the change or keep the change.

The element and events described above are demonstrated by the web page, textarea.htm.

Web page 10.6 textarea.htm

```
<html>
<head>
    <title>Forms and JavaScript</title>
<script type="text/javascript">
/**********************************
*Suggestions onfocus event function*
**********************************/
function focus_sug()
/*
* Store the contents of the element in a global variable
* when the element receives the focus.
* This global variable is used to remove the change
* to the contents of the element after the onblur event,
* if this choice is selected by the user.
*/
{
//declare local variable
var e = document.formtextarea.suggestions;
//the following is a global variable because it is not declared
using var
initial_sug_value  = e.value;
}
/*********************************
*Suggestions onblur event function*
*********************************/
function blur_sug()
/*
* If the contents of the text element have changed,
* display the number of characters added and give
* the option to keep the change or remove the change
*/
{
//declare local variables
var e = document.formtextarea.suggestions;
var s = e.value;
var change;
var ret;
if( s == "")
//no text
   return;
//initial_sug_value is a global variable created by the
focus_sug() function
if (e.value != initial_sug_value)
  {
    change = e.value.length - initial_sug_value.length
    if (change == 0)
    {
     ret = confirm("You have added " + change + " characters to
     the text,\nbut characters in the text have changed.\n\n" +
          "If you want to keep the change, click OK.\n" +
         "If you want to cancel the change, click Cancel.");
    }
    else
    {
     ret = confirm("You have added " + change + " characters to
     the text.\n\n" +
          "If you want to keep the characters, click OK.\n" +
          "If you want to cancel the characters, click Cancel.");
     }
    if (ret == false)
    {
     //cancel changes
     e.value = initial_sug_value;
     e.focus();
    }
  }
}
/***********************************
*Suggestions ondblclick event function*
***********************************/
function dblclick_sug()
/*
* Insert date and time on a new line in the textarea element.
*/
{
//declare local variables
var e = document.formtextarea.suggestions;
var s = e.value;
var d = new Date();
var datestring = d.toLocaleString();
if (s != "")
//add a new line
   e.value = s + "\n"+ datestring + ": ";
else
   e.value = datestring + ": ";
}
</script>
</head>
<body>
<h1>Form - textarea; ondblclick, onfocus, onblur</h1>
<h3>
Type characters in the text element.<br>
All data entered is valid, including space and Enter characters.
</h3>
Suggestions is a textarea element.
<br><br>
The data and time are entered on a new line in the text
element after the ondblclick event.<br>
The initial contents of the text element is stored after
the onfocus event.<br>
The choice to either keep or cancel the change to the contents is
given by a dialog box after the onblur event.
<p><br>
<form name="formtextarea" method=post enctype="text/plain">
<p>
```

```
<fieldset>
<!-- Text Area Element -->
<legend> <font color=blue><b>Text Area Element</b></font>
</legend>
Suggestions:  (double-click to insert date and time)
<br><textarea name="suggestions" ondblclick="dblclick_sug()"
onfocus="focus_sug()" onblur="blur_sug()" rows=4 cols=60
maxlength=300 wrap></textarea>
</fieldset>
</form>
<p>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

Click the link, textarea.htm, in the menu. The JavaScript in this demonstration is devised to illustrate the three events, dblclick, focus and blur, and would not likely be useful in an actual application, except perhaps for the use of the dblclick event.

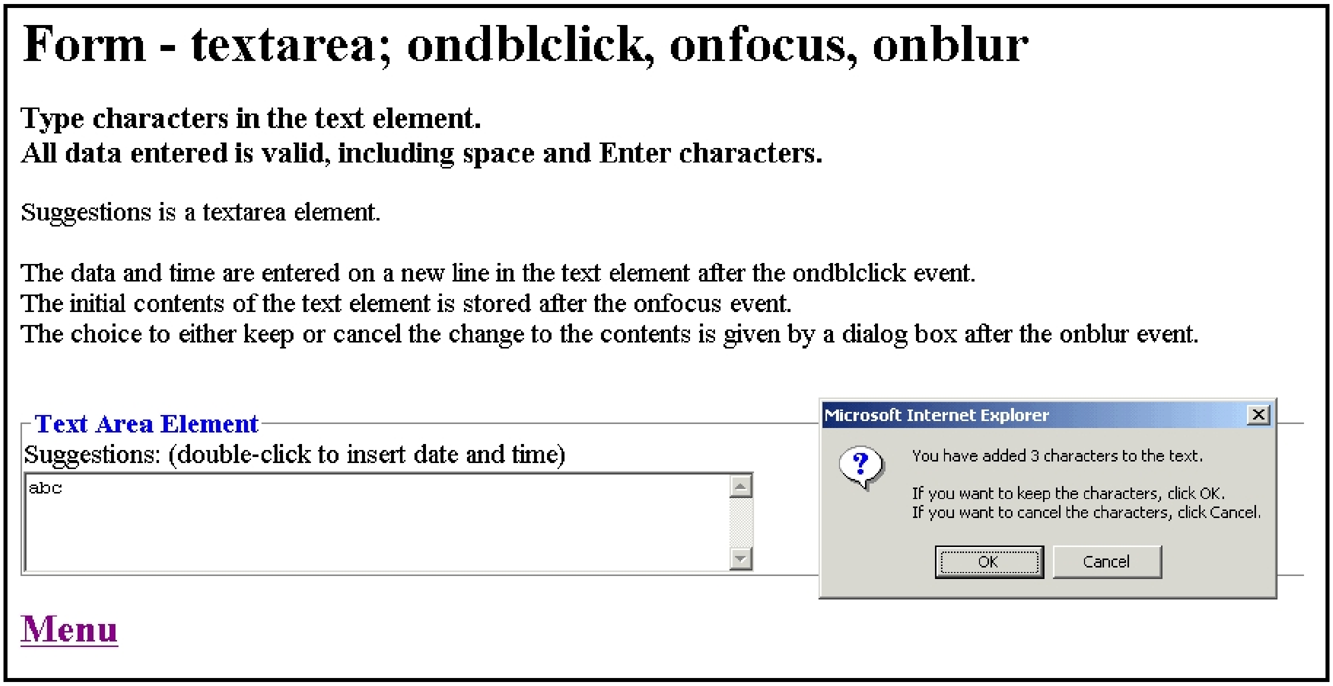
Click once in the textarea element to give it the focus. Type, abc, in the textarea element. Then double click inside the textarea element. The time and date appears on the next line. Press, Tab. A dialog box displays the number of characters, 42, which were added to the text and gives the choice to keep or cancel the change. Click, Cancel, which deletes the characters and leaves the cursor in the textarea element.

Repeat the same thing and this time click OK instead of Cancel. Then click once in the element to give it the focus again, type, abc, double click and then click Cancel in the dialog box. The last characters are deleted.

Delete all text by selecting it and then pressing, Delete. Then press, Tab. Note that when the text box is empty, the dialog box does not appear even though there was a change in the textarea element.

Type other values, then press, Tab, or click outside the element, and choose Cancel or OK in the dialog box, until it is understood what JavaScript does.

Figure 10.7 shows the display of the web page, textarea.htm, after the text, abc, is typed in the text element, Suggestions, and Tab is pressed.



Form - textarea; ondblclick, onfocus, onblur

Type characters in the text element.
All data entered is valid, including space and Enter characters.

Suggestions is a textarea element.

The data and time are entered on a new line in the text element after the ondblclick event.
The initial contents of the text element is stored after the onfocus event.
The choice to either keep or cancel the change to the contents is given by a dialog box after the onblur event.

Text Area Element
Suggestions: (double-click to insert date and time)
abc

Microsoft Internet Explorer

You have added 3 characters to the text.

If you want to keep the characters, click OK.
If you want to cancel the characters, click Cancel.

Figure 10.7 Text area web page: textarea.htm

10.9.7 Form - button, submit, reset (elements); onclick (event)

Elements

The three elements, button, submit and reset, are all elements in the button category. All are used to initiate an action. The element, button, has no default behaviour and so needs an onclick event handler (function). The other two elements do have default behaviour, as described below, but also can also have an onclick event handler.

button: The button element is used to cause some action on a web page. In the demonstration web page, a button is used to display instructions in a dialog box.

submit: The submit element is a specialized button (has default behaviour) that sends the values of all elements to the server as specified by the action attribute of the form.

reset: The reset element is a specialized button (has default behaviour) that sets the values of all elements to their default values, which is the empty string unless they are explicitly given a default value in the HTML instruction that creates the element.

Event

The basic difference between an element in the button category and an element in the text category (see Table 10.1) is that a button element responds to a click event and a text element does not, and secondly, a text element responds to a change event and a button element does not.

onclick: The click event occurs when the left mouse button is clicked anywhere inside the button element.

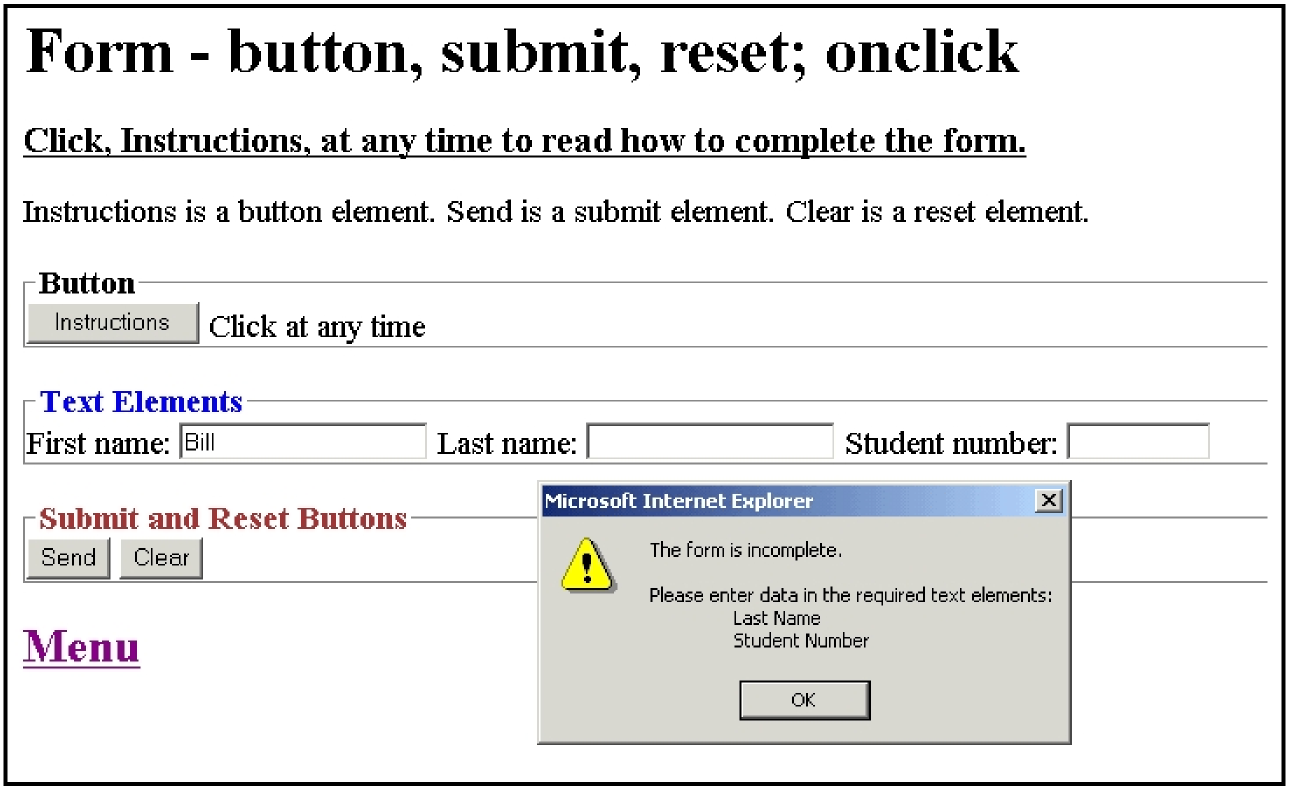
An event handler is specified by the event attribute, onclick, and is executed after the click event.

The elements and event described above are demonstrated by the web page, button.htm.
To receive an email, change the action attribute to your personal email address in the <form> tag.

Web page 10.7 button.htm

```
<html>
<head>
    <title>Forms and JavaScript</title>
<script type="text/javascript">
/************************************
*Instructions onclick event function*
************************************/
function displayInstructions()
{
var str = "\
****Text Elements****\n\
Complete the text elements at any time. This data is required.\n\
The data in the text elements is not validated until\n\
   after the onclick event for the submit button.\n\
Then each text element is tested if it is empty (a zero-length
string)\n\
   and if it is not empty it is considered to be valid.\n\n";
str += "\
****Submit and Reset Buttons****\n\
Click either button at any time.\n\
Submit\n\
   If the form is not complete, a dialog box gives instructions
to complete the form.\n\
   If the form is complete, a dialog box asks if the user wishes
to submit the form.\n\
Reset\n\
    A dialog box asks if the user wishes to clear the form.";
alert(str);
}
/******************************
*Submit onclick event function*
******************************/
function vldt_submit()
/*
* Validate user input to the form.
* If the input is incomplete,
* instruct the user to compete the form.
* If the input is complete,
* ask if form is to be submitted now.
*/
{
//declare local variables
var f = document.emailForm;
var empty, msg;
var i;
var retval;
//validate text elements
empty = "";
if (f.firstname.value == "")
    empty += "\tFirst Name\n";
if (f.lastname.value == "")
    empty += "\tLast Name\n";
if (f.id.value == "")
    empty += "\tStudent Number\n";
//form is incomplete
//construct the message and display message
msg = "";
if (empty != "")
  {
    msg = "The form is incomplete.\n\n";
    msg += "Please enter data in the required text elements:\n";
    msg += empty;
    alert (msg);
    return false;
  }
//form is complete
//ask user for confirmation to submit the data to the server
retval = confirm("The form is complete.\n\nDo you wish to submit
the form now?");
return retval;
}
/*****************************
*Reset onclick event function*
*****************************/
function vldt_reset()
/*
* Ask if form is to be reset now.
*/
{
var retval = confirm("If the form is cleared, all data entered in
the form will be removed.\n\n\
Do you wish to clear the form now?");
return retval;
}
</script>
</head>
<body>
<h1>Form - button, submit, reset; onclick</h1>
<h3>
<u><b>Click, Instructions, at any time to read how to complete
the form.</b></u>
</h3>
Instructions is a button element.
```

```
Send is a submit element.
Clear is a reset element.
<p>
<form name="emailForm" method=post enctype="text/plain"
      action="mailto:myname@cse.concordia.ca">
<!--replace the email address, after mailto:, by your email
address-->
<!-- Button -->
<fieldset>
<legend> <font color=black><b>Button</b></font> </legend>
<input type=button value="Instructions" name="instructionButton"
onclick="displayInstructions()"> Click at any time<br>
</fieldset>
<p>
<!-- Text Elements -->
<fieldset>
<legend> <font color=blue><b>Text Elements</b></font> </legend>
First name: <input type="text" name="firstname"
onchange="vldt_firstname()"
    onkeypress="return cancelCharacter()">
Last name: <input type="text" name="lastname"
onchange="vldt_lastname()"
    onkeypress="return cancelCharacter()">
Student number: <input type="password" name="id"
onchange="vldt_id()"
    onkeypress="return cancelCharacter()" size=10 maxlength=10>
<br></fieldset>
<p><p>
<!-- Submit and Reset Buttons -->
<fieldset>
<legend> <font color=brown><b>Submit and Reset Buttons</b></font>
</legend>
<input type="submit" value="Send" onclick="return vldt_submit()">
<input type="reset" value="Clear" onclick="return vldt_reset()">
<br></fieldset>
</form>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```



Form - button, submit, reset; onclick

Click, Instructions, at any time to read how to complete the form.

Instructions is a button element. Send is a submit element. Clear is a reset element.

Button

Instructions Click at any time

Text Elements

First name: Bill Last name: Student number:

Submit and Reset Buttons

Send Clear

Menu

Microsoft Internet Explorer

The form is incomplete.

Please enter data in the required text elements:
Last Name
Student Number

OK

Figure 10.8 Button web page: button.htm

Click the link, button.htm, in the menu.
Click the button, Instructions, and read the instructions.
Click the button, Send, and then click the button, Clear.
Type text in one or more text elements and then click Send and Clear.
When the form is complete, and Send is clicked and then OK, the name-value pairs of the text elements will be sent to the email address given by the action attribute in the form.

Figure 10.8 shows the display of the web page, button.htm, after only the text, Bill, is typed in the text element, First name, and the button, Send, is clicked.

10.9.8 Form - radio, checkbox (elements); onclick (event)

Elements

radio: The radio element is one of a group of two or more buttons, which are small circles whose appearance resemble pushbuttons on some radios. The radio buttons represent mutually exclusive options (data items), that is, only one button can be selected at any time. When a button is clicked, the previously selected button is deselected. If a button is selected, clicking the same button again does not deselect the button. In the demonstration, radio buttons represent different ranges of age of the user who is completing the form. If the age is not known for some reason, another radio button is need with the label, for example, Not Known, because a radio button cannot be deselected.

checkbox: The checkbox element is one of a group of one or more boxes which are small squares. The checkboxes represent non-mutually exclusive options (data items), that is, more than one box can be selected at any time. If a box is selected, clicking the same box again deselects the box. In the demonstration, the checkboxes represent different courses that could have been taken by the user who is completing the form.

Event
onclick: The click event occurs when the left mouse button is clicked anywhere inside the radio or checkbox element. An event handler is specified by the attribute, onclick, and is executed after the click event.

The elements and event described above are demonstrated by the web page, radio.htm.
To receive an email, change the action attribute to your personal email address in the <form> tag.

Web page 10.8 radio.htm

```
<html>
<head>
    <title>Forms and JavaScript</title>
<script type="text/javascript">
/************************************
************************************
*Instructions onclick event function*
************************************
************************************/
function displayInstructions()
{
var str = "\
****Text Elements****\n\
Complete the text elements at any time. This data is required.\n\
The data in the text elements is not validated until\n\
    after the onclick event for the submit button.\n\
Then each text element is tested if it is empty (a zero-length
string)\n\
    and if it is not empty it is considered to be valid.\n\n";
str += "\
****Radio Buttons****\n\
Choose a radio button. This is required.\n\
The data is validated after the onclick event for the submit
button.\n\n";
str += "\
****Checkboxes****\n\
Select one or more checkboxes. This is optional.\n\
If no checkboxes are selected and the form is complete,\n\
    after the onclick event for the submit button, a dialog box
asks\n\
    if the user wishes to make a selection before the form is
submitted.\n\n";
str += "\
****Submit and Reset Buttons****\n\
Click either button at any time.\n\
Submit\n\
    If the form is not complete, a dialog box gives instructions
to complete the form.\n\
    If the form is complete, a dialog box asks if the user wishes
to submit the form.\n\
Reset\n\
    A dialog box asks if the user wishes to clear the form.";
alert(str);
}
/******************************
******************************
*Submit onclick event function*
******************************
******************************/
function vldt_submit()
/*
* Validate user input to the form.
* If the input is incomplete,
* instruct the user to compete the form.
* If the input is complete,
* ask if form is to be submitted now.
*/
{
//declare local variables
var f = document.emailForm;
var empty, msg;
var i;
var radioChecked, checkboxChecked;
var retval;
//validate text elements
empty = "";
if (f.firstname.value == "")
    empty += "\tFirst Name\n";
if (f.lastname.value == "")
    empty += "\tLast Name\n";
if (f.id.value == "")
    empty += "\tStudent Number\n";
//validate radio buttons
radioChecked = false;
for (i = 0; i < f.age.length; i++)
    if (f.age[i].checked)
    {
        radioChecked = true;
        break;
    }
//validate checkboxes
```

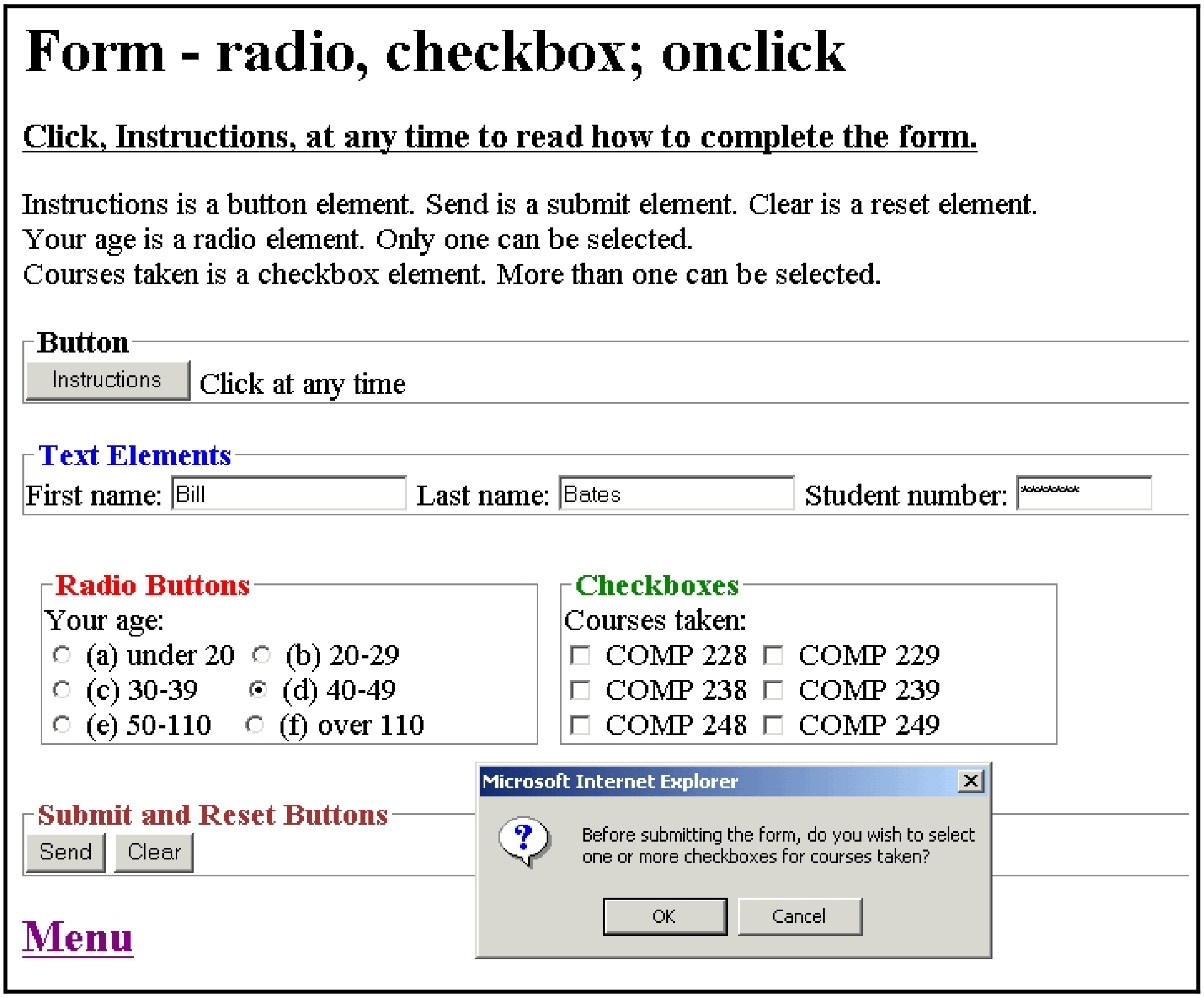
```
checkboxChecked = false;
for (i = 0; i < f.courses.length; i++)
    if (f.courses[i].checked)
    {
        checkboxChecked = true;
        break;
    }
//construct the message
msg = "";
if (empty != "")
  {
    msg = "Please enter data in the required text elements:\n";
    msg += empty;
  }
if (radioChecked == false)
    msg += "Please click the radio button for your age.\n";
//inform user of missing data
if (msg != "")
  {
    msg = "The form is incomplete.\n\n" + msg;
    alert (msg);
    return false;
  }
else if (checkboxChecked == false)
  {
    retval = confirm("Before submitting the form, do you wish to
select\n" +
        "one or more checkboxes for courses taken?");
    if (retval == true)
       return false;
  }
 /form is complete
//ask user for confirmation to submit the data to the server
var retval = confirm("The form is complete.\n\nDo you wish to
submit the form now?");
return retval;
}
/******************************
******************************
*Reset onclick event function*
******************************
*****************************/
function vldt_reset()
/*
* Ask if form is to be reset now.
*/
{
```

```
retval = confirm("If the form is cleared, all data entered in the
form will be removed.\n\n\
Do you wish to clear the form now?");
return retval;
}
/*****************************
******************************
*Radio onclick event function*
******************************
*****************************/
function displayradio(val)
/*
* display in an alert box the button which is chosen
* val = the value of the radio button which is chosen
*/
{
   alert("You chose the button: " + val);
}
/********************************
*********************************
*Checkbox onclick event function*
*********************************
********************************/
function displaycheckbox(box)
/*
* display in an alert box the checkbox
* which is either checked or unchecked and
* the list of all checkboxes which are checked
* box = this keyword, which is the checkbox which was clicked
*     = document.emailForm.course[i]
* The use of this is mandatory, since the value of i is not
known,
* i = 0 to 5 for six checkboxes
*/
{
   var f = document.emailForm;
   var check = box.checked;    //true if checked, false if
unchecked
   var sel;
   var courselist = "";
   if (check == true)
      sel = "checked";
   else
      sel = "unchecked";
   for (i = 0; i < f.courses.length; i++)
   {
      if (f.courses[i].checked)
```

```
            courselist += "\t" + f.courses[i].value + ",\n";
   }
   if (courselist == "")
      courselist = "\tnone";
   else
      //remove the newline and comma from the end of the list
      courselist = courselist.substr(0, courselist.length - 2);
   alert("You " + sel + "  the box: " + box.value + "\n\n" +
         "Boxes checked:\n" + courselist);
}
</script>
</head>
<body>
<h1>Form - radio, checkbox; onclick</h1>
<h3>
<u><b>Click, Instructions, at any time to read how to complete
the form.</b></u>
</h3>
Instructions is a button element.
Send is a submit element.
Clear is a reset element.<br>
Your age is a radio element. Only one can be selected.<br>
Courses taken is a checkbox element. More than one can be
selected.
<p>
<form name="emailForm" method=post enctype="text/plain"
      action="mailto:myname@cs.concordia.ca">
<!--replace the email address, after mailto:, by your email
address-->
<!-- Button -->
<fieldset>
<legend> <font color=black><b>Button</b></font> </legend>
<input type=button value="Instructions" name="instructionButton"
onclick="displayInstructions()"> Click at any time
</fieldset>
<p>
<!-- Text Elements -->
<fieldset>
<legend> <font color=blue><b>Text Elements</b></font> </legend>
First name: <input type="text" name="firstname">
Last name: <input type="text" name="lastname">
Student number: <input type="password" name="id"size=10
maxlength=10>
</fieldset>
<p>
<table cellspacing=10>
<tr>
```

```
<td width=300>
<!-- Radio Buttons -->
<fieldset>
<legend> <font color=red><b>Radio Buttons</b></font> </legend>
Your age:
<br><input type="radio" name="age"
onclick=displayradio(this.value) value="age&lt;20"> (a) under 20
<input type="radio" name="age" onclick=displayradio(this.value)
value="age20-29"> (b) 20-29
<br><input type="radio" name="age"
onclick=displayradio(this.value) value="age30-39"> (c) 30-39
&nbsp;&nbsp;&nbsp;&nbsp;<input type="radio" name="age"
onclick=displayradio(this.value) value="age40-49"> (d) 40-49
<br><input type="radio" name="age"
onclick=displayradio(this.value) value="age50-110"> (e) 50-110
&nbsp;&nbsp;<input type="radio" name="age"
onclick=displayradio(this.value) value="age&gt;110"> (f) over 110
</fieldset>
</td>
<td width=300>
<!-- Checkboxes -->
<fieldset>
<legend> <font color=green><b>Checkboxes</b></font>
</legend>
Courses taken:
<br><input type="checkbox" name="courses"
onclick=displaycheckbox(this) value="c228"> COMP 228
<input type="checkbox" name="courses"
onclick=displaycheckbox(this) value="c229"> COMP 229
<br><input type="checkbox" name="courses"
onclick=displaycheckbox(this) value="c238"> COMP 238
<input type="checkbox" name="courses"
onclick=displaycheckbox(this) value="c239"> COMP 239
<br><input type="checkbox" name="courses"
onclick=displaycheckbox(this) value="c248"> COMP 248
<input type="checkbox" name="courses"
onclick=displaycheckbox(this) value="c249"> COMP 249
</fieldset>
</td>
</tr>
</table>
<p>
<!-- Submit and Reset Buttons -->
<fieldset>
<legend> <font color=brown><b>Submit and Reset
Buttons</b></font> </legend>
```

```
<input type="submit" value="Send" onclick="return
vldt_submit()">
<input type="reset" value="Clear" onclick="return
vldt_reset()">
<br></fieldset>
</form>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```



Form - radio, checkbox; onclick

Click, Instructions, at any time to read how to complete the form.

Instructions is a button element. Send is a submit element. Clear is a reset element.
Your age is a radio element. Only one can be selected.
Courses taken is a checkbox element. More than one can be selected.

Button

Instructions Click at any time

Text Elements

First name: Bill Last name: Bates Student number: ******

Radio Buttons

Your age:
○ (a) under 20 ○ (b) 20-29
○ (c) 30-39 ● (d) 40-49
○ (e) 50-110 ○ (f) over 110

Checkboxes

Courses taken:
□ COMP 228 □ COMP 229
□ COMP 238 □ COMP 239
□ COMP 248 □ COMP 249

Submit and Reset Buttons

Send Clear

Menu

Microsoft Internet Explorer

Before submitting the form, do you wish to select one or more checkboxes for courses taken?

OK Cancel

Figure 10.9 Radio web page: radio.htm

Click the link, radio.htm, in the menu.
Click the button, Instructions, and read the instructions.
Note that there are no radio or checkbox elements chosen by default. Click a number of different radio elements and note that only one radio button can be selected at a

time. Click a number of checkbox elements and note that any number from zero to all boxes can be selected at a time.

Figure 10.9 shows the display of the web page, radio.htm, after data is entered in all elements except the checkboxes and the button, Send, is clicked.

10.9.9 Form - select (element); onchange (event)

Element

select: The select element is a list of options from which the user can choose. If the attribute, multiple, is not present, only one option may be selected at a time and the select element has radio button behaviour. If the attribute, multiple, is present, more than one option may selected at a time and the select item has checkbox behaviour. Although the select element has the same behaviour as radio or checkbox elements, it is displayed differently than a radio or checkbox element. The display of a select element is more compact than a radio or checkbox element and so is preferable when there are a relatively large number of options. The options can be displayed in two ways: as a list box or as a drop-down menu (sometimes called a combo box). The list box can display all of the options at the same time or part of the options and a scroll bar to access the rest of the options. The drop-down menu displays only one of the options and the others are displayed by clicking on the menu. The drop-down menu saves the most space on a form.

In the demonstration web page, the select element is a drop-down menu with mutually exclusive options, places of birth, and so it has the same behaviour as a radio element; only one choice can be made by the user who is completing the form.

Event

onchange: The change event occurs when the left mouse button is clicked on an option which is not selected or, in the case when the multiple attribute is present, when an option is deselected. An event handler is specified by the attribute, onchange, and is executed after the change event.

The element and event described above are demonstrated by the web page, select.htm.

To receive an email, change the action attribute to your personal email address in the <form> tag.

Web page 10.9 select.htm

```
<html>
<head>
    <title>Forms and JavaScript</title>
<script type="text/javascript">
/**********************************
**********************************
*Instructions onclick event function*
**********************************
*********************************/
function displayInstructions()
{
var str = "\
****Text Elements****\n\
Complete the text elements at any time. This data is required.\n\
The data in the text elements is not validated until\n\
    after the onclick event for the submit button.\n\
Then each text element is tested if it is empty (a zero-length
string)\n\
    and if it is not empty it is considered to be valid.\n\n";
str += "\
****Radio Buttons****\n\
Choose a radio button. This is required.\n\
The data is validated after the onclick event for the submit
button.\n\n";
str += "\
****Checkboxes****\n\
Select one or more checkboxes. This is optional.\n\
If no checkboxes are selected and the form is complete,\n\
    after the onclick event for the submit button, a dialog box
asks\n\
    if the user wishes to make a selection before the form is
submitted.\n\n";
str += "\
****Select List****\n\
Select one option. This is required.\n\
The data is validated after the onclick event for the submit
button.\n\n";
str += "\
****Submit and Reset Buttons****\n\
Click either button at any time.\n\
Submit\n\
    If the form is not complete, a dialog box gives instructions
to complete the form.\n\
    If the form is complete, a dialog box asks if the user wishes
to submit the form.\n\
Reset\n\
    A dialog box asks if the user wishes to clear the form.";
alert(str);
}
/*******************************
*******************************
*Select onchange event function*
*******************************
******************************/
function displaySelect()
/*
* display in an alert box the option which is chosen
*/
{
var f = document.emailForm;
```

```
if (f.birthplace.value != "")
    alert("Option chosen: " + f.birthplace.value);
else
    alert("Option chosen: none");
}
/******************************
******************************
*Submit onclick event function*
******************************
******************************/
function vldt_submit()
/*
* Validate user input to the form.
* If the input is incomplete,
* instruct the user to compete the form.
* If the input is complete,
* ask if form is to be submitted now.
*/
{
//declare local variables
var f = document.emailForm;
var empty, msg;
var i;
var radioChecked, checkboxChecked, birthplaceSelected;
var retval;
//validate text elements
empty = "";
if (f.firstname.value == "")
    empty += "\tFirst Name\n";
if (f.lastname.value == "")
    empty += "\tLast Name\n";
if (f.id.value == "")
    empty += "\tStudent Number\n";
//validate radio buttons
radioChecked = false;
for (var i = 0; i < f.age.length; i++)
    if (f.age[i].checked)
    {
        radioChecked = true;
        break;
    }
//validate checkboxes
checkboxChecked = false;
for (i = 0; i < f.courses.length; i++)
    if (f.courses[i].checked)
    {
        checkboxChecked = true;
        break;
    }
//validate select list
birthplaceSelected = false;
if (f.birthplace.value != "")
    birthplaceSelected = true;
//construct the message
msg = "";
if (empty != "")
  {
    msg = "Please enter data in the required text elements:\n";
    msg += empty;
  }
if (radioChecked == false)
    msg += "Please click the radio button for your age.\n";
if (birthplaceSelected == false)
    msg += "Please select your place of birth.\n";
//inform user of missing data
if (msg != "")
  {
    msg = "The form is incomplete.\n\n" + msg;
    alert (msg);
    return false;
  }
else if (checkboxChecked == false)
  {
    retval = confirm("Before submitting the form, do you wish to
select\n" +
          "one or more checkboxes for courses taken?");
    if (retval == true)
       return false;
  }
 /form is complete
//ask user for confirmation to submit the data to the server
retval = confirm("The form is complete.\n\nDo you wish to submit
the form now?");
return retval;
}
/******************************
******************************
*Reset onclick event function*
******************************
*****************************/
function vldt_reset()
/*
* Ask if form is to be reset now.
*/
```

```
{
var retval = confirm("If the form is cleared, all data entered in
the form will be removed.\n\n\
Do you wish to clear the form now?");
return retval;
}
</script>
</head>
<body>
<h1>Form - select; onchange</h1>
<h3>
<u><b>Click, Instructions, at any time to read how to complete
the form.</b></u>
</h3>
Instructions is a button element.
Send is a submit element.
Clear is a reset element.<br>
Your age is a radio element. Only one can be selected.<br>
Courses taken is a checkbox element. More than one can be
selected.<br>
Place of birth is a select element.
<p>
<form name="emailForm" method=post enctype="text/plain"
      action="mailto:myname@cs.concordia.ca">
<!--replace the email address, after mailto:, by your email
address-->
<!-- Button -->
<fieldset>
<legend> <font color=black><b>Button</b></font> </legend>
<input type=button value="Instructions" name="instructionButton"
onclick="displayInstructions()"> Click at any time
</fieldset>
<p>
<!-- Text Elements -->
<fieldset>
<legend> <font color=blue><b>Text Elements</b></font> </legend>
First name: <input type="text" name="firstname">
Last name: <input type="text" name="lastname">
Student number: <input type="password" name="id" size=10
maxlength=10>
</fieldset>
<p>
<table cellspacing=10>
<tr>
<td width=300>
<!-- Radio Buttons -->
<fieldset>
```

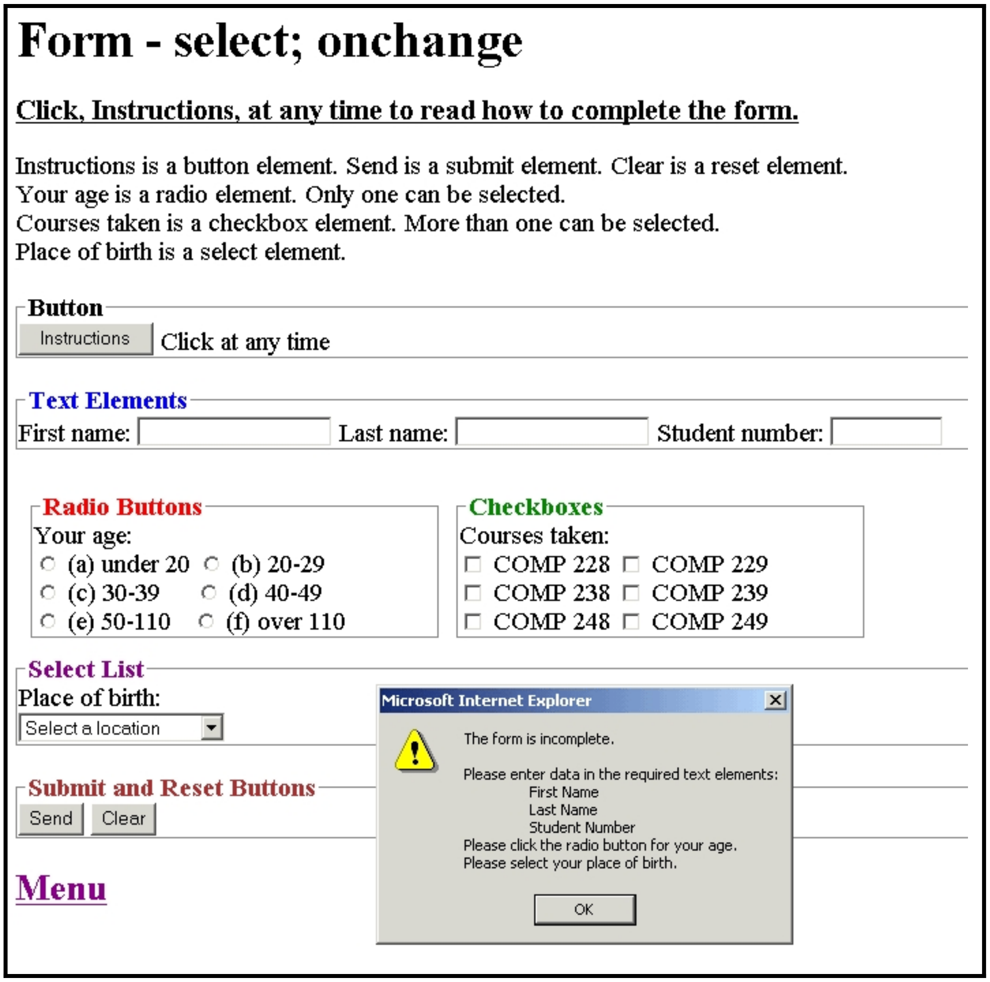
```
<legend> <font color=red><b>Radio Buttons</b></font> </legend>
Your age:
<br><input type="radio" name="age" value="age&lt;20"> (a) under
20
<input type="radio" name="age" value="age20-29"> (b) 20-29
<br><input type="radio" name="age" value="age30-39"> (c) 30-39
&nbsp;&nbsp;&nbsp;&nbsp;<input type="radio" name="age"
value="age40-49"> (d) 40-49
<br><input type="radio" name="age" value="age50-110"> (e) 50-110
&nbsp;&nbsp;<input type="radio" name="age" value="age&gt;110">
(f) over 110
</fieldset>
</td>
<td width=300>
<!-- Checkboxes -->
<fieldset>
<legend> <font color=green><b>Checkboxes</b></font> </legend>
Courses taken:
<br><input type="checkbox" name="courses" value="c228"> COMP 228
<input type="checkbox" name="courses" value="c229"> COMP 229
<br><input type="checkbox" name="courses" value="c238"> COMP 238
<input type="checkbox" name="courses" value="c239"> COMP 239
<br><input type="checkbox" name="courses" value="c248"> COMP 248
<input type="checkbox" name="courses" value="c248"> COMP 249
</fieldset>
</td>
</tr>
</table>
<!-- Select List -->
<fieldset>
<legend> <font color=purple><b>Select List</b></font> </legend>
Place of birth:
<br><select size=1 name="birthplace" onchange="displaySelect()">
<option>Select a location</option>
<option value="BC">British Columbia</option>
<option value="AB">Alberta</option>
<option value="SK">Saskatchewan</option>
<option value="MB">Manitoba</option>
<option value="ON">Ontario</option>
<option value="QC">Quebec</option>
<option value="NB">New Brunswick</option>
<option value="NS">Nova Scotia</option>
<option value="NF">Newfoundland</option>
<option value="PE">Prince Edward Island</option>
<option value="US">United States</option>
<option value="EL">Elsewhere</option>
</select>
```

```
</fieldset>
<p>
<!-- Submit and Reset Buttons -->
<fieldset>
<legend> <font color=brown><b>Submit and Reset Buttons</b></font>
</legend>
<input type="submit" value="Send" onclick="return vldt_submit()">
<input type="reset" value="Clear" onclick="return vldt_reset()">
<br></fieldset>
</form>
<h2>
<a href="menujs.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

Click the link, select .htm, in the menu.
Click the button, Instructions, and read the instructions.

Note that the first line in Select List is a message (Select a location), not an option. To observe when a change occurs, click a place, say Quebec, click OK in the dialog box, then click Quebec again, then click Ontario and so on.

Figure 10.10 shows the display of the web page, select.htm, when no data is entered in the elements and the button, Send, is clicked.



Form - select; onchange

Click, Instructions, at any time to read how to complete the form.

Instructions is a button element. Send is a submit element. Clear is a reset element.
Your age is a radio element. Only one can be selected.
Courses taken is a checkbox element. More than one can be selected.
Place of birth is a select element.

Button
Instructions Click at any time

Text Elements
First name: Last name: Student number:

Radio Buttons
Your age:
○ (a) under 20 ○ (b) 20-29
○ (c) 30-39 ○ (d) 40-49
○ (e) 50-110 ○ (f) over 110

Checkboxes
Courses taken:
□ COMP 228 □ COMP 229
□ COMP 238 □ COMP 239
□ COMP 248 □ COMP 249

Select List
Place of birth:
Select a location

Submit and Reset Buttons
Send Clear

Menu

Microsoft Internet Explorer

The form is incomplete.

Please enter data in the required text elements:
First Name
Last Name
Student Number
Please click the radio button for your age.
Please select your place of birth.

OK

Figure 10.10 Select web page: select.htm

.

11 State

11.1 Introduction

State is memory retained between HTTP transactions. HTTP (hypertext transfer protocol) is the set of rules used on the world wide web by clients and servers to communicate with each other. An HTTP transaction consists of two parts:

(1) the request for a web page by the client sent to the server and

(2) the response by the server sent to the client which includes the web page which was requested

Figure 11.1 shows an HTTP transaction.

After a transaction is complete, the server disconnects from the client and removes all information about the transaction. There is no provision by HTTP to store information from a transaction after the transaction ends. This circumstance is described by saying that HTTP has no state or is stateless. In this context state means memory, that is, the capability to store data.

Many applications require that state be maintained after a transaction ends. State maintenance means that data sent or generated in a transaction is stored and can be used in a later transaction. A common example is a shopping cart application which has several web pages and requires a user to access more than one of the web pages. The same user makes more than one transaction and the user must be identified as the same user for each transaction. Also the selections of items made by the user must be stored as the user accesses various web pages (makes various transactions).

In this chapter a simple application is used to illustrate methods to maintain state. The application is an adding machine which adds numbers typed on the keyboard. The user types a number in a form and then clicks the submit button of the form. The client sends the data to the server and the server adds the number to the previous total of the numbers already typed. This data is sent back to the client which displays the new total, as well as the number of numbers which have been added. To start a new sum, the letter, s, is typed instead of a number.

It is useful to contrast a program running on a computer which adds numbers and a web application that adds numbers where in both cases the user types the numbers to be added on the keyboard. The program automatically has state since data that the user supplies to the program is stored in variables in the program. The variables are always available until the program is terminated by the user. In a web page of an application, elements of a form are used to store data supplied by the user. When the same web page is requested later from the server, the web page has no memory of the data that was entered by the user on the form the previous time. The web page is essentially a program that runs and terminates each time it is requested.

.

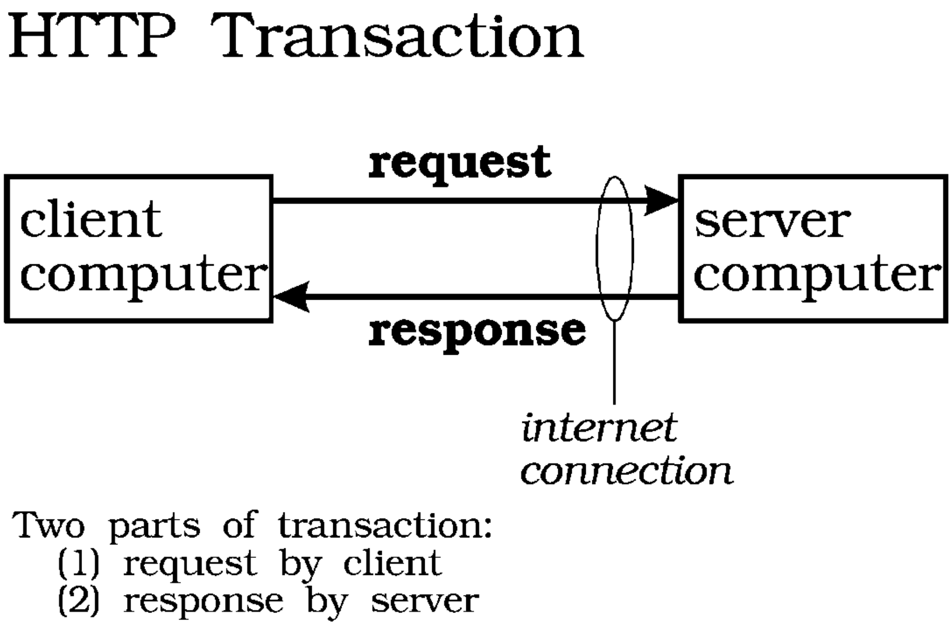


Figure 11.1 HTTP Transaction

11.2 Methods to maintain state

State maintenance means to effectively give web pages a memory by sending information from one web page to another web page. There are four methods to maintain state. They are summarized in Table 11.1

Table 11.1 State Maintenance

Method	Location of State Information	Brief Syntax
hidden variable	element in a form	<input type="hidden" name= ... value= ...>
cookie	file on client computer	setcookie(name, ...)
session	file on server computer	session_start()
URL rewriting	appendant to URL	<a href="URL"?namevaluepair1 &namevaluepair2 ...>

11.2.1 Hidden variable

A hidden variable is a variable stored in a form. It is not displayed by the browser. The value of a hidden variable cannot be changed by either the browser or the user. Only by a script can change the value of a hidden variable.

11.2.2 Cookie

A cookie is a small file that is stored on the hard disk of the client computer. The file is used to store variables.

11.2.3 Session

A session is the period of time a connection is made by a user to a web site. During the session, session files are stored on the hard disk of the server computer. The session files are used to store variables.

11.2.4 URL Rewriting

URL rewriting is the appending of variables to the URL which is a link in a web page. URL (uniform resource locator) is the address of a web page. The components of a URL, excluding the appendants (appended variables), are described in the chapter, HTML.

11.3 Hidden variable

11.3.1 Definition

A hidden variable is a form variable with the type, hidden. It is not displayed on the form but its value is sent to the server along with the values of all the other variables on the form when the submit button is clicked.

11.3.2 Declaration

The value of the variable can be static (a constant) or dynamic.
Declaration of static hidden variable in a form:

```
<input type="hidden" name="var_name" value="var_value">
```

Declaration of dynamic hidden variable in a form:

```
<input type="hidden" name="var_name" value="<?php echo
$var_name ?>">
```

In the case of dynamic variables, it is assumed that the variable name in the form, var_name, is the same as the variable name in the script and so in the script the variable, var_name, has value, $var_name. A variable name used in the web pages in this chapter is, sum, and so the value of the variable is, $sum. Initially, $sum is 0 and is used to store the total of a series of numbers.

A hidden variable that is used to maintain state is dynamic. Before the web page is sent by the server to the browser, a script can change the value of the hidden variable and then put the new value in the form using echo, as shown above.

11.3.3 Storage

The variables of a form, including the hidden variables are stored in the $_POST array if in the form tag, method=POST, or the $_GET array if in the form tag, method=GET. Usually the method, POST, is used.

11.4 Cookie

There are two versions of each web page, except the menu: with and without the source code displayed. The name of the web page with the source code displayed is the name of the web page without the source code followed by a hyphen and the letters, withcode. Also, the web page with the source code is displayed in an interactive web page, which is explained below. So, each demonstration is has three links in the menu. For the first demonstration, First JavaScript Page, the links are hello.htm, hello-withcode.htm and interactive.php. Note that the names of the links are the same as the names of the web pages.

11.4.1 Definition

There are two definitions of a cookie.
Definition 1
A cookie is a small, flat, sweetened cake, often round, made from stiff dough baked on a large, flat pan. This is the definition of cookie in culinary science.
Definition 2
A cookie is data that a web server sends to a browser (on a client computer) and which the browser stores temporarily in main memory of the client computer or permanently in a text file on the hard disk of the client computer. This is the definition of cookie in computer science.

11.4.2 History of cookie

The cookie was introduced by Netscape in 1994, the same year that Netscape was founded. The technical name originally used by Netscape was, persistent client-side state information. Netscape documentation states that the name, cookie, is chosen for no compelling reason. This is not too plausible and one popular speculation is that cookie is named after the Chinese fortune cookie, which contains a piece of paper with a small amount of text on the paper.

However, the inventor of the cookie, Lou Montulli, says that the name comes from the UNIX term, magic cookie, which is defined in the New Hackers Dictionary as: something passed between routines or programs that enables the receiver to perform some operation.

11.4.3 Analogy to cookie

The dry-cleaner analogy, which follows, is found on some web sites about cookies.
There are two transactions. You, a client, take a shirt to be cleaned at the dry-cleaner, a server. The dry-cleaner (server) gives you (client) a claim ticket, which corresponds to a cookie. This is the end of the first transaction. When you (client) return to the dry-cleaner (server) to pick up the shirt, you bring the claim ticket (cookie) with you and you give the dry-cleaner (server) your claim ticket (cookie). The dry-cleaner (server) uses the information on the claim ticket (cookie) to find your clean shirt and return it to you (client). This is the end of the second transaction.

The claim ticket (cookie) relates the two transactions between the client and server. Without the cookie, the second transaction could not be completed. In the analogy above, without the claim ticket (cookie) the server (dry-cleaner) could not find the shirt (without other information from you).

11.4.4 setcookie()

The function, setcookie(), creates a cookie variable. The function has several arguments which are defined below. The call to setcookie() must be made in a web page before any HTML code, because values of the cookie variables are sent in the header to the browser.

The name and value of cookie variables are stored in the array, $_COOKIE.

The declaration of setcookie() is

```
   boolean setcookie(name[, value[, expire[, path[, domain[,
   secure]]]]])
```

The function, setcookie(), returns TRUE if the function runs successfully, FALSE otherwise.
The function has six arguments. All arguments, except the first, are optional as indicated by square brackets. String arguments are skipped by an empty string, “”. Integer arguments are skipped by zero, 0.

name: a string, name of a variable
value: a string, value of the variable
expire: an integer, expiration time of the cookie, in seconds after the beginning of 1970
If the expiration time of a cookie is not specified, the browser saves the cookie only in main memory. In this case when the browser closes, the cookie disappears.
If a value of argument, expire, is given, the cookie is stored on the hard disk of the client for a period of time specified by expire. For example, if the argument is, time() + 86400, the cookie is stored on the hard disk for one day (86400 seconds = 1 day), that is, the cookie expires in one day. The function, time(), returns the current time in number of seconds after the beginning of 1970.
path: a string, directory (and any subdirectories of the directory) containing web pages for which the cookie is valid
domain: a string, name of the server for which the cookie is valid
If not specified, the default is the name of the server that generated the cookie.
secure: an integer, 1 if the cookie is to be transmitted by HTTPS, that is, a secure web server; not 1 if the cookie is to be transmitted by HTTP

11.4.5 Storage of cookie

1. Client computer hard disk
Cookies are stored on the client hard disk, if the argument, expire, is specified in the function, setcookie(). These are sometimes called persistent cookies.
Otherwise, cookies are stored only in main memory of the client and the cookie disappears when the browser terminates. These are sometimes called non-persistent cookies.
2. Server computer hard disk
There is no storage of cookie data or other data on the server hard disk.

A cookie is sent back and forth between the server and browser as follows.

Abbreviations: b -> s stands for browser to server
s -> b stands for server to browser

Requests are made by the browser either by typing a URL in the address bar or by clicking on a link.

1. b -> s: first request for a web page that is programmed (contains a script) to send a cookie
   The browser sends the URL of the web page to the server. The server retrieves the web page from the server hard disk and executes the script on the web page.
2. s -> b: response to first request
   The server sends the web page and a cookie to the browser.
   The browser stores the cookie in a text file on the hard disk of the client computer, if argument, expire, is specified.
3. b -> s: second request for the same web page or other pages from the same web site
   The browser sends the cookie back to the server with the URL of the web page. The server retrieves the web page from its hard disk and executes the script on the web page.
4. s -> b: response to second request
   The server sends the web page and the same cookie to the browser since the content of the cookie may have changed.
   The browser stores the cookie in the same text file as before, replacing the previous contents of the file.
5. b -> s: third request . . . (and so on) . . .

11.4.6 Location of cookie

The cookies are stored in the client computer and a typical location for Firefox on a Linux system is: ~/.mozilla/firefox/<profile path>/cookies.sqlite. The cookies are stored in a table named moz_cookies and can be accessed directly using sqlite. Cookies are stored on disk only if the expiration time is set by setcookie(). If expiration time is not set, the cookies are stored only in main memory and disappear when the browser is closed. A cookie is machine-specific, so the user cannot use another computer and still maintain state with a cookie.

To manage the cookies a facility provided by most browsers could be used. In Firefox it is Edit>Preference>Privacy & Security or by using the menu button and choosing Preference followed by choosing Privacy & Security panel and go to the Cookies and Site Data section and select one of: Clear Data/Manage Data/Manage Permission. The amount of storage for all cookies could add up to several hundred kilobytes.

11.4.7 Information in a cookie

Using a cookie, the server stores data on and retrieves the data from the client computer as described above. The exchange of cookies must be done with the cooperation of the client, since it is possible for the browser to block cookies sent by a server. For example, in Internet Explorer 6 the user can choose to accept all cookies, accept some cookies or block all cookies, from menu item, Tools/Internet Options/Privacy.

The data stored in the cookie consists of name-value pairs and the name of the web site. Also, the name of the web site is part of the name of the cookie (name of the file).

The cookie (information stored by the web site on a user's computer) is sent back to the web server by the browser the next time that the browser requests a page from the web site.

The data in the cookie is stored in the array, $_COOKIE, when the cookie is received by the server.
Most web sites store only a user ID in the cookie, as well as the name of the web site. In that case, the user ID is stored in a database with information about the user, for example, how many times the user visited the site and user preferences. The database would be accessed using the user ID.

A cookie file is a text file created on the server, which will be a Linux text file instead of a DOS text file if the operating system of the server is Linux. The difference between the two types of text files is the newline character. DOS uses two characters to represent the newline and Linux uses only one character.

To display the contents of a cookie, use Windows Explorer to list the contents of the directory, Temporary Internet Files. Double click on a cookie file. A dialog box appears which gives a warning and asks, “Do you wish to continue?”. Click, Yes. Because the cookie file has the extension, .txt, Notepad opens and displays the contents of the cookie. All of the data is on one line with an unprintable character indicated by a rectangle. This character is a newline which Notepad does not understand since Notepad expects a DOS text file, but the file is a Linux text file. Save the file and then open it with WordPad which does recognize the Linux newline character.

11.4.8 Limitations

The maximum size of a cookie which can be stored is 4 kB.
A client computer can store a maximum of 300 cookies on the hard disk.
A client computer can store a maximum of 20 cookies for a particular server.

11.5 Session

11.5.1 Definition

A session is the period of time during which data is stored on the server hard disk for a particular web site and user, and is accessible when repeated visits are made to the web site by the same user.

11.5.2 session_start()

The function, session_start(), is used to begin or continue a session. This function has no arguments.
The call to session_start() must be made in a web page before any HTML code, because a cookie which contains the session ID is sent in the header to the browser.

After session_start() is called, values of variables can be assigned to the session array, $_SESSION. For example, the variable, count, can be given a value and then assigned to the session array as follows.

```
session_start();
$count = 0;
$_SESSION[count] = $count;
```

Variables were stored in a session by the function, session_register(), instead of assigning variables to the $_SESSION array, but this method is now deprecated (its use is discouraged). The process of storing variables in a session is called registering variables because of the name of this function.

11.5.3 Storage of session data

1. Client computer
    Cookie-based
        A cookie is stored which contains only the session ID.
        The storage is either in main memory or on the hard disk.
        The name of the cookie file includes the URL of the server (web site).
    URL-based
        No data is stored.
        The session ID is embedded in the URL by adding it to the end of the URL.
2. Server computer
    The name and value of session variables are stored on the hard disk in the session file.
    The name of the session file includes the session ID.

The cookie and session data is sent back and forth between the server and browser as follows.
Abbreviations: b -> s stands for browser to server
s -> b stands for server to browser
Requests are made by the browser either by typing a URL in the address bar or by clicking on a link.

1. b -> s: first request for a web page that is programmed (contains a script) to start a session
    The browser sends the URL of the web page to the server. The server retrieves the web page from the server hard disk and executes the script on the web page. The server stores session data (variables which consist of name-value pairs) on the server hard disk in a session file whose name includes a session ID.
2. s -> b: response to first request
    The server sends the web page and a cookie to the browser.
    The browser stores the cookie in a text file on the hard disk of the client computer.
    The cookie contains one variable, the session ID, followed by the URL of the web site
    The default name of the variable in the cookie is PHPSESSID (which stands for PHP session ID) and the value (session ID) is a 32-digit hexadecimal number, which is shorthand for a 128-bit binary number.
3. b -> s: second request for the same web page or other pages from the same web site
    The browser sends the cookie (which identifies the session) back to the server.
    Using the session ID in the cookie, the server reads the session data from the session file on the server hard disk into an array of session variables, $_SESSION. The server retrieves the web page from the server hard disk and executes the script on the web page. After executing the PHP script, the server writes the session data back to the hard disk, in case it has changed.
4. s -> b: response to second request
    The server sends the web page and the same cookie to the browser in case the session ID has been changed (the function, session_id(), changes the session ID).
    The browser stores the cookie in the same text file as before, replacing the previous contents of the file.

5. b -> s: third request . . . (and so on) . . .

11.5.4 Location of storage

Session cookie

The cookie is stored in a file on the client hard disk in the same directory as cookies without a session.

Session data

The session data is stored in a file on the server hard disk in the default directory, /tmp, if the operating system is Linux or UNIX. If the operating system is Windows, the directory could be C:\temp, for example, or some other directory.

11.5.5 Information in a session

The session cookie is stored in a file on the client hard disk in the directory, Temporary Internet Files.
The contents of the cookie can be displayed as described in the previous section, Cookies.
For example the first two lines of the cookie could be

```
PHPSESSID
739b5ef38af853f51744cc6a57c2e650
```

The first line is the name of variable, PHPSESSID (which stands for PHP session ID), and the second line is the value of the variable which is a 32-digit hexadecimal number. The value is the session ID assigned by the server.

The session data is stored on the server hard disk in the directory, /tmp. The name of the session file starts with the characters, sess_, followed by the session ID. So for the session with the name session ID above, the name of the file is, sess_739b5ef38af853f51744cc6a57c2e650. The list of the session files can be seen by logging in to the server, changing the directory to /tmp and then giving the command, ls. The contents of the session files cannot be displayed, for example by the command, more; only the superuser can open the session files.

The session data consists of name-value pairs stored on the server hard disk in the session file.
The session data is read from the server hard disk and stored in the array, $_SESSION.

11.6 URL Rewriting

11.6.1 Definition

URL rewriting appends variables to the URL in a link. When the link is clicked, the variables are propagated to the server and stored in the array, $_GET. The URL variables are visible in the address bar of the browser. The number of characters which can be appended to the URL is limited.

11.6.2 Format

Variables are appended as name-value pairs. A question mark, ?, separates the URL value from the first name-value pair. If there is more than one name-value pair, an ampersand, &, separates the name-value pairs. The ampersand has a special meaning in a web page: it inserts characters in a web page. Its special meaning must be removed by replacing it with its escape sequence, either &#38; or &amp;.

11.6.3 Declaration

Declaration of URL variables in a link can be static or dynamic.
Two variables (name-value pairs) are shown in the examples below. The ampersand, &, is replaced by &amp;.
Declaration of static variables in a link:

```
   <a href="URL"?var_name1="var_value1"&amp;
      var_name2="var_value2">Send webpage1</a>
```

Declaration of dynamic variables in a link:

```
   <a href="URL"?var_name1="<?php echo $var_name1 ?>"&amp;
      var_name2="<?php echo $var_name2 ?>">Send webpage1</a>
```

In the case of dynamic variables, it is assumed that the variable name in the form, var_name1, is the same as the variable name in the script and so in the script the variable, var_name1, has value, $var_name1, and similarly for var_name2.

11.7 Programming Demonstrations

The application, an adding machine, which is described in the introduction is implemented by a C program and three different web pages each using a different method to maintain state: hidden variables, cookies and sessions. The other method to maintain state, URL rewriting, is not demonstrated because it cannot be used for this application, since in this application a link is not used to request a web page. URL rewriting is used to maintain state in the chapter, Applications.

Each web application uses one web page which is self-referencing, which means that the web page requests the same web page (the web page requests itself) when the form is submitted. One part of the web page is a form to enter a number to be added and the other part of the web page validates the data and adds the number to the total of the previous numbers. The number of numbers which were added and the sum of the numbers is displayed. To start a new sum, the letter, s, is typed instead of a number.

The value of the sum is formatted by the PHP function, sprintf(), with two digits after the decimal point.

Data is validated by the PHP function, is_numeric(*value*). If *value* is a number, the function returns TRUE; otherwise the function returns FALSE. In each application, enter nonnumeric data to test the validation, for example, 1..2, 1.2.3, 1-, 1-2, - 3 (space after -), 2a and no data. Note that any number starting with - (minus) and without a space after the - (minus) is a negative number.

The web pages must be stored on a server computer and displayed on a client computer with a browser. It is assumed that programmer remotely logs in to the server. The instructions which follow are for the case when the operating system on the server is Linux. In the document home directory on the server, create the directory, c11state, and then set the permissions of the directory to 707 (write permission is required because the interactive web page writes files on the server).

Store the web pages which follow in the directory, c11state. Set the permission of each web page to 604. The third digit of the permissions, 4, allows anyone to read (display) the web page. Use a text editor, emacs for example, to type and edit the web pages, or copy and paste the web pages.

The following commands are used.

```
        pwd                          (display working directory)
        mkdir c11state               (create directory,  c11state)
        chmod 707 c11state           (change permissions of directory,  c11state)
        cd c11state                  (change working directory to  c11state)
```

11.7.1 Menu

The web page, menustate.htm, is a menu which gives the links to the demonstration web pages in this chapter.

Start a text editor, emacs in Linux for example. The command to start emacs and open a file with name, menustate.htm, is

```
        emacs menustate.htm          (open file, menustate.htm, in emacs)
```

Type the following file, or copy and paste it, and then save it in directory, c11state. After, exit emacs.

Set the permissions for the menu web page to 604.

```
        chmod 604 menustate.htm      (change permissions of file, menustate.htm)
```

Web Page 11.1 menustate.htm

```
<html>
<head>
<title>Menu for State</title>
</head>
<body>
<h2>
<a href="../index.html">Main Menu</a>
</h2>
<h1>Menu for State</h1>
<table width=100% border=0><tr>
<td width=30%>
<h3>
11.1 Hidden Variable
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a href="addm_hid.php">addm_hid.php</a>
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a href="addm_hidwithcode.php">addm_hidwithcode.php</a>
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a
href="interactive.php?fn=addm_hid.php">interactive.php</a><br><br
>
```

```
11.2 Cookie
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a href="addm_coo.php">addm_coo.php</a>
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a href="addm_coowithcode.php">addm_coowithcode.php</a>
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a
href="interactive.php?fn=addm_coo.php">interactive.php</a><br><br
>
11.3 Session
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a href="addm_ses.php">addm_ses.php</a>
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a href="addm_seswithcode.php">addm_seswithcode.php</a>
<br>&nbsp;&nbsp;&nbsp;&nbsp;
<a
href="interactive.php?fn=addm_ses.php">interactive.php</a><br><br
>
</h3>
</td>
<td width=70% height=100% valign=top>
<iframe src=statemainten.htm width=100% height=100%
frameborder=0></iframe>
</td></tr></table>
</body>
</html>
```

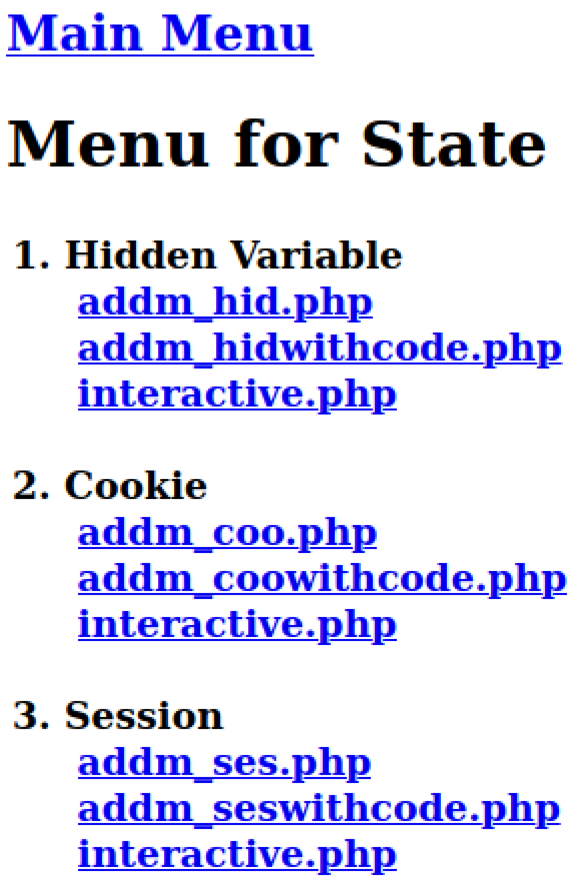
Figure 11.2 shows the menu as displayed by the browser. The web page with the menu also includes Table 11.1 State Maintenance which is not shown in the figure.

There are two versions of each web page, except the menu: with and without the source code displayed. The name of the web page with the source code displayed is the name of the web page without the source code followed by the letters, withcode. Also, the web page with the source code is displayed in an interactive web page, which is explained below. So, each demonstration has three links in the menu. Note that the names of the links are the same as the names of the web pages.

The web page, interactive.php, for each demonstration shows the source code on the left side of the page and the output of the source code on the right side. The source code can be changed by typing on the left side and then Submit can be clicked to see the output of the changed source code on the right side. The original source code and its output can be displayed by clicking, original page.

11.7.2 C program

The C program, addmach.c, adds numbers and displays the number of numbers in the sum and the sum of the numbers after each number is added. The C program has state (memory) since the data is stored in variables in the program and the program continues to run until terminated by the user. The operation of the program is explained by comments in the source code.



Main Menu

Menu for State

1. Hidden Variable
   addm_hid.php
   addm_hidwithcode.php
   interactive.php

2. Cookie
   addm_coo.php
   addm_coowithcode.php
   interactive.php

3. Session
   addm_ses.php
   addm_seswithcode.php
   interactive.php

Figure 11.2 Menu for web pages: menustate.htm

Start a text editor, emacs for example, and type the following file, or copy and paste it. Save the file in directory, c11state, using the name, addmach.c,. This is the source file. Compile the source file to produce the executable file using the gcc compiler in Linux. Give the name, addmach, to the executable file. The Linux command is

```
gcc addmach.c -o addmach
```

The program is executed by typing the name of the executable file at the Linux prompt,

```
addmach
```

Cprogram 11.1 addmach.c

```
/*
addmach.c
2005 March 24
Updated: August 2019
Description
   An adding machine which adds numbers
```

```
  typed on the keyboard and displays the sum
Program steps:
1.  Message output to screen (prompt)
2.  Data input from keyboard
3.  Data validation
4.  Data output to screen
5.  Do step 1 or terminate program
*/
#include <stdio.h>              /*printf, scanf*/
#include <stdlib.h>             /*strtod*/
#include <errno.h>              /*errno*/
main()
{
double sum = 0;
double num;
int count = 0;
char str[20];
char *tail;
while(1)
{
   /*1. MESSAGE OUTPUT TO SCREEN
        program ---> user*/
   /*prompt for next sum*/
    printf("\n*****Adding Machine*****\n");
    printf("(1) Type a number or\n(2) type s to start a new sum 
or\n");
    printf("(3) type x to exit the program\n\tand then press 
Enter\n\n");
    while(1)
    {
        /*prompt for next number*/
        printf("Number: ");
  /*2. DATA INPUT FROM KEYBOARD
       user ---> program*/
  scanf("%s", str);
  /*errno contains a nonzero integer after an error*/
  errno = 0;
  /*convert string to double*/
  /*the function, strtod, reads the string until the first 
character
     that is not part of a number.
     Examples: 12a is converted to 12, 12.1.2 is converted to 
12.1*/
  num = strtod(str, &tail);
  /*3. DATA VALIDATION*/
  if ((num != 0) || ((num == 0) && (errno == 0) && (*tail == 
0)))
```

```
   /*valid number: nonzero or zero*/
   /*4. DATA OUTPUT TO SCREEN
        program ---> user*/
   {
     sum += num;
     count++;
     printf("\t\tsum of %d numbers = %.2f\n", count, sum);
     continue;
   }
   else if (strcmp(str, "s") == 0)
   /*new sum*/
   {
     printf("\nStart a new sum\n");
     sum = 0;
     count = 0;
     break;
   }
   else if (strcmp(str, "x") == 0)
   /*exit program*/
   {
     printf("Program is terminated\n");
     exit(0);
   }
   else
   /*invalid number*/
     printf("\tNumber is invalid: %s\n", str);
    } /*end of inner while*/
}  /*end of outer while*/
}
```

Figure 11.3 shows the output of the C program, addmach, when three numbers are typed on the keyboard and then s is typed to start a new sum.

11.7.3 Hidden variable

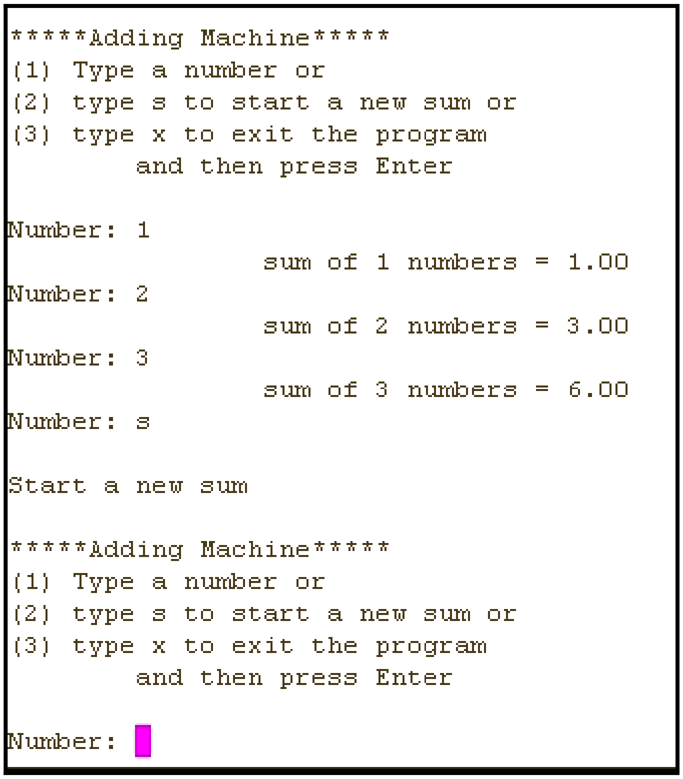
The web page, addm_hid.php, implements the adding machine application by using hidden variables to maintain state. The application adds numbers that are typed in a form. The number of numbers in the sum and the sum of the numbers after each number is added are displayed.

Start a text editor, emacs for example, and type the following file, or copy and paste it. Save the file in directory, c11state, using the name, addm_hid.php. Then change the permissions of the file to 604.

Web Page 11.2 addm_hid.php

```
<!--
addm_hid.php
2005 March 24
Updated: August 2019
```

```
Description:
   An adding machine which adds numbers typed on the keyboard
       and displays the sum on the screen
   State is maintained by hidden variables in forms
   The values of the hidden variables are set by the server
running
       a php script before the page is sent to the browser,
       except for the first time the page is sent
Steps:
server
```



```
*****Adding Machine*****
(1)  Type a number or
(2)  type s to start a new sum or
(3)  type x to exit the program
        and then press Enter

Number: 1
                sum of 1 numbers = 1.00
Number: 2
                sum of 2 numbers = 3.00
Number: 3
                sum of 3 numbers = 6.00
Number: s

Start a new sum

*****Adding Machine*****
(1)  Type a number or
(2)  type s to start a new sum or
(3)  type x to exit the program
        and then press Enter

Number: 
```

Figure 11.3 Output of C program: addmach

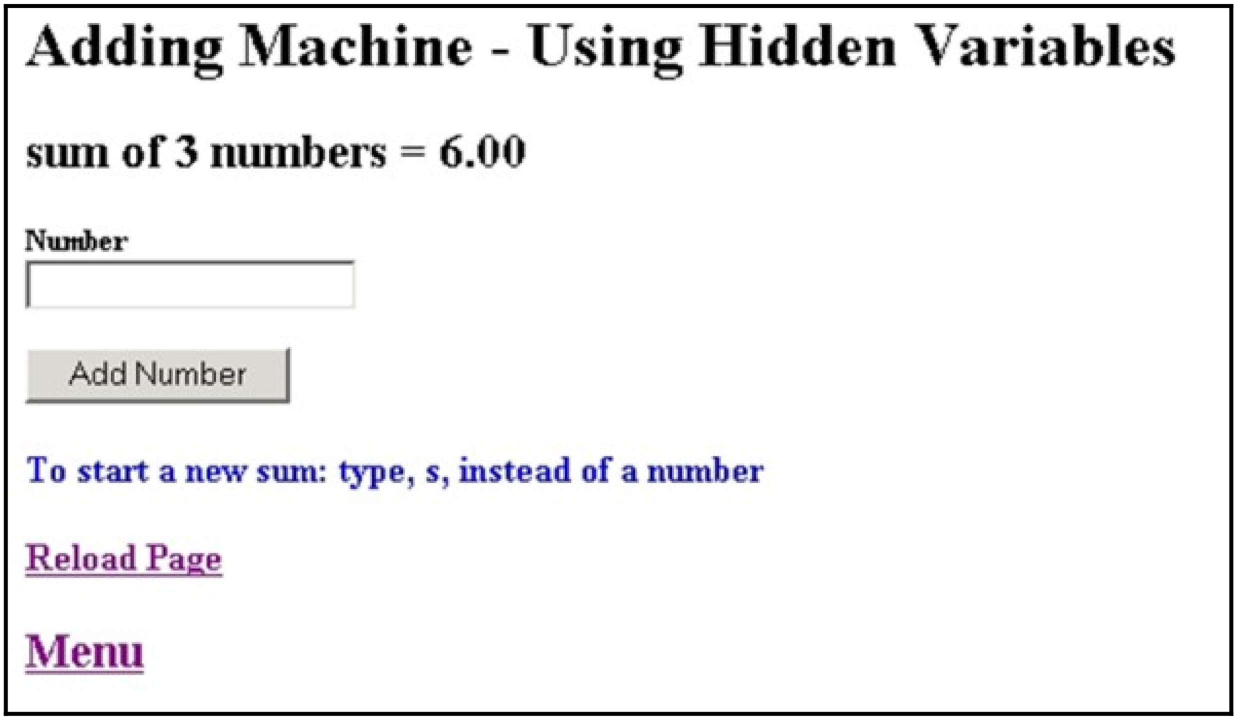
```
1.  Data assignment from form
2.  Data validation and calculation
browser
3.  Data output to screen
4.  Message output to screen by a form (corresponds to a prompt
in a C program)
5.  Data input from keyboard to the form
6.  Do step 1
-->
<?php
/*
1. ASSIGN DATA TYPED ON FORM TO VARIABLE
*/
```

```
//$number = $_POST['number'];   //data entry variable
if (isset($_POST['number']))
  $number = $_POST['number'];
else
  $number = NULL;
/*
2. DATA VALIDATION AND CALCULATION
*/
//set message variables
$msg1 = "";
$msg2 = "";
$msg3 = "";
if (is_null($number)) {
//first time page is sent to browser
  $sum = 0;
  $count = 0;
  $msg1 = "<h3><font color = green>First time that this page is 
sent to the browser</font></h3><br>";
}
else {
//not first time page is sent to browser
  //assign hidden variables to program variables
  $sum = $_POST['sum'];    //hidden variable
  $count = $_POST['count'];  //hidden variable
  if (is_numeric($number)){
    //valid number
    $sum = $sum + $number;
    $count++;
  }
  else if (! strcmp($number, "s")){
    //start new sum
    $count = 0;
    $sum = 0;
    $msg1 = "<h3><font color = blue>Start new 
sum</font></h3><br>";
  }
  else if ($number == "")
    //number not typed
    $msg1 = "<h2><u><font color = red>Type a 
number</font></u></h2>";
  else
    //invalid number
    $msg1 = "<h2><u><font color = red>Invalid number: 
$number</font></u></h2>";
}
/*
3. DATA OUTPUT TO SCREEN
```

```
*/
if (! is_null($number)) {
  //this is not executed when the page is sent to the browser for
the first time
  //display the sum
  $msg2 = sprintf("<h2>sum of %d numbers = %.2f</h2>", $count,
$sum);
  //display a message and a link
  $page = $_SERVER['PHP_SELF'];   //link to same page
  $msg3 = "
  <h3><font color=blue>To start a new sum: type, s, instead of a
number</font>
  <p><a href= $page >Reload Page</a></h3>
  ";
}
?>
<!--
4. MESSAGE OUTPUT TO SCREEN
    Create and display form
5. DATA INPUT FROM KEYBOARD
    Enter data on form
-->
<html>
<head><title>Adding Machine - Hidden Variable</title></head>
<body>
<h1>Adding Machine - Using Hidden Variables</h1>
<?php
    echo $msg1;
    echo $msg2;
?>
<!--
FORM: one text element, two hidden elements
-->
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
<strong>Number</strong><br>
<input type="text" name="number"><br>
<!-- Two Hidden variables -->
<input type="hidden" name="sum" value="<?php echo $sum ?>">
<input type="hidden" name="count" value="<?php echo $count ?
>"><br>
<input type="submit" value="Add Number"><br>
</form>
<?php
    echo $msg3;
?>
<h2>
<a href="menustate.htm" target=_top>Menu</a>
```

```
</h2>
</body>
</html>
```

Figure 11.4 shows the display of the web page, addm_hid.php, after the three numbers, 1, 2 and 3, are typed in the form.



Adding Machine - Using Hidden Variables

sum of 3 numbers = 6.00

Number

To start a new sum: type, s, instead of a number

Reload Page

Menu

Figure 11.4 Hidden variables web page: addm_hid.php

11.7.3.1 Explanation of hidden-variable web page, addm_hid.php

The PHP script in the hidden-variable web page starts by determining if the value of number in the form exists. This must be done using the function, isset(), which returns FALSE if number does not exist and TRUE otherwise. The value, number, does not exist the first time that the web page is requested, that is, when the link on the menu is clicked because the form is not submitted. In this case the variable, $number, is assigned the value, NULL. When the web page is requested after the first time, the value of number is sent from the form and so number exists. In this case the variable, $number, is assigned the value from the form, $_POST['number'].

The three message variables, $msg1, $msg2 and $msg3, are declared and assigned the value, "", which is the zero-length string.

The variable, $number, is tested for value, NULL, by the function, is_null().

(a) First time web page is requested

If $number is NULL, the web page is requested for the first time and so the variables, $count and $sum, are initialized to zero. The value of variable, $msg1, is set to, "First time that this page is sent to the browser".

When the web page is requested for the first time, none of the PHP code is executed after $msg1 is set until after the HTML code begins (tag <html>). Before the form, the two variables, $msg1 and $msg2, are displayed. The value of $msg1 is given above and the value of $msg2 is the zero-length string, that is, nothing is displayed for $msg2.

In the form, the action attribute is given the value, address of current web page, $_SERVER['PHP_SELF']. So when submit is clicked the same web page will be requested again. The hidden variables in the form are given values. Sum is given the value of $sum which is 0, and count is given the value of $count which is 0.

After the form, the variable, $msg3, is displayed, which is the zero-length string.
This completes the execution of the PHP code in the web page by the server. Now the server sends the web page to the browser and the browser displays the web page on the screen.

(b) Second and subsequent times web page is requested
The web page is requested after the first time by clicking Add Number on the screen, where Add Number is the name of the submit button on the form. The same web page, as given by the action attribute, is requested. The values of the hidden variables in the form, sum and count, and the value of the text variable, number, are sent to the server and stored in the $_POST array, since the method in the form is POST.

Now the value of number exists and so $number is assigned the value from the form,

```
$number = $_POST['number'];
```

As when the web was requested for the first time, the three message variables, $msg1, $msg2 and $msg3, are declared and assigned the value, "", which is the zero-length string.

Since $number is not NULL, the previous values of sum and count from the form are assigned to variables in the script, $sum and $count,

```
$sum = $_POST['sum'];          //hidden variable
$count = $_POST['count'];      //hidden variable
```

Then, if $number is a valid number, the $number which was typed in the form is added to $sum, and $count is incremented (1 is added to $count).

Before the beginning of the HTML code, values are assigned to variables, $msg2 and $msg3.
The variable, $msg2, is set to,
"sum of $count numbers = $sum".
The variable, $msg3 is set to,
"To start a new sum: type, s, instead of a number
Reload Page".

After the HTML code begins the same PHP code is executed as when the web page was requested for the first time. Then the server sends the web page to the browser and the browser displays the web page on the screen.

Different parts of the PHP code are executed if letter, s, is typed which starts a new sum, and when an invalid number or no number is typed in the form. The comments in the web page describe the operation in these cases.

11.7.4 Cookie

The web page, addm_coo.php, implements the adding machine application by using cookies to maintain state. The application adds numbers that are typed in a form. The number of numbers in the sum and the sum of the numbers after each number is added are displayed.

Start a text editor, emacs for example, and type the following file, or copy and paste it. Save the file in directory, c11state, using the name, addm_coo.php. Then change the permissions of the file to 604.

Web Page 11.3 addm_coo.php

```
<!--
addm_coo.php
2005 March 24
Updated: August 2019
Description:
   An adding machine which adds numbers typed on the keyboard
       and displays the sum on the screen
   State is maintained by cookies
   See: http://www.phpbuilder.com/manual/function.setcookie.php
   The values of the cookie variables are set by the function,
       setcookie()
   A call to setcookie() must be made before any html code,
       that is, before the <html> tag, because the values
       of the cookies are sent in the header to the browser
   Cookie variables are stored in the array, $_COOKIE
Steps:
Server
  1.  Data assignment from form
  2.  Create or modify cookie
  3.  Data validation and calculation
Browser
  4.  Data output to screen
  5.  Message output to screen by a form (corresponds to a prompt
in a C program)
  6.  Data input from keyboard to the form
  7.  Do step 1
-->
<?php
/*
1. ASSIGN DATA TYPED ON FORM TO VARIABLE
*/
//$number = $_POST['number'];   //data entry variable
if (isset($_POST['number']))
```

```
  $number = $_POST['number'];
else
  $number = NULL;
/*
2.  CREATE OR MODIFY COOKIE
*/
//Syntax: setcookie(name[, value[, expire[, path[, domain[, 
secure]]]]])
//The third argument of setcookie() is the expiration time of the 
cookie,
//in seconds after the beginning of 1970.
//If the expiration time of a cookie is not specified, the 
browser saves the cookie
//only in main memory, that is, not on disk.  In this case when 
the browser exits,
//the cookie disappears.
//set message variables
$msg1 = "";
$msg2 = "";
$msg3 = "";
//set expire variable
$expire = time() + 3600;    // cookie expires 1 hour after it is 
created or changed
if (is_null($number)) {
  //first time page is sent to browser
  $msg1 = "<h3><font color = green>First time that this page is 
sent to the browser</font></h3><br>";
  $count = 0;
  $sum = 0;
  //create cookie or change value of cookie
  setcookie('sum', $sum, $expire);
  setcookie('count', $count, $expire);
}
/*
3. DATA VALIDATION AND CALCULATION
*/
else  {
//not first time page is sent to browser
  //assign cookie variables to program variables
  $sum = $_COOKIE['sum'];   //calculated variable
  $count = $_COOKIE['count']; //calculated variable
  if (is_numeric($number)){
    //valid number
    $sum = $sum + $number;
    $count++;
    //change value of cookie variable
    setcookie('sum', $sum, $expire);
    setcookie('count', $count, $expire);
  }
  else if (! strcmp($number, "s")){
    //start new sum
    $msg1 = "<h3><font color = blue>Start new 
sum</font></h3><br>";
    $count = 0;
    $sum = 0;
    //change value of cookie variable
    setcookie('sum', $sum, $expire);
    setcookie('count', $count, $expire);
  }
  else if ($number == "")
    //number not typed
    $msg1 = "<h2><u><font color = red>Type a 
number</font></u></h2>";
  else
    //invalid number
    $msg1 = "<h2><u><font color = red>Invalid number: 
$number</font></u></h2>";
}
/*
4. DATA OUTPUT TO SCREEN
*/
if (! is_null($number)) {
  //this is not executed when the page is sent to the browser for 
the first time
  //display the sum
  $msg2 = sprintf("<h2>sum of %d numbers = %.2f</h2>", $count, 
$sum);
  //display a message and a link
  $page = $_SERVER['PHP_SELF'];   //link to same page
  $msg3 = "
  <h3><font color=blue>To start a new sum: type, s, instead of a 
number</font>
  <p><a href= $page >Reload Page</a></h3>
  ";
}
?>
<html>
<head><title>Adding Machine - Cookies</title></head>
<body>
<h1>Adding Machine - Using Cookies</h1>
<?php
   echo $msg1;
   echo $msg2;
?>
```

```
<!--
5. MESSAGE OUTPUT TO SCREEN
    Create and display form
6. DATA INPUT FROM KEYBOARD
    Enter data on form
-->
<!--
FORM: one text element
-->
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
<strong>Number</strong><br>
<input type="text" name="number">
<p><input type="submit" value="Add Number"><br>
</form>
<?php
   echo $msg3;
?>
<h2>
<a href="menustate.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

The display of the web page, addm_coo.php after the three numbers, 1, 2 and 3, are typed in the form is the same as in Figure 11.4.

11.7.4.1 Explanation of cookie web page, addm_coo.php

The PHP script in the cookie web page starts by determining if the value of number in the form exists. This part of the script is the same as in the web page which demonstrates hidden variables, addm_hid.php. If the web page is requested for the first time, number does not exist and $number is assigned the value, NULL. When the web page is requested after the first time, number exists and $number is assigned the value from the form, $_POST['number'].

The three message variables, $msg1, $msg2 and $msg3, are declared and assigned the value, "", which is the zero-length string. The variable, $expire, is assigned a value such that the cookie expires one hour after it is created.

The variable, $number, is tested for value, NULL, by the function, is_null().

(a) First time web page is requested

The PHP variable, $number, is NULL when the web page is requested for the first time. The value of variable, $msg1, is set to,

"First time that this page is sent to the browser".

The variables, $count and $sum, are initialized to zero. The two variables are stored in a cookie by calling the function, setcookie(), once for $count and once for $sum,

```
setcookie('sum', $sum, $expire);
```

```
   setcookie('count', $count, $expire);
```

But the variables in the cookie cannot be accessed until the next time that the web page is requested, that is, setcookies() does not store the variables in the array, $_COOKIE. The cookie with the two variables will be sent to the browser and saved by the browser. When the web page is requested next time, the cookie will be sent by the browser to the server and the server will store the cookie variables in the array, $_COOKIE.

The next PHP code which is executed is after the HTML code begins. The two variables, $msg1 and $msg2, are displayed. The value of $msg1 is given above and the value of $msg2 is the zero-length string, that is, nothing is displayed for $msg2.

In the form, the action attribute is given the value, address of current web page, $_SERVER['PHP_SELF']. So when submit is clicked the same web page will be requested again.

After the form, the variable, $msg3, is displayed, which is the zero-length string.

This completes the execution of the PHP code in the web page by the server. Now the server sends the web page and the cookie to the browser. The browser displays the web page on the screen and stores the cookie.

(b) Second and subsequent times web page is requested
The web page is requested after the first time by clicking Add Number on the screen, where Add Number is the name of the submit button on the form. The same web page, as given by the action attribute, is requested. The value of the text variable, number, is sent to the server and stored in the $_POST array, since the method in the form is POST. Also the cookie is sent to the server and the two name-value pairs in the cookie, with names sum and count, are stored in the array, $_COOKIE.

Now the value of number exists and so $number is assigned the value from the form,

```
   $number = $_POST['number'];
```

As when the web page was requested for the first time, the three message variables, $msg1, $msg2 and $msg3, are declared and assigned the value, "", which is the zero-length string. The variable, $expire, is assigned a value such that the cookie expires one hour after it is created.

Since $number is not NULL, the values of sum and count from the cookie, stored in array $_COOKIE, are assigned to variables in the script, $sum and $count,

```
   $sum = $_COOKIE['sum'];
   $count = $_COOKIE['count'];
```

Then, if $number is a valid number, the $number which was typed in the form is added to $sum, and $count is incremented (1 is added to $count). Then the new values of $sum and $count are stored in the cookie by calling setcookie() once for $sum and once for $count,

```
   setcookie('sum', $sum, $expire);
   setcookie('count', $count, $expire);
```

Before the beginning of the HTML code, values are assigned to variables, $msg2 and $msg3. The variable, $msg2, is set to,
"sum of $count numbers = $sum".
The variable, $msg3 is set to,
"To start a new sum: type, s, instead of a number
Reload Page".

After the HTML code begins the same PHP code is executed as when the web page was requested for the first time. Then the server sends the web page and the cookie to the browser. The browser displays the web page on the screen and stores the cookie.

Different parts of the PHP code are executed if letter, s, is typed which starts a new sum, and when an invalid number or no number is typed in the form. The comments in the web page describe the operation in these cases.

11.7.5 Session

The web page, addm_ses.php, implements the adding machine application by using a session to maintain state. The application adds numbers that are typed in a form. The number of numbers in the sum and the sum of the numbers after each number is added are displayed.

Start a text editor, emacs for example, and type the following file, or copy and paste it. Save the file in directory, c11state, using the name, addm_ses.php. Then change the permissions of the file to 604.

Web Page 11.4 addm_ses.php

```
<!--
addm_ses.php
2005 March 24
Updated: August 2019
Description:
   An adding machine which adds numbers typed on the keyboard
       and displays the sum on the screen
   State is maintained by a session
   A session is started or continued by the function,
session_start()
   Do not use the function, session_register() to register
       session variables
       This function is deprecated (its use is discouraged
because
          it is replaced)
       Also, this function does not work if, register_globals =
OFF,
          in the PHP configuration
   See: http://www.phpbuilder.com/manual/ref.session.php
   Session variables are stored in the array, $_SESSION
   Register a session variable by assigning it to the session
array:
       $_SESSION['variable_name'] = $variable_name
   A call to session_start() must be made before any html code,
       that is, before the <html> tag,
       because a cookie which contains the session ID is sent
       in the header to the browser
Steps:
Server
1.  Data assignment from form to variable
2.  Start or continue session
3.  Data validation and calculation
Browser
4.  Data output to screen
5.  Message output to screen by a form (corresponds to a prompt
in a C program)
6.  Data input from keyboard to the form
7.  Do step 1
-->
<?php
/*
1. ASSIGN DATA TYPED ON FORM TO VARIABLE
*/
//$number = $_POST['number'];   //data entry variable
if (isset($_POST['number']))
  $number = $_POST['number'];
else
  $number = NULL;
/*
2.  START OR CONTINUE SESSION
*/
//set message variables
$msg1 = "";
$msg2 = "";
$msg3 = "";
$lifetime = 3600;             //lifetime of session cookie in
                              seconds
session_set_cookie_params ($lifetime); //set lifetime of session
cookie
session_start();              //start or continue a session
if (is_null($number)) {
//first time page is sent to browser
  //session is started
  //cookie, PHPSESSID, will be sent to the browser
  //if lifetime > 0, cookie will be stored on disk
  //if lifetime = 0, cookie is stored only in main memory
  //and disappears after the browser is closed
  /*Code to display cookie parameters:
    $params = session_get_cookie_params();
    echo "<strong>Session Cookie Parameters</strong><br>";
    foreach ($params as $k => $v)
         echo "$k = $v<br>";
  */
  $msg1 = "<h3><font color = green>First time that this page is
sent to the browser</font></h3><br>";
  $count = 0;
  $sum = 0;
  //assign values to session variables
  //this also registers (creates) the session variables when they
do not exist
  $_SESSION['sum'] = $sum;
  $_SESSION['count'] = $count;
}
/*
3. DATA VALIDATION AND CALCULATION
*/
else  {
//not first time page is sent to browser
  //assign session variables to program variables
  $sum = $_SESSION['sum'];      //calculated variable
  $count = $_SESSION['count']; /        /calculated variable
  if (is_numeric($number)){
    //valid number
    $sum = $sum + $number;
    $count++;
    //assign values to session variables
    $_SESSION['sum'] = $sum;
    $_SESSION['count'] = $count;
  }
  else if (! strcmp($number, "s")){
    //start new sum
    $msg1 = "<h3><font color = blue>Start new
sum</font></h3><br>";
    $count = 0;
    $sum = 0;
    //assign values to session variables
    $_SESSION['sum'] = $sum;
    $_SESSION['count'] = $count;
}
  else if ($number == "")
    //number not typed
    $msg1 = "<h2><u><font color = red>Type a
number</font></u></h2>";
  else
```

```
    //invalid number
    $msg1 = "<h2><u><font color = red>Invalid number:
$number</font></u></h2>";
}
/*
4. DATA OUTPUT TO SCREEN
*/
if (! is_null($number)) {
  //this is not executed when the page is sent to the browser for
the first time
  //display the sum
  $msg2 = sprintf("<h2>sum of %d numbers = %.2f</h2>", $count,
$sum);
  //display a message and a link
  $page = $_SERVER['PHP_SELF'];   //link to same page
  $msg3 = "
     <h3><font color = blue>To start a new sum: type, s, instead
of a number</font>
     <p><a href= $page >Reload Page</a></h3>
  ";
}
?>
<html>
<head><title>Adding Machine - Session</title></head>
<body>
<h1>Adding Machine - Using a Session</h1>
<?php
    echo $msg1;
    echo $msg2;
?>
<!--
5. MESSAGE OUTPUT TO SCREEN
    Create and display form
6. DATA INPUT FROM KEYBOARD
    Enter data on form
-->
<!--
FORM: one text element
-->
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="POST">
<strong>Number</strong><br>
<input type="text" name="number">
<p><input type="submit" value="Add Number"><br>
</form>
<?php
echo $msg3;
?>
```

```
<h2>
<a href="menustate.htm" target=_top>Menu</a>
</h2>
</body>
</html>
```

The display of the web page, addm_ses.php after the three numbers, 1, 2 and 3, are typed in the form is the same as in Figure 11.4.

11.7.5.1 Explanation of session web page, addm_ses.php

The PHP script in the web page starts by determining if the value of number in the form exists. This part of the script is the same as in the web page which demonstrates cookies, addm_coo.php. If the web page is requested for the first time, number does not exist and $number is assigned the value, NULL. When the web page is requested after the first time, number exists and $number is assigned the value from the form, $_POST['number'].

The three message variables, $msg1, $msg2 and $msg3, are declared and assigned the value, "", which is the zero-length string.

The function, session_set_cookie_params(), is called. Then function, session_start(), is called to start a session or to continue a session if it has been already started. The function, session_set_cookie_params(), sets the lifetime of the session cookie and must be called before session_start() and every time that session_start() is called. The session cookie with one name-value pair will be sent to the browser. The content of the cookie is the session id with default name, PHPSESSID,

The variable, $number, is tested for value, NULL, by the function, is_null().

(a) First time web page is requested

The PHP variable, $number, is NULL when the web page is requested for the first time. The value of variable, $msg1, is set to,

"First time that this page is sent to the browser".

The variables, $count and $sum, are initialized to zero. Session variables are created for $count and $sum by

```
$_SESSION['sum'] = $sum;
$_SESSION['count'] = $count;
```

If the session variables exist, the same two statements assign new values to the session variables, as is the case when the web page is requested after the first time.

The next PHP code which is executed is after the HTML code begins. The two variables, $msg1 and $msg2, are displayed. The value of $msg1 is given above and the value of $msg2 is the zero-length string, that is, nothing is displayed for $msg2.

In the form, the action attribute is given the value, address of current web page, $_SERVER['PHP_SELF']. So when submit is clicked the same web page will be requested again.

After the form, the variable, $msg3, is displayed, which is the zero-length string.
This completes the execution of the PHP code in the web page by the server. Now the server sends the web page and the session cookie to the browser. The browser displays the web page on the screen and stores the session cookie.

(b) Second and subsequent times web page is requested
The web page is requested after the first time by clicking Add Number on the screen, where Add Number is the name of the submit button on the form. The same web page, as given by the action attribute, is requested. The value of the text variable, number, is sent to the server and stored in the $_POST array, since the method in the form is POST. Also the session cookie, which contains the session id, is sent to the server.

Now the value of number exists and so $number is assigned the value from the form,

```
   $number = $_POST['number'];
```

As when the web was requested for the first time, the three message variables, $msg1, $msg2 and $msg3, are declared and assigned the value, "", which is the zero-length string.

The function, session_set_cookie_params(), is called to set the lifetime of the session cookie. Then function, session_start(), is called to continue the session.

Since $number is not NULL, the values of sum and count from the session, stored in array $_SESSION, are assigned to variables in the script, $sum and $count,

```
   $sum = $_SESSION['sum'];
   $count = $_SESSION['count'];
```

Then, if $number is a valid number, the $number which was typed in the form is added to $sum, and $count is incremented (1 is added to $count). The new values of $sum and $count are stored by assigning them to the session variables,

```
   $_SESSION['sum'] = $sum;
   $_SESSION['count'] = $count;
```

Before the beginning of the HTML code, values are assigned to variables, $msg2 and $msg3.
The variable, $msg2, is set to,
    "sum of $count numbers = $sum".
The variable, $msg3 is set to,
    "To start a new sum: type, s, instead of a number
     Reload Page".

After the HTML code begins, the same PHP code is executed as when the web page was requested for the first time. Then the server sends the web page and the session cookie to the browser. The browser displays the web page on the screen and stores the session cookie.

Different parts of the PHP code are executed if letter, s, is typed which starts a new sum, and when an invalid number or no number is typed in the form. The comments in the web page describe the operation in these case.

12 MySQL API

12.1 Introduction

MySQL is the most popular open source database management system. Open source software can be used by anyone without cost. In contrast, Oracle is the most popular database management system which is not open source and therefore must be bought before it is commercially used.

The MySQL database management system (DBMS) is a program which responds to requests to access the database. So the MySQL DBMS is a server program and is sometimes called the MySQL server.

The MySQL API (application programming interface) is a set of declarations of functions in PHP which communicate with a MySQL database. The declarations are used to call the PHP functions. The name of every function starts with the letters, mysqli, followed by underscore, _. Only the declarations of the functions are available to the programmer, not the definitions of the functions.

There are basically two different ways to access a MySQL database.
(1) mysql, a shell program where commands are typed at the prompt
(2) MySQL API, functions called in a PHP script
This chapter covers the MySQL API. Chapter 2 MySQL Shell covers the shell program, mysql.

MySQL can be installed on a home computer so that web pages which access a database can be tested before transferring the web pages to a web site on a remote computer. The instructions to install MySQL on a home computer are given in Chapter 2 MySQL Shell.

12.2 Sample Database

The sample database consists of one table which is used to introduce the basic MySQL functions. This is the same table as used in Chapter 3 SQL Basics. The figure in Chapter 3 of the ER diagram, the schema and the instance of the database are repeated in Figure 12.1.

The terms, field, attribute and column, all mean the column of a database table. The term, attribute, is not used in this chapter. The terms, field and column, are both used. There are four fields (columns) in the table, Person, in Figure 12.1: fName, weight, height and gender.

12.3 Common MySQL Functions

The names and brief descriptions of the common functions of the MySQL API in PHP are given in Table 12.1. Most of the MySQL functions in Table 12.1 are demonstrated in this chapter. The function, mysqli_close(), is not demonstrated because disconnect from MySQL always occurs at the end of a PHP script.

Table 12.1 Common MySQLi_ API functions

Category	Name	Description
Connect	mysqli_connect()	connect to MySQL server
	mysqli_close()	disconnect from MySQL server
	mysqli_select_db()	select a MySQL database
	mysqli_query()	execute a SQL command
Display	mysqli_fetch_array()	fetch next row of the result as an array: associative , numerical or both
	mysqli_data_seek()	move internal row pointer
Error	mysqli_errno()	return number of last error message
	mysqli_error()	return text of last error message
Number	mysqli_num_fields()	return number of columns(fields) in the result of a query
	mysqli_num_rows()	return number of rows in the result of a SELECT command
	mysqli_affected_rows()	return number of affected rows by last INSERT, UPDATE or DELETE
Name	mysqli_fetch_field_direct()	return information for a particular column from a result
Structure	mysqli_fetch_field_direct()	returns the details for one column of a result: used in a loop for all columns

	mysqli_fetch_field() OR mysqli_fetch_field_direct()	return flags of result lt (e.g., not null, primary key, auto increment)

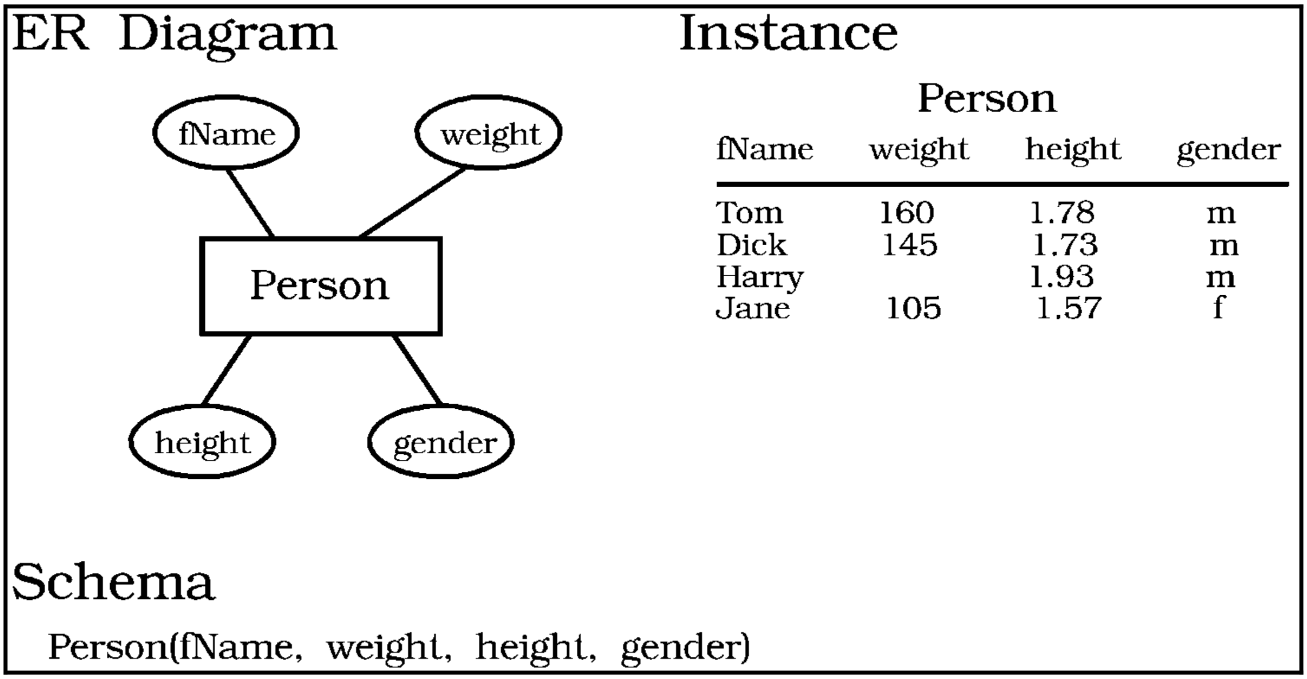
Declarations of MySQL functions

The MySQL API is the set of declarations of the MySQL functions which access a MySQL database. The declarations of the functions in Table 12.1 are given below. The declarations and the descriptions of the arguments and return values are from the PHP Manual. The web site of the PHP Manual which contains the declarations of all of the MySQL functions is given in the last section, References.

The declaration of the MySQL functions in Table 12.1 are given in alphabetical order by names of the functions.

<u>Alphabetical list of names of MySQL functions</u>

1. Affected rows
2. Close
3. Connect
4. Data seek
5. Errno
6. Error
7. Fetch array
8. Fetch field
9. Field flags
10. Field name
11. Field table
12. Num fields
13. Num rows
14. Query
15. Select db



Person

fName	weight	height	gender
Tom	160	1.78	m
Dick	145	1.73	m
Harry		1.93	m
Jane	105	1.57	f

Person(fName, weight, height, gender)

Figure 12.1 Sample Database

12.3.1 Affected rows

Declaration
    int mysqli_affected_rows ([resource link_identifier])
Arguments
    link_identifier - return value of mysqli_connect()
Return value
    number of affected rows if success, -1 if failure

12.3.2 Close

Declaration
    bool mysqli_close ([resource link_identifier])
Arguments
    link_identifier - return value of mysqli_connect()
Return value
    TRUE if success, FALSE if failure

12.3.3 Connect

Declaration
    resource mysqli_connect ([string servername [, string username [, string password [, bool new_link [, int client_flags]]]]])
Arguments
    servername - name of MySQL server
    username - name of user
    password - password of user
    new_link - if TRUE, new return value when mysqli_connect is called a second time
    client_flags - MYSQL_CLIENT_COMPRESS, MYSQL_CLIENT_IGNORE_SPACE or MYSQL_CLIENT_INTERACTIVE
Return value
    Link identifier if success, FALSE if failure

12.3.4 Data seek

Declaration
    bool mysqli_data_seek (resource result, int row_number)
Arguments
    result - return value of mysqli_query()
    row_number - row number to be assigned to internal row pointer
        (0 is the number of the first row, 1 is the second row, and so on)
Return value
    TRUE if success, FALSE if failure

12.3.5 Errno

Declaration
    int mysqli_errno ([resource link_identifier])
Arguments
    link_identifier - return value of mysqli_connect()
Return value

error number from last MySQL function, 0 if no error occurred

12.3.6 Error

Declaration

string mysqli_error ([resource link_identifier])

Arguments

link_identifier - return value of mysqli_connect()

Return value

error text from last MySQL function, “” (empty string) if no error occurred

12.3.7 Fetch array

Declaration

array mysqli_fetch_array (resource result [, int result_type])

Arguments

result - return value of mysqli_query()

result_type - type of array to be returned: MYSQL_ASSOC, MYSQL_NUM, MYSQL_BOTH

Return value

array which contains the fetched row if success, FALSE if no row

12.3.8 Fetch field

Declaration

object mysqli_fetch_field (resource result [, int field_offset])

Arguments

result - return value of mysqli_query()

field_offset - field number of the next field to be fetched
(0 is the number of the first field, 1 is the second field, and so on)

Return value

if success: object containing the following properties of the field:

name - column name
table - name of the table the column belongs to
max_length - maximum length of the column
not_null - 1 if the column cannot be NULL, 0 otherwise
primary_key - 1 if the column is a primary key, 0 otherwise
unique_key - 1 if the column is a unique key, 0 otherwise
multiple_key - 1 if the column is a non-unique key, 0 otherwise
numeric - 1 if the column is numeric, 0 otherwise
blob - 1 if the column is a BLOB, 0 otherwise
type - the type of the column
unsigned - 1 if the column is unsigned, 0 otherwise
zerofill - 1 if the column is zero-filled, 0 otherwise

if failure: FALSE

12.3.9 Field flags

Declaration

string mysqli_field_flags (resource result, int field_offset)
Arguments
result - return value of mysqli_query()
field_offset - field number of the next field
(0 is the number of the first field, 1 is the second field, and so on)
Return value
flags of the column (one word for each flag) separated by one space, FALSE if failure
(Examples of flags: "not_null", "primary_key", "unique_key")

12.3.10 Field name

Declaration
string mysqli_field_name (resource result, int field_offset)
Arguments
result - return value of mysqli_query()
field_offset - field number of the next field name
(0 is the number of the first field, 1 is the second field, and so on)
Return value
name of the field if success, FALSE if failure

12.3.11 Field table

Declaration
string mysqli_field_table (resource result, int field_offset)
Arguments
result - return value of mysqli_query()
field_offset - field number of the next field name
(0 is the number of the first field, 1 is the second field, and so on)
Return value
name of the table which contains the field if success, FALSE if failure

12.3.12 Num fields

Declaration
int mysqli_num_fields (resource result)
Arguments
result - return value of mysqli_query()
Return value
number of fields if success, FALSE if failure

12.3.13 Num rows

Declaration
int mysqli_num_rows (resource result)
Arguments
result - return value of mysqli_query()
Return value
number of rows if success, FALSE if failure

12.3.14 Query

Declaration

resource mysqli_query (string query [, resource link_identifier])

Arguments

query - SQL command (does not end with ;)

link_identifier - return value of mysqli_connect()

Return value

SELECT, SHOW, DESCRIBE, EXPLAIN: resource if success, FALSE if failure

UPDATE, DELETE, DROP, and all others: TRUE if success, FALSE if failure

12.3.15 Select db

Declaration

bool mysqli_select_db (string database_name [, resource link_identifier])

Arguments

database_name - name of database

link_identifier - return value of mysqli_connect()

Return value

TRUE if success, FALSE if failure

12.4 Outline of a Database Web Page

Every database web page which performs an operation on a MySQL database has a similar organization.

The mandatory steps in every database web page, together with the MySQL functions that are used, are listed in Table 12.2. At the beginning of every PHP script it is necessary to connect to the MySQL server, Step 1, because at the end of every script disconnect occurs automatically. After connecting, an SQL command can be executed, by Steps 2, 3 and 4. The optional steps can include executing other MySQL functions, for example a function to display data after a SELECT command, or Steps 3 and 4 can be repeated for a different SQL command.

12.5 Demonstrations of MySQL Functions

The MySQL functions are demonstrated one at a time, using the sample database in Figure 12.1 when required. The return value of each MySQL function must be examined to determine if the function was successful, that is, if the function accomplished its task. In some cases if the function is not successful, the MySQL error functions, mysqli_errno() or mysqli_error() or both, are executed and then the PHP script is terminated. As shown in Table 12.2, the error functions are always executed if mysqli_connect(), mysqli_select_db() or mysqli_query() are not successful.

It is assumed that either EasyPHP or wamp was used to install MySQL as instructed in Chapter 2 MySQL Shell. Assuming that the document root directory of Apache is, C:/EasyPHP/www, or, C:/wamp/www, make a directory in www, with name c12mysqlapi. Directories can be made by Windows Explorer.

All web pages in this chapter will be stored in directory

C:/EasyPHP/www/c12mysqlapi or

C:/wamp/www/c12mysqlapi

A web page, webpage1.htm, for example, is requested from the Apache server by typing in the address bar of a browser, Internet Explorer for example, the address
http://localhost/c12mysqlapi/webpage1.htm

Table 12.2 Outline of Web Page

Step		MySQL Function
Mandatory	1. Connect to MySQL	mysqli_connect()
	If not successful	mysqli_errno(), mysqli_error()
	2. Select a database	mysqli_select_db()
	If not successful	mysqli_errno(), mysqli_error()
	3. Make a SQL command	(none)
	4. Execute SQL command	mysqli_query()
	If not successful	mysqli_errno(), mysqli_error()
Optional	5. Perform other operations	(various)

Use the text editor, Notepad, to create the web pages. Start Notepad by clicking,
Start/Programs/Accessories/Notepad

When files are saved in Notepad, the extension, .txt, may be added automatically, unless the extension is already, .txt. For example, the file with name, webpage1.htm, may be saved with name, webpage1.htm.txt. To prevent this, when saving the file in Notepad, enclose the name of the file in quotation marks, “webpage1.htm”.

12.5.1 Menu

A web page with a menu, menuapi.htm, has links to all of the web pages which demonstrate the MySQL API functions. Each demonstration web page has one form and also a link back to the menu web page. Table 12.1 Common MySQL API Functions is also displayed on the right side of the menu web page.

Every demonstration web page has the HTML source code of the web page displayed at the top of the page and the output of the HTML source code at the bottom. The lines of source code and the output of the corresponding lines of code can be readily related since the pages are short. Note that the value of the action attribute in the form tag is the web page to which the data entered in the form is sent when the submit button is clicked. All demonstration web pages and the menu web page are in the same directory.

Web page - menuapi.htm

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, menuapi.htm, in directory, c12mysqlapi.

(Beginning of web page)

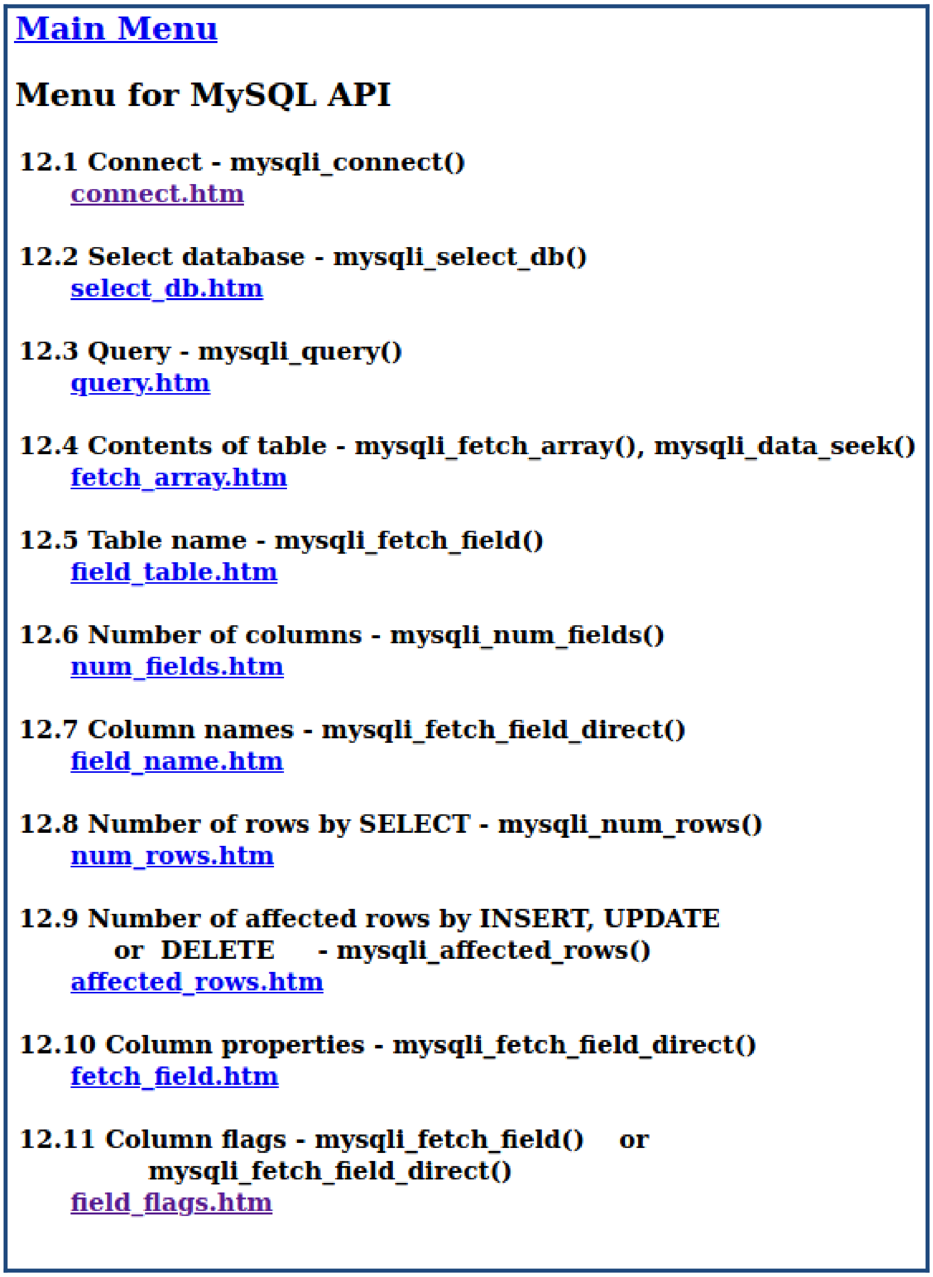
```
<HTML>
<HEAD>
<TITLE>Menu for MySQL API</TITLE>
</HEAD>
<BODY>
<h2>
<a href="../index.html">Main Menu</a>
</h2>
<h1>Menu for MySQL API</h1>
<table width=100% border=0><tr>
<td width=40%>
<h3>
<!--*****************
*****1. Connect******
******************-->
12.1 Connect - mysqli_connect()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="connect.htm">connect.htm</a><br /><br />
<!--**************************
*****2. Select database*******
***************************-->
12.2 Select database - mysqli_select_db()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="select_db.htm">select_db.htm</a><br /><br />
<!--*******************
*****3. Query**********
********************-->
12.3 Query - mysqli_query()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="query.htm">query.htm</a><br /><br />
<!--*****************************
*****4. Contents of table********
******************************-->
12.4 Contents of table - mysqli_fetch_array(),
mysqli_data_seek()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="fetch_array.htm">fetch_array.htm</a><br /><br />
<!--**********************
*****5. Table name********
***********************-->
12.5 Table name - mysqli_field_table()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="field_table.htm">field_table.htm</a><br /><br />
```

```
<!--*****************************
*****6. Number of columns********
******************************-->
12.6 Number of columns - mysqli_num_fields()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="num_fields.htm">num_fields.htm</a><br /><br />
<!--*****************************
*****7. Column names*************
******************************-->
12.7 Column names - mysqli_field_name()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="field_name.htm">field_name.htm</a><br /><br />
<!--**************************
*****8. Number of rows********
***************************-->
12.8 Number of rows by SELECT - mysqli_num_rows()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="num_rows.htm">num_rows.htm</a><br /><br />
<!--*************************************************************
**
*****9. Number of affected rows by INSERT, UPDATE or
DELETE********
****************************************************************--
>
12.9 Number of affected rows by INSERT, UPDATE or DELETE
 - mysqli_affected_rows()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="affected_rows.htm">affected_rows.htm</a><br /><br />
<!--*****************************
*****10 Column properties*******
******************************-->
12.10 Column properties - mysqli_fetch_field()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="fetch_field.htm">fetch_field.htm</a><br /><br />
<!--*****************************
*****11 Column flags************
******************************-->
12.11 Column flags - mysqli_field_flags()<br />
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<a
href="field_flags.htm">field_flags.htm</a><br /><br />
</h3>
</td>
<td width=60% height=100% valign=top>
<iframe src=apifunctions.htm width=100% height=100%
frameborder=0></iframe>
</td></tr></table>
</BODY>
```

```
</HTML>
```

(End of web page)

Figure 12.2 shows the first part of the menu web page. Table 12.1 which is displayed by the browser is not shown in Figure 12.2.



Main Menu

Menu for MySQL API

12.1 Connect - mysqli_connect()
connect.htm

12.2 Select database - mysqli_select_db()
select_db.htm

12.3 Query - mysqli_query()
query.htm

12.4 Contents of table - mysqli_fetch_array(), mysqli_data_seek()
fetch_array.htm

12.5 Table name - mysqli_fetch_field()
field_table.htm

12.6 Number of columns - mysqli_num_fields()
num_fields.htm

12.7 Column names - mysqli_fetch_field_direct()
field_name.htm

12.8 Number of rows by SELECT - mysqli_num_rows()
num_rows.htm

12.9 Number of affected rows by INSERT, UPDATE
or DELETE - mysqli_affected_rows()
affected_rows.htm

12.10 Column properties - mysqli_fetch_field_direct()
fetch_field.htm

12.11 Column flags - mysqli_fetch_field() or
mysqli_fetch_field_direct()
field_flags.htm

Figure 12.2 Menu web page, menuapi.htm

12.5.2 Connect - mysqli_connect()

The function, mysqli_connect(), called in a PHP script connects the script to the MySQL server. This process is also called logging on or logging in to the server. The servername, username and password are supplied by the user.

Web page - democonnect.php
Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, democonnect.php, in directory, c12mysqlapi.
(Beginning of web page)

```
<?php
//democonnect.php
//servername from connect form
$servername = $_POST['servername'];
//username from connect form
$username = $_POST['username'];
//password from connect form
$password = $_POST['password'];
echo "<br>servername = $servername<br>
   username = $username<br>
   password = $password<br>";
//connect to MySQL
//  $servername = 'localhost';
//  $username = 'root';
//  $password = '';
  $conn = @mysqli_connect($servername, $username, $password);

/*****************
Test for connect error
*****************/
  if ($conn == false)
    {
      $estring = mysqli_error();
      $enumber = mysqli_errno();
      echo "<br>CONNECT FAILED! Error number " . $enumber. ": " .
$estring;
      echo "<br><br>Click, <b>Back</b>";
      exit;
    }
  else
    {
    echo "<br>connect successful";
    echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
    }
?>
```

(End of web page)

Procedure

Click the link, connect.htm, in the menu. In the form, Connect, there are three text boxes: servername, username and password.

Assume that the servername is localhost, username is root and root does not have a password.

Type the following incorrect values and observe the error message. Click, Back, in the browser to return to the forms web page.

Servername: (blank)	Username: (blank)	Password: (blank)
Servername: a	Username: b	Password: c
Servername: localhost	Username: b	Password: c
Servername: localhost	Username: root	Password: c

Now type the correct values. Click the link, Form, to return to the form web page.

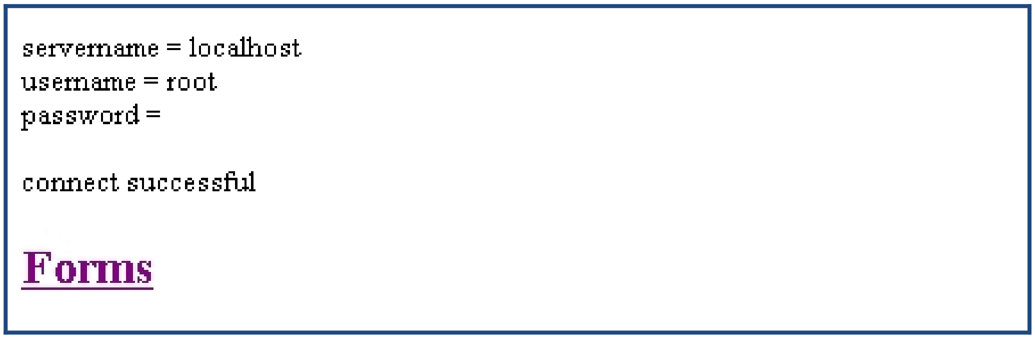
Servername: localhost	Username: root	Password: (blank)

It is unnecessary to type the servername when both the server (MySQL) and client (browser) are running on the same computer. Type only the username, root.

Servername: (blank)	Username: root	Password: (blank)

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.3 shows the output of the web page, democonnect.php, with valid values of servername, username and password.



```
servername = localhost
username = root
password =

connect successful
```

Forms

Figure 12.3 Output: democonnect.php, for valid values

12.5.3 User-defined functions

Every web page which accesses a database must connect to the MySQL server, since a web page disconnects from the server when the web page terminates. So the instructions in web page, democonnect.php, must be appear in all web pages that access a database. Obviously, the instructions in this web page should be constructed as a function which is called from each web page that connects to the MySQL server.

Functions which are called by more than one web page are put in a separate file. This file is included by every web page that uses the functions. There are two different instructions to include a file in a PHP script. They are

```
include ('filename');
require ('filename');
```

Both instructions combine the web page and the file, '*filename*', by replacing the instruction, include ('filename'); or , require ('*filename*'); by the contents of the entire file, *filename*. The two statements are identical except when the statements fail. The statements fail when the file does not exist. The include statement issues a warning and then the execution of the script continues. The require statement terminates execution of the script. The require statement should be used when the script cannot be executed unless the file is written into the web page.

Web page - userfunctions.php
Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, userfunctions.php, in directory, c12mysqlapi.

(Beginning of web page)

```
<?php
//userfunctions.php
//////////////////////////
/////////connect//////////
//////////////////////////
function connect()
{
  //If necessary, change the value of $username to your username
  //and change the value of $password to your password
  //The default administrator account has username, root, and no
password
  $servername = 'localhost';
  $username = 'root';
  $password = '';
  $conn = mysqli_connect($servername, $username, $password);
/*********************
Test for connect error
*********************/
  if ($conn == false)
    {
      $estring = mysqli_error();
      $enumber = mysqli_errno();
      echo "<br>CONNECT FAILED! Error number " . $enumber. ": " .
$estring;
      echo "<br><br>Click, <b>Back</b>";
      exit;
    }
  else
    {
      echo "<br>connect successful<br>";
      return $conn;
    }
}
////////////////////////////
/////////select db//////////
```

```
////////////////////////////
function selectdb($dbname, $conn)
{
//select db
//$dbname = "webdb";
$selectdb_return = mysqli_select_db($dbname, $conn);
/***********************
Test for select db error
***********************/
  if ($selectdb_return == false)
    {
      $estring = mysqli_error();
      $enumber = mysqli_errno();
      echo "<br>SELECT DATABASE FAILED! Error number " .
$enumber. ": " . $estring;
      echo "<br><br>Click, <b>Back</b>";
      exit;
    }
  else
    {
      echo "<br>select database successful";
      echo "<br>database name: $dbname<br>";
    }
}
////////////////////////
/////////query//////////
////////////////////////
function query($sql, $conn)
{
//query
$result = mysqli_query($sql, $conn);
/*******************
Test for query error
*******************/
if ($result == false)
  {
     $estring = mysqli_error();
     $enumber = mysqli_errno();
     echo "<br>QUERY FAILED! Error number " . $enumber. ": " .
$estring;
     echo "<br><br>Click, <b>Back</b>";
     exit;
  }
else
  {
     echo "<br>query successful<br>";
     return $result;
  }
}
////////////////////////////////////
/////////contents of table//////////
////////////////////////////////////
function contents($result, $sql)
{
echo "<h2>Column Data<br>";
echo "SQL command: <font color=red>$sql</font></h2>";
//column names
//move the internal row pointer to the first row, row 0
setrowpointer($result, 0);
echo "<table border=2>";
echo "<tr>";
$row = mysqli_fetch_array($result, MYSQL_ASSOC);
foreach ($row as $k => $col_value)
   echo "<th>$k</th>";
echo "</tr>";
//data
//move the internal row pointer to the first row, row 0
setrowpointer($result, 0);
while ($row = mysqli_fetch_array($result, MYSQL_ASSOC)) {
   echo "<tr>";
   foreach ($row as $col_value)
       echo "<td>$col_value</td>";
   echo "</tr>";
}
echo "</table>";
}
////////////////////////////////////
/////////set row pointer////////////
////////////////////////////////////
//called by function, contents()////
function setrowpointer($result, $rownumber)
{
$seekreturn = @mysqli_data_seek($result, $rownumber);
  if ($seekreturn == false)
    {
      $estring = mysqli_error();
      $enumber = mysqli_errno();
      echo "<br>DATA SEEK FAILED! Error number " . $enumber. ": "
. $estring;
      echo "<br><br>Click, <b>Back</b>";
      exit;
    }
}
////////////////////////////////////
```

```
/////////strip slashes/////////////
/////////////////////////////////////
//php adds slashes if magic_quotes_gpc is on
//this function strips slashes if magic_quotes_gpc is on
function ss($value)
{
     return ini_get('magic_quotes_gpc') ? stripslashes($value) : 
$value;
}
?>
```

(End of web page)

The file, userfunctions.php, contains six functions. The functions are used in the demonstrations in this chapter, but also can be used to access any MySQL database. The function names and a brief description of the functions follow.

(1) connect()
- calls mysqli_connect(), mysqli_error() and mysqli_errno()
- action: connect and test for error
- the servername, username and password are values of strings assigned in connect()
- edit the web page, userfunctions.php, to assign the correct servername, username and password to the variables with the same name. Use Notepad or some other text editor to edit the web page.

(2) selectdb()
- calls mysqli_select_db(), mysqli_error() and mysqli_errno()
- action: select a database and test for error

(3) query()
- calls mysqli_query(), mysqli_error() and mysqli_errno()
- action: execute a query and test for error

(4) contents()
- calls mysqli_fetch_array() and mysqli_data_seek()
- action: display contents of rows and columns of a table

(5) setrowpointer()
- calls mysqli_data_seek(), mysqli_error() and mysqli_errno()
- action: set row number and test for error
- called by contents() to set row number to 0

(6) ss()
- strips slashes if magic_quotes_gpc is on

12.5.4 Select db - mysqli_select_db()

The function, mysqli_select_db(), selects a database.

Web page - demoselectdb.php

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demoselectdb.php, in directory, c12mysqlapi.

This web page includes the web page, userfunctions.php, by the statement, require ('userfunctions.php');

The function, connect(), in userfunctions.php is called in demoselectdb.php. The values of servername, username and password are assigned in the function, connect(). Change the values, if necessary.

(Beginning of web page)

```
<?php
//demoselectdb.php
require ('userfunctions.php');
//connect
$conn = connect();
//Database name from form
//@ suppresses Notice: Undefined index
//   when "dbname" is null
$db = @$_POST['dbname'];
echo "<br>database name: $db<br>";
//select database
$selectdb_return = mysqli_select_db($db, $conn);
/*****************
Test for select db error
*****************/
  if ($selectdb_return == false)
    {
      $estring = mysqli_error();
      $enumber = mysqli_errno();
      echo "<br>SELECT DATABASE FAILED! Error number " .
$enumber. ": " . $estring;
      echo "<br><br>Click, <b>Back</b>";
      exit;
    }
  else
    {
    echo "<br>select database successful<br>";
    echo "<h2>current database: <font color=red>$db</font><h2>";
    echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
    }
?>
```

(End of web page)

Procedure

Click the link, select_db.htm, in the menu. In the form, Select database, there is a textbox for the database name.

Click, Submit, without typing any value in the textbox. An error message appears. Click, Back, in the browser.

Type

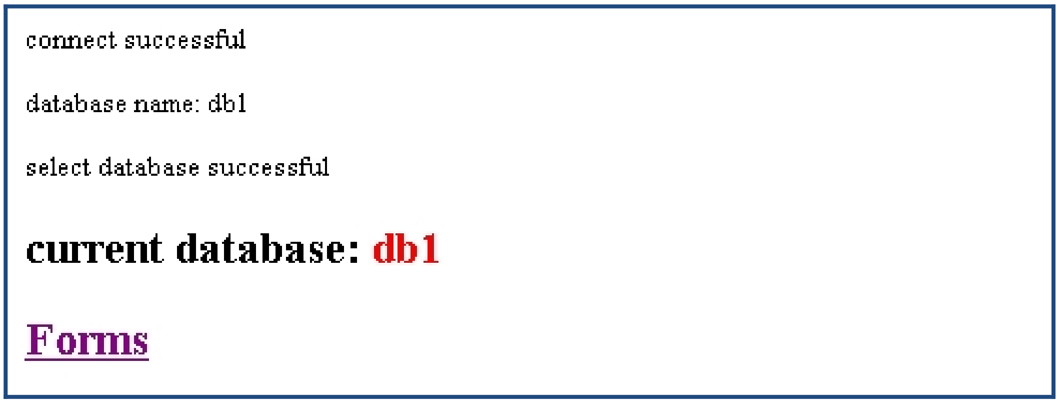
xyz

in the textbox. Assuming that there is no database with this name, an error message appears. Click, Back, in the browser.
Type
webdb
in the textbox. Assuming that there is a database with this name, the operation is successful.

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

If the database, webdb, does not exist it can be created by using mysql, the MySQL shell covered in Chapter 2. Also, it can be created by typing in the form, Query, (which follows the form, Select database) the command
create database webdb

Figure 12.4 shows the output of the web page, demoselectdb.php, when a valid database name is typed.



connect successful

database name: db1

select database successful

current database: db1

Forms

Figure 12.4 Output; demoselectdb.php, for a valid database name

12.5.5 Query - mysqli_query()

The function, mysqli_query(), sends to the MySQL server a SQL command or other commands recognized by the MySQL server.

Web page - demoquery.php
Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demoquery.php, in directory, c12mysqlapi.

This web page includes the web page, userfunctions.php, by the statement,
require ('userfunctions.php');
The functions, connect() and selectdb(), in userfunctions.php are called in demoquery.php.

(Beginning of web page)

```
<?php
//demoquery.php
```

```
require ('userfunctions.php');
//connect
$conn = connect();
//select db
$dbname = "webdb";
selectdb($dbname, $conn);
//sql command from form
//@ suppresses Notice: Undefined index
//   when "sqlcommand" is null
$sql = @$_POST["sqlcommand"];
echo "<br>SQL command: $sql<br>";
//query
$result = mysqli_query($sql, $conn);
/*****************
Test for query error
*****************/
if ($result == false)
{
   $estring = mysqli_error();
   $enumber = mysqli_errno();
   echo "<br>QUERY FAILED! Error number " . $enumber. ": " .
$estring;
   echo "<br><br>Click, <b>Back</b>";
   exit;
}
else
{
   echo "<br>query successful";
   echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
}
?>
```

(End of web page)

Procedure

Click the link, query.htm, in the menu. In the form, Query, there is a textbox for the SQL command.

Click, Submit, without typing any value in the textbox. An error message appears informing the user that the query is empty. Click, Back, in the browser.

Type

xxx

in the textbox, which is an invalid command. An error message appears informing the user that there is a SQL syntax error. Click, Back, in the browser.

Type

select * from zzz

in the textbox. Assuming the table, zzz, does not exist in database, webdb, an error message appears informing the user that table, webdb.zzz, does not exist.

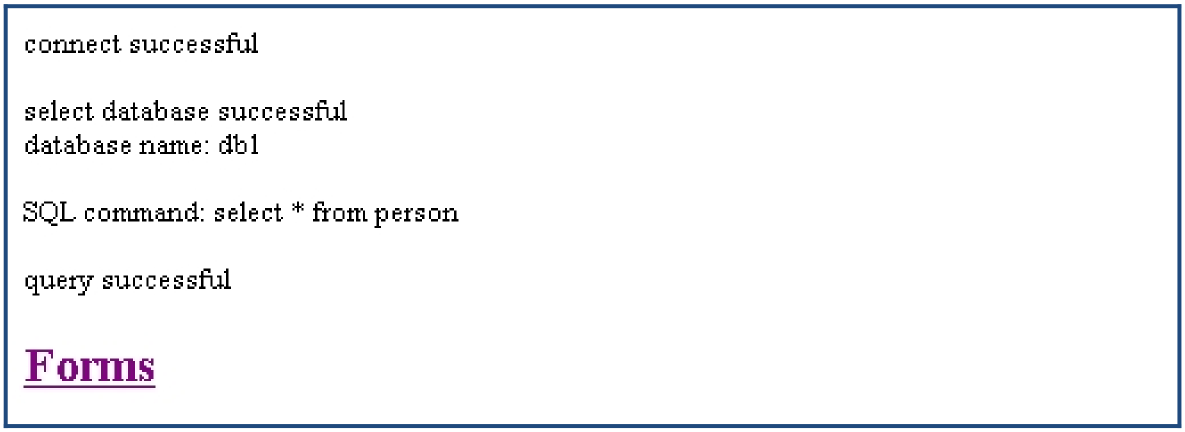
Table, Person, was created and populated using mysql previously. Assuming that this was done, the commands to select the contents of the table will be executed without error. However, another function is needed to actually display the table, covered in the next subsection.
Type

```
select * from Person
```

in the textbox. Assuming the table, Person, exists there is no error.

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.5 shows the output of the web page, demoquery.php, when a valid SQL command is typed.



```
connect successful

select database successful
database name: db1

SQL command: select * from person

query successful
```

Figure 12.5 Output: demoquery.php, for a valid SQL command

If table, Person, does not exist, create and populate the table, shown in Figure 12.1, with the following two commands. If table, Person, does exist the same commands can be tried by creating and populating another table, with name, Person2, for example.

The database is created by the CREATE TABLE command. This database consists of only one table, Person. The schema of Person in Figure 12.1 gives the names of the columns of the table which must be created. The following data types are chosen for the columns: fName: VARCHAR(10), weight: INTEGER(3), height: FLOAT(3,2), gender: CHAR(1). The column, fName, is declared to be the primary key and gender is declared not null.
Create the table, Person.

```
create table Person (
fName varchar(10) primary key,
weight integer(3),
height float(3,2),
gender char(1) not null)
```

The database is populated with data by the INSERT INTO command. An instance of the table, Person, is shown in Figure 12.1 which is used to populate the table.
Populate the table, Person.

```
insert into Person values
('Tom', 160, 1.78, 'm'),
('Dick', 145, 1.73, 'm'),
('Harry', null, 1.93, 'm'),
('Jane', 105, 1.57, 'f')
```

12.5.6 Contents of table - mysqli_fetch_array(), mysqli_data_seek()

The function, mysqli_fetch_array(), fetches the next row of the result as an associative array, numerical array or both. The function, mysqli_data_seek(), sets the row number of the next row to be fetched. The function, mysqli_fetch_array(), fetches row number 0 the first time it is called and then increments the row counter. The second time the function is called it fetches row number 1, and so on. So the function, mysqli_data_seek(), is not needed unless the rows are to be fetched in some other order.

Web page - demofetcharray.php
Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demofetcharray.php, in directory, c12mysqlapi.

This web page includes the web page, userfunctions.php, by the statement,
require ('userfunctions.php');
The functions, connect(), selectdb() and query(), in userfunctions.php are called in demofetcharray.php.

(Beginning of web page)

```
<?php
//demofetcharray.php
require ('userfunctions.php');
//connect
$conn = connect();
//select db
$dbname = "webdb";
selectdb($dbname, $conn);
//table or sql command from form
//@ suppresses Notice: Undefined index
//   when "table" is null
$t = @$_POST["table"];
$sql = @$_POST["sqlcommand"];
$f = $_POST["contents"];
if ($t != "" && $f == "total")
//total contents of table
  {
    $sql = "select * from $t";
    echo "<br>SQL command: $sql";
  }
else if ($sql != "" && $f == "partial")
```

```
//partial contents of table
  {
    //sql command from form
    echo "<br>SQL command: $sql";
  }
else
//both text boxes, table and sqlcommand, are blank
    $sql = "";
//query
$result = query($sql, $conn);
echo "<h3>SQL command: <font color=red>$sql</font></h3>";
//associative array: result_type=MYSQL_ASSOC
//fetch rows
//display all rows using foreach
//1. Display data only
echo "<h3>Table 1: Data only</h3>";
echo "<table border=1>";
while ($row = mysqli_fetch_array($result, MYSQL_ASSOC)) {
   echo "<tr>";
   foreach ($row as $col_value)
       echo "<td>$col_value</td>";
   echo "</tr>";
}
echo "</table>";
//2. Display column names and data
echo "<h3>Table 2: Column names and data</h3>";
echo "<table border=2>";
echo "<tr>";
//column names
//move the internal row pointer to the first row, row 0
mysqli_data_seek($result, 0);
$row = mysqli_fetch_array($result, MYSQL_ASSOC);
foreach ($row as $k => $col_value)
   echo "<th>$k</th>";
echo "</tr>";
//data
//move the internal row pointer to the first row, row 0
mysqli_data_seek($result, 0);
while ($row = mysqli_fetch_array($result, MYSQL_ASSOC)) {
   echo "<tr>";
   foreach ($row as $col_value)
       echo "<td>$col_value</td>";
   echo "</tr>";
}
echo "</table>";
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, fetch_array.htm, in the menu. In Contents of table, there are two forms. In the form, Total table contents, there is a textbox for the name of a table. In the form, Partial table contents, there is a textbox for a SQL command.

Form: Total table contents

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

```
abc
```

in the textbox and then click, Submit. Assuming that abc is not the name of a table, an error message appears. Click, Back, in the browser.

Type

```
Person
```

in the textbox and then click, Submit. Assuming that Person is the name of a table in the database, there is no error. Note that the SQL command which was executed is, select * from Person.

Click the link, Form, to return to the form web page.

Form: Partial table contents

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

```
Person
```

in the textbox and then click, Submit. The command, Person, is not a valid command and so an error message appears. Click, Back, in the browser.

Type

```
select * from Person
```

in the textbox and then click, Submit. This is a valid command and assuming table, Person, exists, there is no error. Note that the SQL command which was executed is, select * from Person, the same as when Person is typed in the form, Total table contents. Click the link, Form, to return to the form web page.

Type the commands, one at a time,

```
select fName, gender from Person
select gender, count(fName) from Person group by gender
select gender, count(fName) as "Number of persons" from Person group
by gender
```

in the textbox and then click, Submit. If table, Person, exists with columns fName and gender, there is no error. Click the link, Form, to return to the form web page after each command.

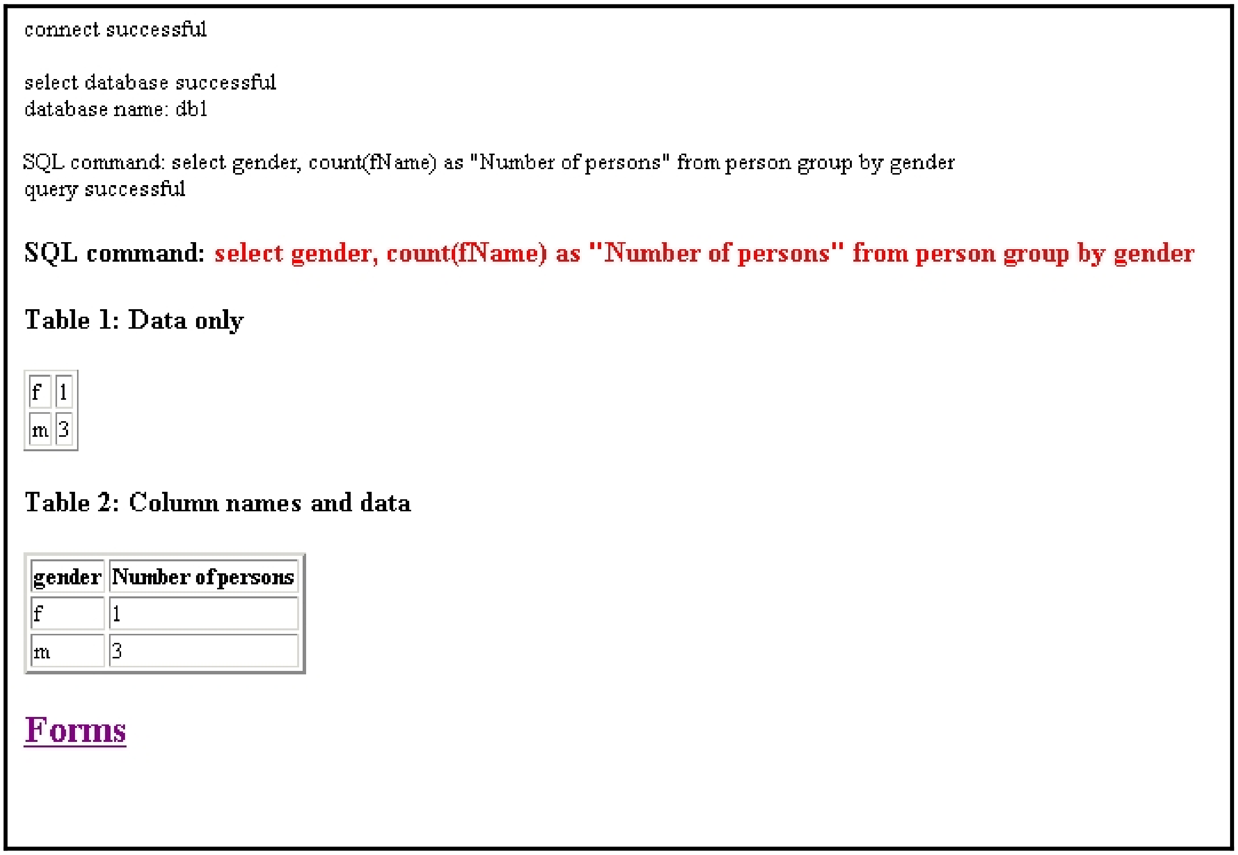
The following commands also display the contents of a table even though they are not SELECT commands. Type the following commands one at a time, all of which are valid and then click, Submit. Click the link, Form, to return to the form web page after each command.

```
desc Person
show columns from Person
```

show databases
show tables

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.6 shows the output of the web page, demofetcharray.php, when a valid SQL select command is typed.



```
connect successful

select database successful
database name: db1

SQL command: select gender, count(fName) as "Number of persons" from person group by gender
query successful
```

SQL command: select gender, count(fName) as "Number of persons" from person group by gender

Table 1: Data only

f	1
m	3

Table 2: Column names and data

gender	Number of persons
f	1
m	3

Forms

Figure 12.6 Output: demofetcharray.php, for a valid SQL select command

12.5.7 Table name - mysqli_field_table()

The function, mysqli_field_table(), returns the name of the table which contains a particular column.

Web page - demofieldtable.php

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demofieldtable.php, in directory, c12mysqlapi.
This web page includes the web page, userfunctions.php, by the statement,

```
require ('userfunctions.php');
```

The functions, connect(), selectdb() and query(), in userfunctions.php are called in demofieldtable.php.

(Beginning of web page)

```
<?php
//demofieldtable.php
require ('userfunctions.php');
//connect
$conn = connect();
//select db
$dbname = "webdb";
selectdb($dbname, $conn);
//sql command from form
//@ suppresses Notice: Undefined index
//   when "sqlcommand" is null
$sql = @$_POST["sqlcommand"];
echo "<br>SQL command: $sql";
//query
$result = query($sql, $conn);
//table name
$table = mysqli_field_table($result, 0);
if ($table == FALSE)
    {
     echo "<h3>Table name: <font color=red>(none)</font><br>";
     echo "Use a SELECT command</h3><br>";
    }
else
     echo "<h3>Table name: <font
color=red>$table</font></h3><br>";
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, field_table.htm, in the menu. In the form, Table name, there is a textbox for a SQL select command.

This function returns the name of the table in a SELECT command which contains the first column in the SELECT command. For example, if column1 is in table1 and column2 is in table2, the function will return table1 for both of the following commands.

select column1, column2 from table1, table2
select column1, column2 from table2, table1

Only one table is used to demonstrate the function.

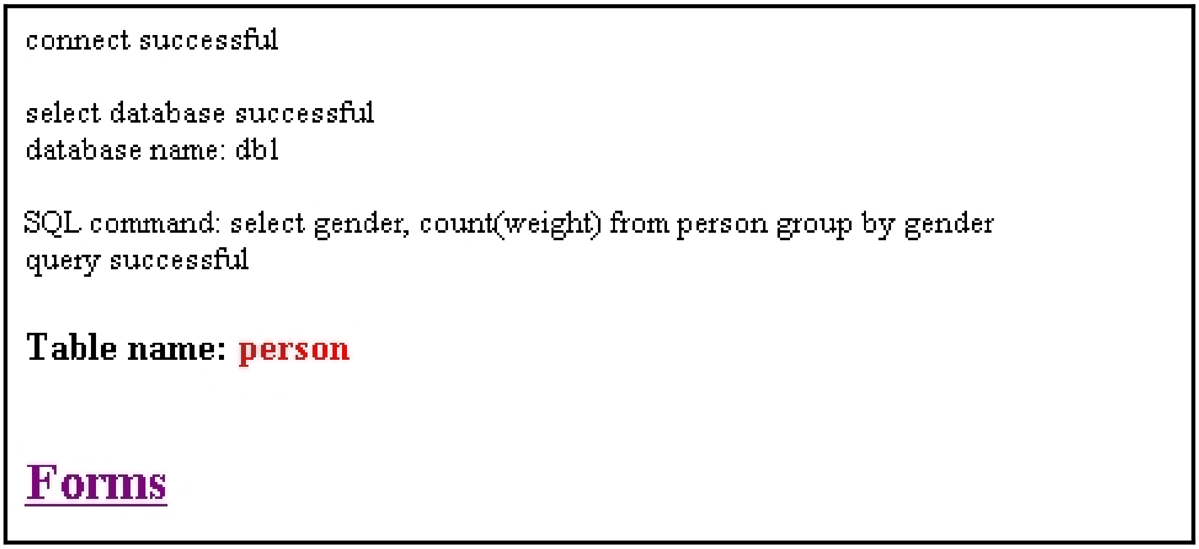
Type the following three commands one at a time, all of which are valid SELECT commands and then click, Submit. Click the link, Form, to return to the form web page after each command.

select * from Person

select gender from Person
select gender, count(weight) from Person group by gender

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.7 shows the output of the web page, demofieldtable.php, when a valid SQL select command is typed.



```
connect successful

select database successful
database name: db1

SQL command: select gender, count(weight) from person group by gender
query successful
```

Table name: person

Forms

Figure 12.7 Output: demofieldtable.php, for a valid SQL select command

12.5.8 Number of columns - mysqli_num_fields()

The function, mysqli_num_fields(), returns the number of columns in the result.

Web page - demonumfields.php

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demonumfields.php, in directory, c12mysqlapi..

This web page includes the web page, userfunctions.php, by the statement,
require ('userfunctions.php');
The functions, connect(), selectdb(), query() and contents(), in userfunctions.php are called in demonumfields.php.

(Beginning of web page)

```
<?php
//demonumfields.php
require ('userfunctions.php');
//connect
$conn = connect();
//select db
```

```
$dbname = "webdb";
selectdb($dbname, $conn);
//table or sql command from form
//@ suppresses Notice: Undefined index
//   when "table" is null
$t = @$_POST["table"];
$sql = @$_POST["sqlcommand"];
$f = $_POST["contents"];
if ($t != "" && $f == "total")
//total contents of table
  {
    $sql = "select * from $t";
    echo "<br>SQL command: $sql";
    $heading = "<u>Total table</u> Total number of columns";
  }
else if ($sql != "" && $f == "partial")
//partial contents of table
  {
    //sql command from form
    echo "<br>SQL command: $sql";
    $heading = "<u>Partial table</u> Number of columns selected";
  }
else
//both text boxes, table and sqlcommand, are blank
    $sql = "";
//query
$result = query($sql, $conn);
//table name
$table = mysqli_field_table($result, 0);
//number of columns
$fields = mysqli_num_fields($result);
echo "<h3>Table name: <font color=red>$table</font></h3><br>";
echo "<h1>$heading: <font color=red>$fields</font></h1>";
//contents of table
contents($result, $sql);
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, num_fields.htm, in the menu. In Number of columns, there are two forms. In the form, Total table, there is a textbox for the name of a table. In the form, Partial table, there is a textbox for a SQL command.

Form: Total table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

abc

in the textbox and then click, Submit. Assuming that abc is not the name of a table, an error message appears. Click, Back, in the browser.

Type

Person

in the textbox and then click, Submit. Assuming that Person is the name of a table in the database, there is no error. Note that the SQL command which was executed is, select * from Person. Click the link, Form, to return to the form web page.

Form: Partial table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

Person

in the textbox and then click, Submit. The command, Person, is not a valid command and so an error message appears. Click, Back, in the browser.

Type the following commands, all of which are valid, assuming table, Person, shown in Figure 12.1 exists. Click, Submit, after each command. Click the link, Form, to return to the form web page after each command.

```
select * from Person
select fName, gender from Person
select gender, count(fName) from Person group by gender
select gender, count(fName) as "Number of persons" from Person group
by gender
desc Person
show columns from Person
show databases
show tables
```

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.8 shows the output of the web page, demonumfields.php, when a valid SQL command is typed.

12.5.9 Column names - mysqli_field_name()

The function, mysqli_field_name(), returns the name of the next column in the result.

Web page - demofieldname.php

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demofieldname.php, in directory, c12mysqlapi.

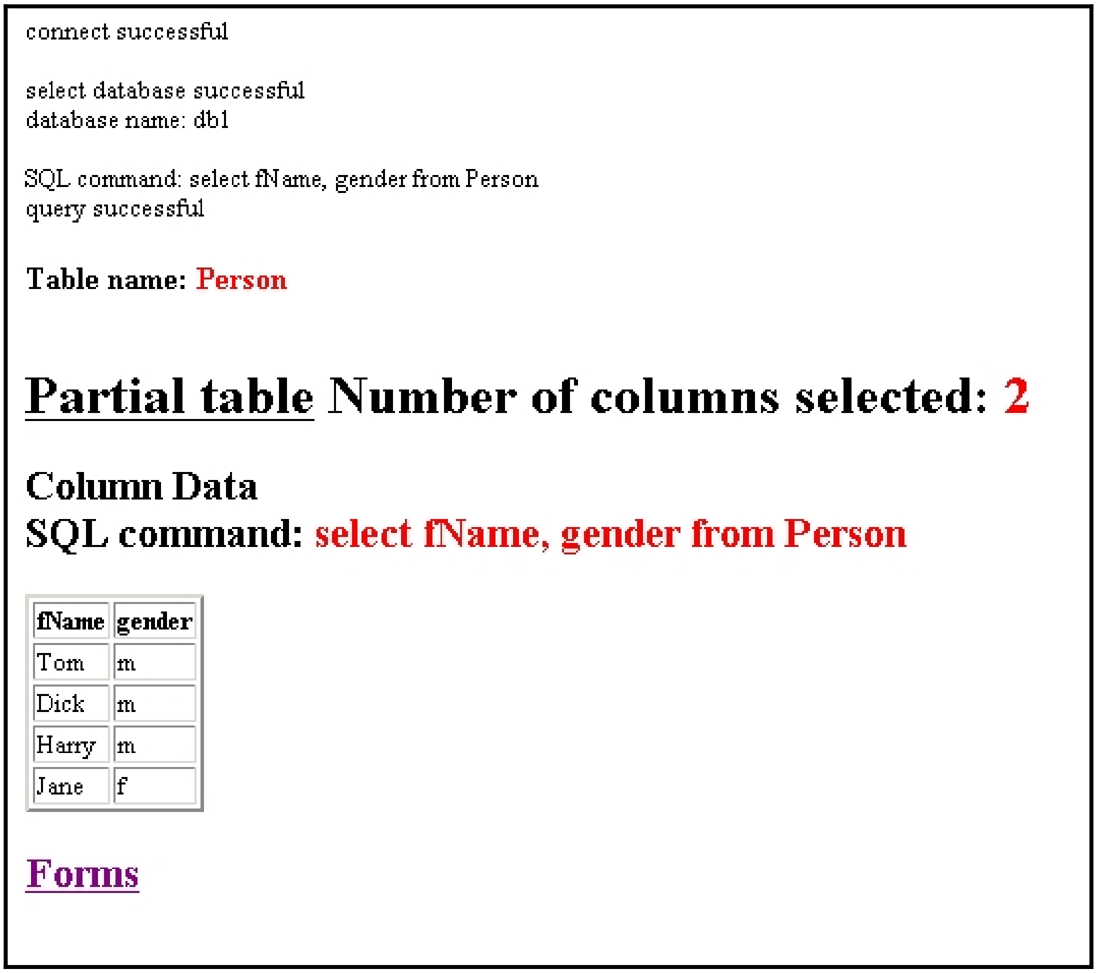
This web page includes the web page, userfunctions.php, by the statement,

```
require ('userfunctions.php');
```

The functions, connect(), selectdb(), query() and contents(), in userfunctions.php are called in demofieldname.php.

(Beginning of web page)

```
<?php
//demofieldname.php
require ('userfunctions.php');
//connect
$conn = connect();
//select db
$dbname = "webdb";
```



connect successful

select database successful
database name: db1

SQL command: select fName, gender from Person
query successful

Table name: Person

Partial table Number of columns selected: 2

Column Data
SQL command: select fName, gender from Person

fName	gender
Tom	m
Dick	m
Harry	m
Jane	f

Forms

Figure 12.8 Output: demonumfields.php, for a valid SQL command

```
selectdb($dbname, $conn);
//table or sql command from form
//@ suppresses Notice: Undefined index
//   when "table" is null
$t = @$_POST["table"];
$sql = @$_POST["sqlcommand"];
```

```
$f = $_POST["contents"];
if ($t != "" && $f == "total")
//total contents of table
  {
    $sql = "select * from $t";
    echo "<br>SQL command: $sql";
    $heading = "<u>Total table</u> Total number of columns";
  }
else if ($sql != "" && $f == "partial")
//partial contents of table
  {
    //sql command from form
    echo "<br>SQL command: $sql";
    $heading = "<u>Partial table</u> Number of columns selected";
  }
else
//both text boxes, table and sqlcommand, are blank
    $sql = "";
//query
$result = query($sql, $conn);
//table name
$table = mysqli_field_table($result, 0);
//number of columns
//$fields = mysqli_num_fields($result);
//number of columns
$n = mysqli_num_fields($result);
echo "<h3>Table: <font color=red>$table</font><br>";
echo "$heading: <font color=red>$n</font></h3><br>";
//column names
echo "<h1>Names of columns:<br>";
echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;";
for($i=0; $i<$n; $i++)
{
     $name = mysqli_field_name($result, $i);
     echo "$name &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;";
}
echo "</h1>";
//contents of table
contents($result, $sql);
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, field_name.htm, in the menu. In Column names, there are two forms. In the form, Total table, there is a textbox for the name of a table. In the form, Partial table, there is a textbox for a SQL command.

Form: Total table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

abc

in the textbox and then click, Submit. Assuming that abc is not the name of a table, an error message appears. Click, Back, in the browser.

Type

Person

in the textbox and then click, Submit. Assuming that Person is the name of a table in the database, there is no error. Note that the SQL command which was executed is, select * from Person. Click the link, Form, to return to the form web page.

Form: Partial table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

Person

in the textbox and then click, Submit. The command, Person, is not a valid command and so an error message appears. Click, Back, in the browser.

Type the following commands, all of which are valid, assuming table, Person, shown in Figure 12.1 exists. Click, Submit, after each command. Click the link, Form, to return to the form web page after each command.

```
select * from Person
select fName, gender from Person
select gender, count(fName) from Person group by gender
select gender, count(fName) as "Number of persons" from Person group
by gender
desc Person
show columns from Person
show databases
show tables
```

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.9 shows the output of the web page, demofieldname.php, when a valid SQL command is typed.

12.5.10 Number of rows by SELECT - mysqli_num_rows()

The function, mysqli_num_rows(), returns the number of rows by SELECT command.

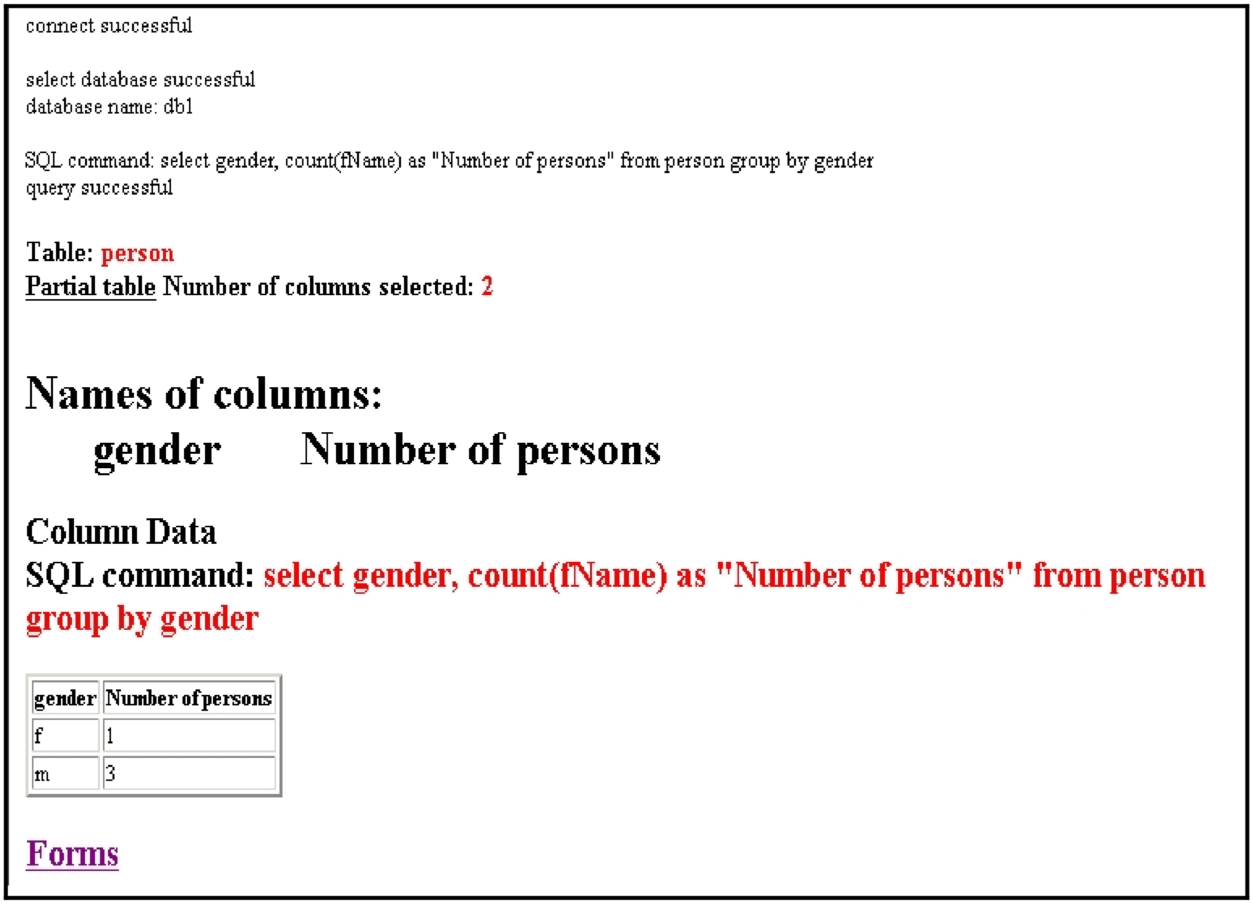
Web page - demonumrows.php

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demonumrows.php, in directory, c12mysqlapi.

This web page includes the web page, userfunctions.php, by the statement,

require ('userfunctions.php');

The functions, connect(), selectdb(), query() and contents(), in userfunctions.php are called in demonumrows.php.



```
connect successful

select database successful
database name: db1

SQL command: select gender, count(fName) as "Number of persons" from person group by gender
query successful
```

Table: person
Partial table **Number of columns selected: 2**

Names of columns:

gender Number of persons

Column Data
SQL command: select gender, count(fName) as "Number of persons" from person group by gender

gender	Number of persons
f	1
m	3

Forms

Figure 12.9 Output: demofieldname.php, for a valid SQL command

(Beginning of web page)

```
<?php
//demonumrows.php
require ("userfunctions.php");
//connect
$conn = connect();
//select db
$dbname = "webdb";
selectdb($dbname, $conn);
//table or sql command from form
//@ suppresses Notice: Undefined index
//   when "table" is null
$t = @$_POST["table"];
```

```
$sql = @$_POST["sqlcommand"];
$f = $_POST["contents"];
if ($t != "" && $f == "total")
//total contents of table
  {
    $sql = "select * from $t";
    echo "<br>SQL command: $sql";
    $headingcols = "<u>Total table</u> Total number of columns";
    $headingrows = "<u>Total table</u> Total number of rows";
  }
else if ($sql != "" && $f == "partial")
//partial contents of table
  {
    //sql command from form
    echo "<br>SQL command: $sql";
    $headingcols = "<u>Partial table</u> Number of columns
selected";
    $headingrows = "<u>Partial table</u> Number of rows
selected";
  }
else
//both text boxes, table and sqlcommand, are blank
    $sql = "";
//query
$result = query($sql, $conn);
//table name
$table = mysqli_field_table($result, 0);
//number of columns
$fields = mysqli_num_fields($result);
//number of rows
$rows = mysqli_num_rows($result);
echo "<h3>Table name: <font color=red>$table</font></h3><br>";
echo "<h1>$headingcols: <font color=red>$fields</font><br>";
echo "$headingrows: <font color=red>$rows</font></h1>";
//contents of table
contents($result, $sql);
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, num_rows.htm, in the menu. In Number of rows by SELECT, there are two forms. In the form, Total table, there is a textbox for the name of a table. In the form, Partial table, there is a textbox for a SQL select command.

Form: Total table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser. Type: abc
in the textbox and then click, Submit. Assuming that abc is not the name of a table, an error message appears.

Click, Back, in the browser and type: Person
in the textbox and then click, Submit. Assuming that Person is the name of a table in the database, there is no error. Note that the SQL command which was executed is, select * from Person. Click the link, Form, to return to the form web page.

Form: Partial table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser. Type Person
in the textbox and then click, Submit. The command, Person, is not a valid command and so an error message appears. Click, Back, in the browser.

Type the following commands, all of which are valid, assuming table, Person, shown in Figure 12.1 exists. Click, Submit, after each command. Click the link, Form, to return to the form web page after each command.

```
select fName, gender from Person
select * from from Person where gender = "f"
select gender,countfName)from Person group by gender
select gender, count(fName) as "Number of persons"
   from Person group by gender
desc Person
show columns from Person
show databases
show tables
```

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.10 shows the output of the web page, demonumrows.php, when a valid SQL command is typed.

12.5.11 Number of rows after updates - mysqli_affected_rows

The function, mysqli_affected_rows(), returns the number of affected rows by the last INSERT, UPDATE or DELETE command.

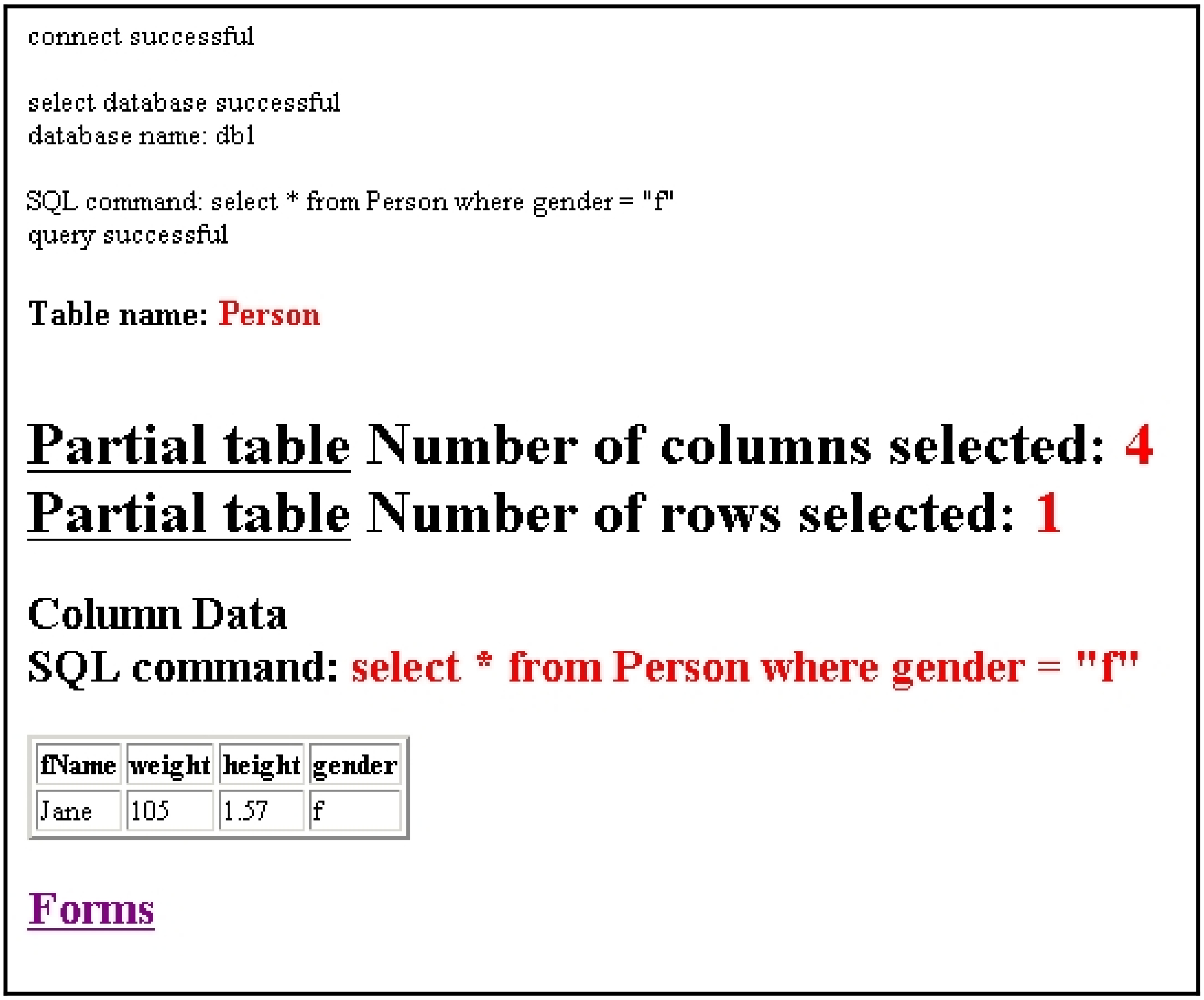
Web page - demoaffectedrows.php

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demoaffectedrows.php, in directory, c12mysqlapi.

This web page includes the web page, userfunctions.php, by the statement,

```
require ('userfunctions.php');
```

The functions, connect(), selectdb(), query() and contents(), in userfunctions.php are called in demoaffectedrows.php.



connect successful

select database successful
database name: db1

SQL command: select * from Person where gender = "f"
query successful

Table name: Person

Partial table Number of columns selected: 4
Partial table Number of rows selected: 1

Column Data
SQL command: select * from Person where gender = "f"

fName	weight	height	gender
Jane	105	1.57	f

Forms

Figure 12.10 Output: demonumrows.php, for a valid SQL command

(Beginning of web page)

```
<?php
//demoaffectedrows.php
require ('userfunctions.php');
//connect
$conn = connect();
//select db
$dbname = "webdb";
selectdb($dbname, $conn);
//sql command from form
$sql = $_POST["sqlcommand"];
echo "<br>SQL command: $sql";
$headingcols = "Number of columns selected";
$headingrows = "Number of rows selected";
```

```
//query
$result = query($sql, $conn);
//table name
$table = @mysqli_field_table($result, 0);
//number of columns
$fields = @mysqli_num_fields($result);
//number of rows
$rows = @mysqli_num_rows($result);
//number of affected rows
$affectedrows = mysqli_affected_rows($conn);
echo "<h3>Table name: <font color=red>$table</font><br>";
echo "$headingcols: <font color=red>$fields</font><br>";
echo "$headingrows: <font color=red>$rows</font><br>";
echo "Number of affected rows: <font
color=red>$affectedrows</font></h3>";
//contents of table
if ($table != FALSE)
     contents($result, $sql);
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, affected_rows.htm, in the menu. In the form, Number of affected rows by INSERT, UPDATE or DELETE, there is a textbox for a SQL insert, update or delete command.

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

```
select * from abc
```

in the textbox and then click, Submit. Assuming that abc is not the name of a table, an error message appears. Click, Back, in the browser.

Type the following commands, all of which are valid, assuming table, Person, shown in Figure 12.1 exists. Click, Submit, after each command. Click the link, Form, to return to the form web page after each command.

Display table

```
select * from Person
```

Insert row

```
insert into Person values ('Sue', 115, 1.62, 'x')
```

Display table

```
select * from Person
```

Update row

```
update Person set gender = 'f' where fname = 'Sue'
```

Display table

```
select * from Person
```

Delete row

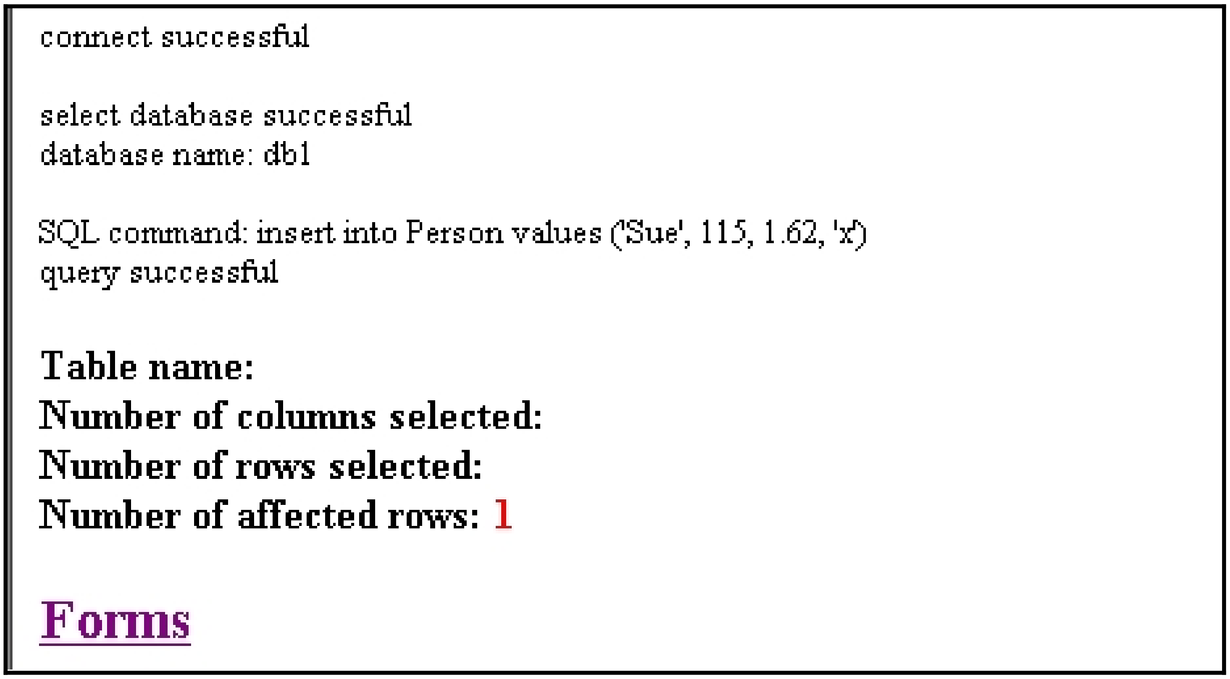
delete from Person where fname = 'Sue'

Display table

select * from Person

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.11 shows the output of the web page, demoaffectedrows.php, when a valid SQL insert command is typed.



```
connect successful

select database successful
database name: db1

SQL command: insert into Person values ('Sue', 115, 1.62, 'x')
query successful

Table name:
Number of columns selected:
Number of rows selected:
Number of affected rows: 1

Forms
```

Figure 12.11 Output: demoaffectedrows.php, for a valid insert SQL

Column properties - mysqli_fetch_field()

The function, mysqli_fetch_field(), returns the properties of columns in the result.

Web page - demofetchfield.php

Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demofetchfield.php, in directory, c12mysqlapi.

This web page includes the web page, userfunctions.php, by the statement,

require ('userfunctions.php');

The functions, connect(), selectdb(), query() and contents(), in userfunctions.php are called in demofetchfield.php.

(Beginning of web page)

```
<?php
//demofetchfield.php
```

```
require ('userfunctions.php');
//connect
$conn = connect();
//select db
$dbname = "webdb";
selectdb($dbname, $conn);
//table or sql command from form
//@ suppresses Notice: Undefined index
//   when "table" is null
$t = @$_POST["table"];
$sql = @$_POST["sqlcommand"];
$f = $_POST["contents"];
if ($t != "" && $f == "total")
//total contents of table
  {
    $sql = "select * from $t";
    echo "<br>SQL command: $sql";
    $headingcols = "<u>Total table</u> Total number of columns";
  }
else if ($sql != "" && $f == "partial")
//partial contents of table
  {
    //sql command from form
    echo "<br>SQL command: $sql";
    $headingcols = "<u>Partial table</u> Number of columns
selected";
  }
else
//both text boxes, table and sqlcommand, are blank
    $sql = "";
//query
$result = query($sql, $conn);
//table name
$table = mysqli_field_table($result, 0);
//number of columns
$fields = mysqli_num_fields($result);
//number of columns
$n = mysqli_num_fields($result);
echo "<h3>Table: <font color=red>$table</font><br>";
echo "$headingcols: <font color=red>$n</font></h3><br>";
//column properties
$i = 0;
$n = mysqli_num_fields($result);        //number of columns
echo "<table border=1>";
echo "<tr>";
echo "<h2>Column Properties<h2>";
//column names
```

```
echo "
<th>Column</th>
<th>name</th>
<th>table</th>
<th>type</th>
<th>max_length</th>
<th>multiple_key</th>
<th>not_null</th>
<th>numeric</th>
<th>primary_key</th>
<th>unique_key</th>
<th>unsigned</th>
<th>blob</th>
<th>zerofill</th>";
echo "</tr>";
while ($i < $n) {
   $prop = mysqli_fetch_field($result, $i);
   if (!$prop) {
       echo "No information available<br>";
   }
//data
echo "<tr>
<td>$i</td>
<td>$prop->name</td>
<td>$prop->table</td>
<td>$prop->type</td>
<td>$prop->max_length</td>
<td>$prop->multiple_key</td>
<td>$prop->not_null</td>
<td>$prop->numeric</td>
<td>$prop->primary_key</td>
<td>$prop->unique_key</td>
<td>$prop->unsigned</td>
<td>$prop->blob</td>
<td>$prop->zerofill</td>
</tr>";
$i++;
}
echo "</table>";
//contents of table
contents($result, $sql);
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, fetch_field.htm, in the menu. In Column properties, there are two forms. In the form, Total table, there is a textbox for the name of a table. In the form, Partial table, there is a textbox for a SQL command.

Form: Total table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

```
abc
```

in the textbox and then click, Submit. Assuming that abc is not the name of a table, an error message appears. Click, Back, in the browser.

Type

```
Person
```

in the textbox and then click, Submit. Assuming that Person is the name of a table in the database, there is no error. Note that the SQL command which was executed is, select * from Person. Click the link, Form, to return to the form web page.

Form: Partial table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type

```
Person
```

in the textbox and then click, Submit. The command, Person, is not a valid command and so an error message appears. Click, Back, in the browser.

Type the following commands, all of which are valid, assuming table, Person, shown in Figure 12.1 exists. Click, Submit, after each command. Click the link, Form, to return to the form web page after each command.

```
select * from Person
select fName, gender from Person
select gender, count(fName) from Person group by gender
select gender, count(fName) as "Number of persons" from Person group
by gender
desc Person
show columns from Person
show databases
show tables
```

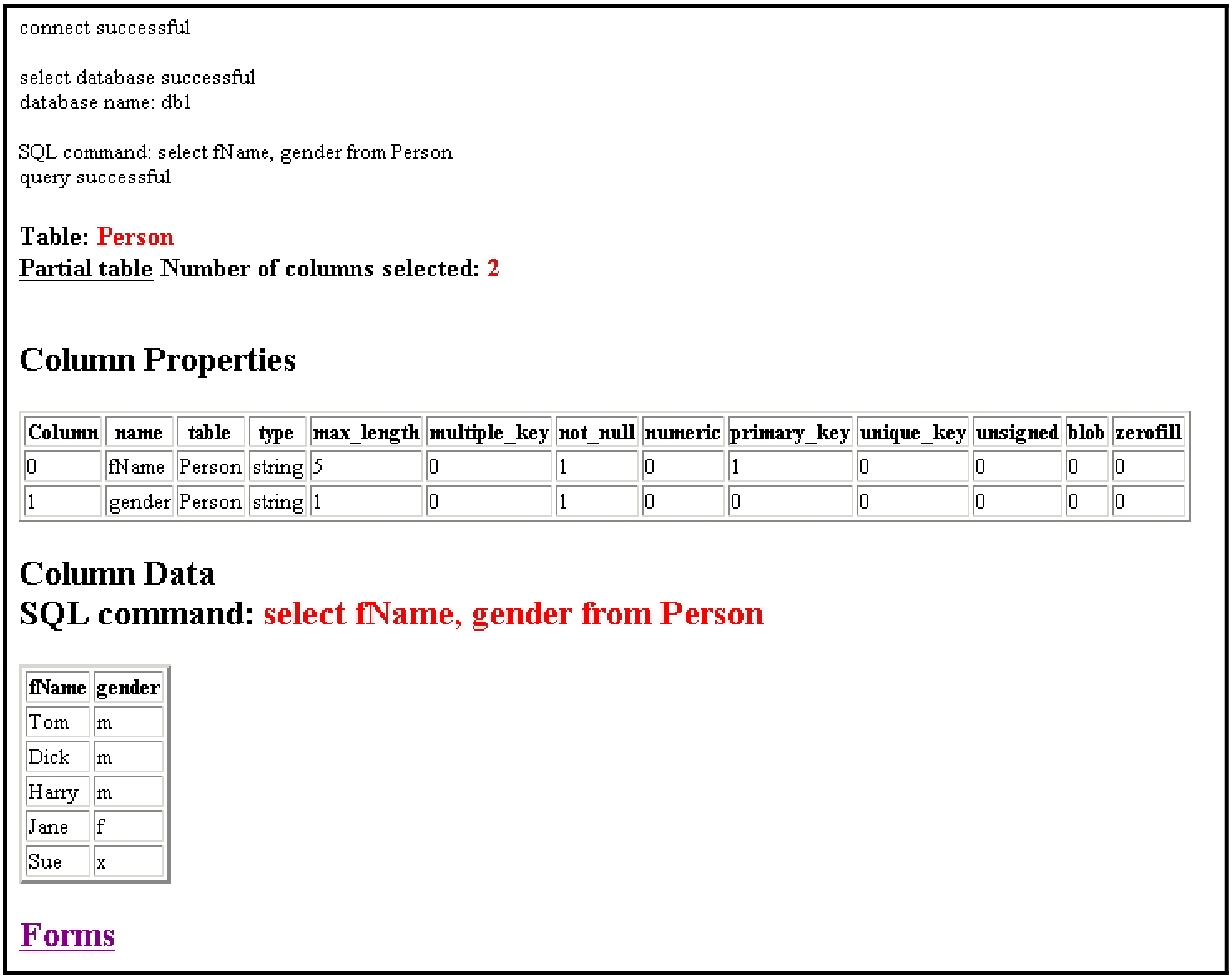
Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.12 shows the output of the web page, demofetchfield.php, when a valid SQL command is typed.

12.5.12 Column flags - mysqli_field_flags()

The function, mysqli_field_flags(), returns the flags of columns in the result.

Web page - demofieldflags.php
Type the following file, or copy and paste it, in Notepad or other text editor and save it using the filename, demofieldflags.php, in directory, c12mysqlapi.



connect successful

select database successful
database name: db1

SQL command: select fName, gender from Person
query successful

Table: Person
Partial table **Number of columns selected: 2**

Column Properties

Column	name	table	type	max_length	multiple_key	not_null	numeric	primary_key	unique_key	unsigned	blob	zerofill
0	fName	Person	string	5	0	1	0	1	0	0	0	0
1	gender	Person	string	1	0	1	0	0	0	0	0	0

Column Data
SQL command: select fName, gender from Person

fName	gender
Tom	m
Dick	m
Harry	m
Jane	f
Sue	x

Forms

Figure 12.12 Output: demofetchfield.php, for a valid SQL command

This web page includes the web page, userfunctions.php, by the statement,
require ('userfunctions.php');
The functions, connect(), selectdb(), query() and contents(), in userfunctions.php are called in demofieldflags.php.

(Beginning of web page)

```
<?php
//demofieldflags.php
require ('userfunctions.php');
//connect
$conn = connect();
//select db
$dbname = "webdb";
selectdb($dbname, $conn);
```

```
//table or sql command from form
//@ suppresses Notice: Undefined index
//   when "table" is null
$t = @$_POST["table"];
$sql = @$_POST["sqlcommand"];
$f = $_POST["contents"];
if ($t != "" && $f == "total")
//total contents of table
  {
    $sql = "select * from $t";
    echo "<br>SQL command: $sql";
    $headingcols = "<u>Total table</u> Total number of columns";
  }
else if ($sql != "" && $f == "partial")
//partial contents of table
  {
    //sql command from form
    echo "<br>SQL command: $sql";
    $headingcols = "<u>Partial table</u> Number of columns
selected";
  }
else
//both text boxes, table and sqlcommand, are blank
    $sql = "";
//query
$result = query($sql, $conn);
//table name
$table = mysqli_field_table($result, 0);
//number of columns
//$fields = mysqli_num_fields($result);
//number of columns
$n = mysqli_num_fields($result);
echo "<h3>Table: <font color=red>$table</font><br>";
echo "$headingcols: <font color=red>$n</font><br>";
// get column flags
echo "Column flags:</h3>";
for($i=0; $i<$n; $i++)
{
   $name = mysqli_field_name($result, $i);
   $j = $i+1;
   echo "$j. Column name: <font color=red>$name</font><br>";
   $flags = mysqli_field_flags($result, $i);
   if ($flags == "")
     {
        echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<b>NO
FLAGS</b><br>";
     }
   else
     {
        echo "&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<b>FLAGS:
$flags</b>";
        $a = explode(' ', $flags);
        echo "<pre>";
        echo "Column flags displayed by print_r(explode(' ', \
$flags)):<br>";
        print_r($a);
           echo "</pre>";
     }
}
//contents of table
contents($result, $sql);
echo "<h2><a href=\"formsch12.htm\">Forms</a></h2>";
?>
```

(End of web page)

Procedure

Click the link, field_flags.htm, in the menu. In Column flags, there are two forms. In the form, Total table, there is a textbox for the name of a table. In the form, Partial table, there is a textbox for a SQL command.

Form: Total table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type abc

in the textbox and then click, Submit. Assuming that abc is not the name of a table, an error message appears. Click, Back, in the browser.

Type Person

in the textbox and then click, Submit. Assuming that Person is the name of a table in the database, there is no error. Note that the SQL command which was executed is, select * from Person. Click the link, Form, to return to the form web page.

Form: Partial table

With the textbox blank, click, Submit. An error message appears. Click, Back, in the browser.

Type Person

in the textbox and then click, Submit. The command, Person, is not a valid command and so an error message appears. Click, Back, in the browser.

Type the following commands, all of which are valid, assuming table, Person, shown in Figure 12.1 exists. Click, Submit, after each command. Click the link, Form, to return to the form web page after each command

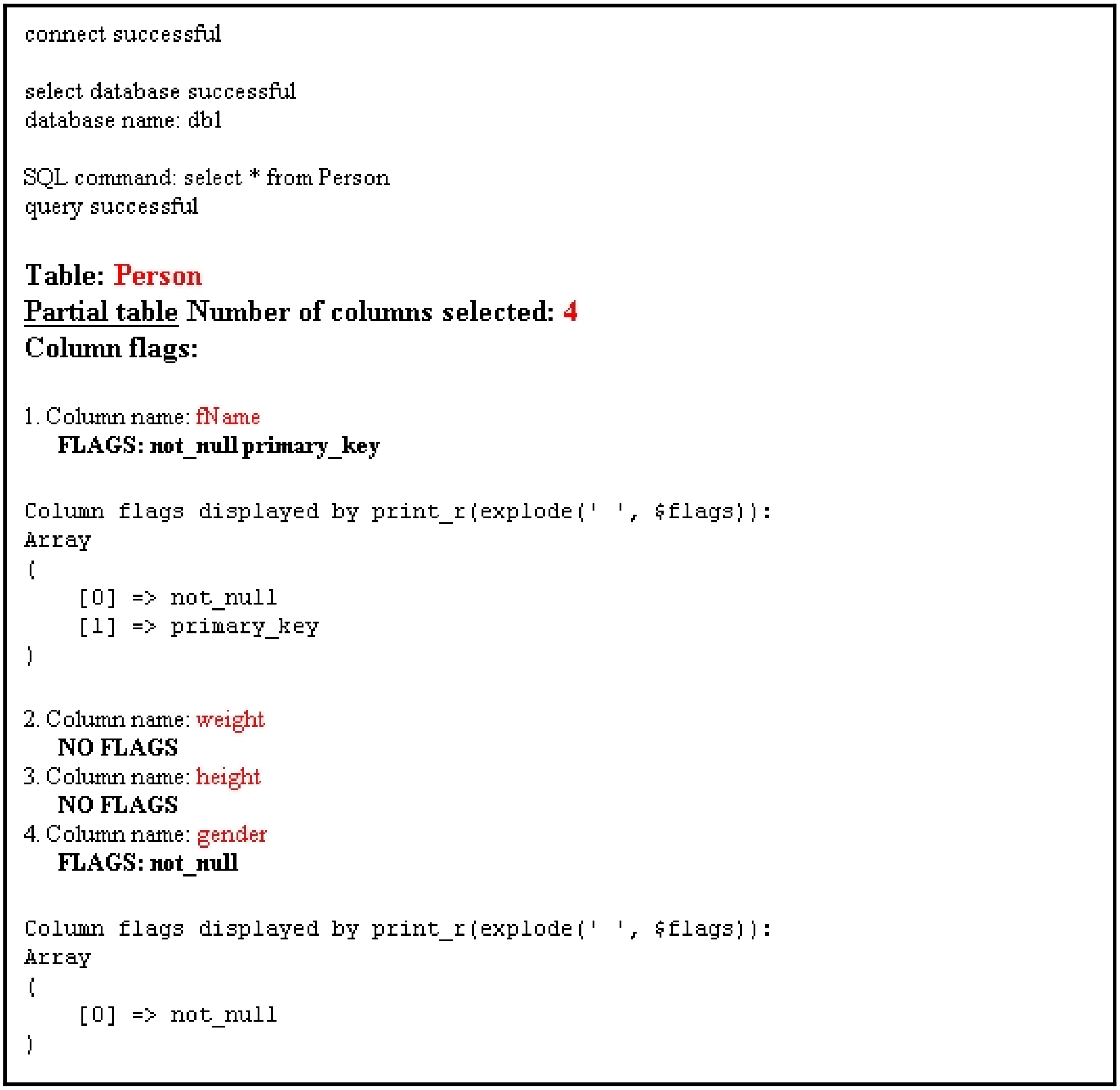
.

select * from Person

select fName, gender from Person
select gender, count(fName) from Person group by gender
select gender, count(fName) as "Number of persons" from Person group by gender
desc Person
show columns from Person
show databases
show tables

Click the link, Form, to return to the form web page. Click the link, Menu, to return to the menu web page.

Figure 12.13 shows the output of the web page, demofieldflags.php, when a valid SQL command is typed.



connect successful

select database successful
database name: db1

SQL command: select * from Person
query successful

Table: Person
Partial table Number of columns selected: 4
Column flags:

1. Column name: fName
   FLAGS: not_null primary_key

```
Column flags displayed by print_r(explode(' ', $flags)):
Array
(
    [0] => not_null
    [1] => primary_key
)
```

2. Column name: weight
   NO FLAGS
3. Column name: height
   NO FLAGS
4. Column name: gender
   FLAGS: not_null

```
Column flags displayed by print_r(explode(' ', $flags)):
Array
(
    [0] => not_null
)
```

Figure 12.13 Output of web page, demofieldflags.php, for a valid SQL command

13 Applications

13.1 Introduction

An application is a program or group of programs designed for users. The user of a well-designed application requires relatively little computer expertise and, in particular, does not need to be a programmer. Application software interacts with system software (operating system, for example) which in turns interacts with the computer hardware.

The database web application in this chapter uses the database in Chapter 5. The database is accessed using forms in a web page and MySQL functions in PHP. The PHP functions which access a MySQL database are covered in the chapter, MySQL API.

Any user with a computer that has a browser and an internet connection can access a database web application. The data in some databases of an application is confidential and so it is necessary to implement security to limit the access of users to the application.

Security in the application in this chapter uses two tables, which are added to the database. A database application which restricts users requires that users logon to the application. In general, users will either have no access (logon not permitted), have partial access or have complete access to the database.

HTTP (hypertext transfer protocol), which is the set of methods used to transfer web pages from one computer to another computer, is stateless. Stateless means that data supplied by the user is not stored after each time a user requests a web page in the same application. In particular, the identity of the user will not be remembered after the first time a web page is requested and so the user would have to logon each time a new web page is requested.

There is more than one way to maintain state, that is, to pass data from one web page to another. In the database application in this chapter, state is maintained by hidden variables (in a form) and by URL rewriting (in a link and a frame). Hidden variables and URL rewriting are used in this chapter to demonstrate these methods. However, the best way to maintain state is by using a session. All methods to maintain state are covered in a previous chapter.

13.2 Security

Database security limits the access of users to a database. There are two parts to database security: authentication and authorization. Authentication determines if a user has access or does not have access to a database. Authorization is done after authentication if a user has access to a database. Authorization determines the extent of access to a database which is allowed for a user.

13.2.1 Authentication

A user of a database is authenticated by identifying the user. Form validation is used. The user types credentials on a form on a web page and then submits the web page. The credentials are the username and password. The username and password of all valid users are stored in a table in the database, called ValidUser in the application in this chapter. The term, logon, is usually used for the process of authentication. After a user logs on, the user is presented with a graphical interface to the database and has access to the database.

13.2.2 Authorization

After a user is authenticated, the user of a database is authorized to have limited access to the database. The limited access which is given to a user is based on the role of a user. For example, a user with least access to a database may be allowed to only display (read) parts of some tables. At the other extreme, there must be at least one user, often called the database administrator, who has no limits on access (has complete access) to the database, and therefore can change data in all tables and change the structure of all tables in the database. The different roles of users are stored in another table in the database, called Role in the application in this chapter.

13.2.3 Tables

One table in the database is needed for authentication. This table should be called, ValidUser, and have at least the following attributes, where the primary key is underlined. The abbreviation, vUID, stands for, valid user identification.

Attributes of ValidUser: vUID, fName, lName, username, password, dateAdded

One table in the database is needed for authorization. This table should be called, Role, and have the following attributes, where the primary key is underlined. The abbreviation, rID, stands for, role identification.

Attributes of Role: rID, rName, rDescription

There are two possibilities for the relationship between the two tables. The roles could be constructed so that each user has only one role. Each role has many users. So in this case the relationship is many-to-one where ValidUser is on the many side of the relationship and therefore has the foreign key, rID, which is the primary key of Role. The schemas of the two tables follow. The foreign key is denoted by, fk -> *tablename,* in brackets where the foreign key, fk, references the primary key in *tablename*.

ValidUser(vUID, rID(fk 🗎 Role), fName, lName, username, password, dateAdded)
Role(rID, rName, rDescription)

Alternatively, the roles could be constructed so that each user could have more than one role. In this case the relationship is many-to-many and a third table is need to store the roles for each user, with attributes, vUID and rID. The third table could be called, HasAccess, and have other attributes, for example, the date when the access is granted, dateGranted. In HasAccess, vUID and rID are together the primary key, vUID is also a foreign key which references the primary key of ValidUser and rID is also a foreign key which references the primary key of Role. The schemas of the three tables follow. The foreign keys are denoted by, fk -> *tablename,* in brackets where the foreign key, fk, references the primary key in *tablename*.

ValidUser(vUID, fName, lName, username, password, dateAdded)
HasAccess(vUID(fk 🗎 ValidUser), rID(fk 🗎 Role), dateGranted)
Role(rID, rName, rDescription)

13.3 State Maintenance

Two methods are used to maintain state in the application in this chapter: hidden variables and URL rewriting. All methods to maintain state are covered in a previous chapter and some information in this section is duplicated for the sake of completeness.

13.3.1 Hidden Variables

Hidden variables are elements in a form created by the tag, input, with the type, hidden. A hidden variable requires two attributes, name and value. There are no optional attributes for hidden variables. The complete tag for a hidden variable is

<input type=hidden name=*variablename* value=*variablevalue*>

A hidden variable is hidden from view, that is, it is not displayed by the browser on the screen. Furthermore, it cannot be ignored or changed by either the browser or the user. Only a script can change the values of hidden variables. When a form is submitted, the name-value pairs of the hidden variables are sent to the server along with the name-value pairs of all other elements in the form. The variables are stored by the server in the array, $_POST.

Hidden variables have many uses. In the application in this chapter, hidden variables are used to maintain state. State maintenance means to effectively give web pages a memory by sending information from one web page to another web page. For example, suppose webpage1 is a menu which can request several other web pages. This is the organization of menus in the application in this chapter. A user has logged on and has permission to view all the web pages in the menu. The user views the menu, webpage1, on the client computer and clicks, Submit, on a form which requests webpage2. The form on webpage1 must send a hidden variable to webpage2 which allows the user to view webpage2. In addition, if the user returns to webpage1, a form in webpage2 must send the same hidden variable back to webpage1 so that webpage1 can send it to the next web page requested by the user, webpage3, and so on.

13.3.2 URL Rewriting

A URL (uniform resource locator) is the address of a web page. URL rewriting is the appending of variables to the URL which is

(a) a link in a web page or
(b) the source attribute of a frame in a frameset.

The variables are sent to the server and stored in the array, $_GET.

Variables are appended as name-value pairs. A question mark, ?, separates the URL value from the first name-value pair. If there is more than one name-value pair, an ampersand, &, separates the name-value pairs. The ampersand has a special meaning in a web page: it inserts characters in a web page. Its special meaning must be removed by replacing it with its escape sequence, either &#38; or &amp;.

The variables are inserted into a link by the following syntax.

<a href = *URL* ? *name1=value1* &amp; *name2=value2* >Send</a>

The variables are inserted into a frame in the same way.

<frame src = *URL* ? *name1=value1* &amp; *name2=value2* >

The names, *name1* and *name2*, are strings. The values, *value1* and *value2*, are the values of variables. So if *value1* is the value of a variable with name, variable1, then *value1* is

<?php echo $variable1 ?>

13.3.3 Storage of state variables

A web page, webpage2, can be requested by another web page, webpage1, in three ways.

(a) click submit in a form in webpage1
(b) click a link in webpage1
(c) by the source attribute in a frame in webpage1

Method (a) sends the data in the form to the server where the data is stored in the array, $_POST. The other two methods, (b) and (c), send the data appended to the URL to the server, and the data is stored in the array, $_GET. So in general, the script in webpage2 that receives the data must determine if the data is in $_POST or $_GET. The request method, POST or GET, is stored in the server variable,

$_SERVER['REQUEST_METHOD']

At the beginning of the script in webpage2, use the following PHP code to store the state variables in local variables of the script.

```
<?php
//request method
if ($_SERVER['REQUEST_METHOD'] == 'POST')
{
   $var1 = $_POST['var1'];
   $var2 = $_POST['var2'];
   // other variables
}
else if ($_SERVER['REQUEST_METHOD'] == 'GET')
{
   $var1 = $_GET['var1'];
   $var2 = $_GET['var2'];
   // other variables
}
else
{
   echo "ERROR: Request method is not POST or GET!!!";
   exit;
}
?>
```

13.4 Database

The database is the same database as in the chapter, SQL Relationships. In addition the database includes two tables used to establish database security.

13.4.1 ER Diagram

The ER diagram is given in Figure 13.1 and consists of two unrelated parts: school objects and security objects. The part, school objects, is the ER diagram in Figure 5.1 which stores the data about the four objects (entities): Student, Course, Department, Professor. The part, security objects, stores the data about the valid users and their roles using two objects: ValidUser, Role. To simplify the database, each user has only one role so that the relationship between ValidUser and Role is many-to-one, instead of the general case where each user can have more than one role and the relationship is many-to-many.

The users in ValidUser include all of the students in Student and professors in Professor, as well as other users. However, to simply the database, there are no relationships between ValidUser and Student or Professor, as shown in Figure 13.1.

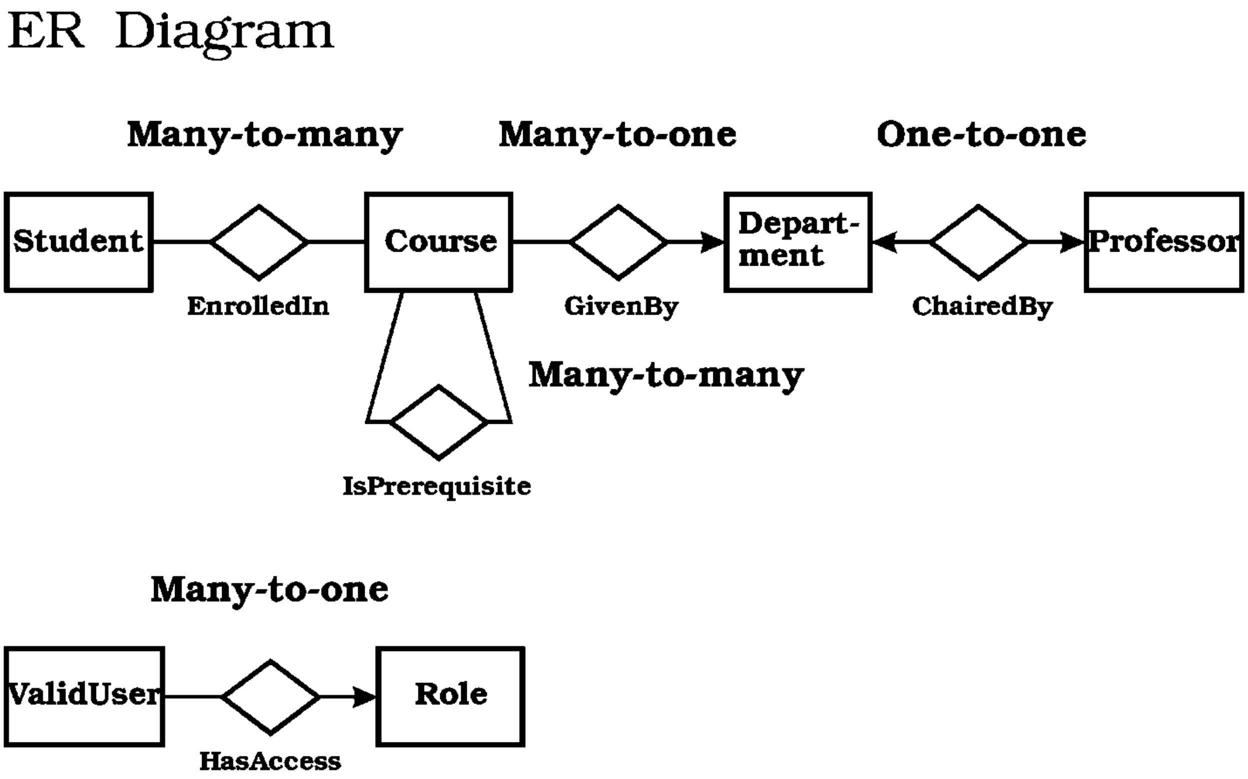


Figure 13.1 Database

13.4.2 Schema

The schema for the database follows. In the schema for each table, primary keys are underlined. Foreign keys are denoted by, fk -> *tablename,* in brackets where the foreign key, fk, references (->) the primary key in table with name, *tablename*. The constraint, UNIQUE, is applied to a foreign key to establish a one-to-one relationship, and is denoted by, unique, in brackets.

Schemas for school objects:

Course(cNumber, dCode(fk 🖹 Department), cName, credits)
Department(dCode, pID(fk 🖹 Professor, unique), dName, location)
Student(sID, sName, gender, major)

Professor(pID, pFirstName, pLastName, dateOfBirth)
EnrolledIn(sID(fk 🗎 Student), cNumber(fk 🗎 Course), grade)
IsPrererequisite(course(fk 🗎 Course), prerequisite(fk 🗎 Course))

Schemas for security objects:
ValidUser(vUID, rID(fk 🗎 Role), fName, lName, username, password, dateAdded)
Role(rID, rName, rDescription)

13.4.3 Creation of Tables

The database is divided into two parts for the purpose of creating and populating the tables. The parts are school objects and security objects, which are described above. As mentioned above, there are no relationships between the entities in the two parts in order to simplify the database. Two scripts are used to create the database.

The part of the database which stores the data about the school objects consists of six tables which were developed in Chapter 5. The script to create the objects tables is ct_db.sql in Chapter 5. The script is reproduced below.

Script 13.1 Create database school tables: ct_db.sql

```
-- file name: ct_db.sql
-- create database with six tables
/* Two mysql built-in commands are used in this script.
print, \p: display the command, that is, all text to left of \p
clear, \c: terminate the command
The two commands together, \p\c, displays all text before the
commands,
which corresponds to command, echo, in a shell script.
*/
*******************************
***CREATION OF SCHOOL TABLES***
***Script: ct_db.sql       ***
******************************* \p\c
-- remove child tables first, then parent tables
DROP TABLE EnrolledIn \p;       -- child table
DROP TABLE IsPrerequisite \p;   -- child table
DROP TABLE Course \p;           -- child table, parent table
DROP TABLE Department \p;       -- child table, parent table
DROP TABLE Student \p;          -- parent table
DROP TABLE Professor \p;        -- parent table
-- create parent tables first, then child tables
CREATE TABLE Student (          -- parent table
   sID   DECIMAL(7),
   sName VARCHAR(20) NOT NULL,
   gender CHAR(1) NOT NULL CHECK (gender IN ('m', 'f')),
   major CHAR(4),
   CONSTRAINT StudentPK PRIMARY KEY(sID)) \p;
```

```
CREATE TABLE Professor (        -- parent table
  pID           DECIMAL(6),
  pFirstName    VARCHAR(20) NOT NULL,
  pLastName     VARCHAR(20) NOT NULL,
  dateOfBirth   date,
  CONSTRAINT ProfPK PRIMARY KEY(pID)) \p;
CREATE TABLE Department (       -- parent table, child table
  dCode         DECIMAL(3),
  pID           DECIMAL(6),
  dName         VARCHAR(30) NOT NULL,
  location      VARCHAR(30),
  CONSTRAINT DepartmentPK PRIMARY KEY(dCode),
  CONSTRAINT DeptFK FOREIGN KEY(pID) REFERENCES Professor(pid),
  CONSTRAINT DeptUK UNIQUE(pID), -- UNIQUE establishes
one-to-one relationship
  CONSTRAINT DepartmentNameUK UNIQUE (dName)) \p;
CREATE TABLE Course (           -- parent table, child table
  cNumber               CHAR(8),
  dCode         DECIMAL(3),
  cName         VARCHAR(30) NOT NULL,
  credits               DECIMAL(1) DEFAULT 3 NOT NULL CHECK
(credits IN (3,4)),
  CONSTRAINT CoursePK PRIMARY KEY(cNumber),
  CONSTRAINT CourseFK FOREIGN KEY(dCode) REFERENCES
Department(dCode),
  CONSTRAINT CourseNameUK UNIQUE (cName, dCode)) \p;
CREATE TABLE EnrolledIn (       -- child table
  sID           DECIMAL(7),
  cNumber               CHAR(8),
  grade         CHAR(2),
  CONSTRAINT EnrolledInPK PRIMARY KEY(sID, cNumber),
  CONSTRAINT EnrolledIn_StudFK FOREIGN KEY(sID) REFERENCES
Student(sID),
  CONSTRAINT EnrolledIn_CourFK FOREIGN KEY(cNumber) REFERENCES
Course(cNumber)) \p;
CREATE TABLE IsPrerequisite (   -- child table
  course                CHAR(8),
  prerequisite CHAR(8),
  CONSTRAINT IsPrerequisitePK PRIMARY KEY(course, prerequisite),
  CONSTRAINT IsPrerequisite_courseFK FOREIGN KEY(course)
REFERENCES Course(cNumber),
  CONSTRAINT IsPrerequisite_prereqFK FOREIGN KEY(prerequisite)
REFERENCES Course(cNumber)) \p;
-- display description of attributes of tables
DESC Course \p;
DESC Department \p;
DESC Student \p;
DESC Professor \p;
DESC EnrolledIn \p;
DESC IsPrerequisite \p;
```

The part of the database that implements security consists of two tables. The script to create the security tables is ct_security.sql. The script is given below.

Script 13.2 Create database security tables: ct_security.sql

```
-- file name: ct_security.sql
-- create database with two tables
/* Two mysql built-in commands are used in this script.
print, \p: display the command, that is, all text to left of \p
clear, \c: terminate the command
The two commands together, \p\c, displays all text before the
commands,
which corresponds to command, echo, in a shell script.
*/
********************************
***CREATION OF SECURITY TABLES***
***Script: ct_security.sql    ***
******************************** \p\c
-- remove child tables first, then parent tables
DROP TABLE ValidUser \p;        -- child table
DROP TABLE Role \p;             -- parent table
-- create parent tables first, then child tables
CREATE TABLE Role (             -- parent table
  rID          DECIMAL(2),
  rName        VARCHAR(10) NOT NULL,
  rDescription VARCHAR(128),
  CONSTRAINT RolePK PRIMARY KEY(rID)) \p;
CREATE TABLE ValidUser (        -- parent table
  vUID         DECIMAL(7),
  rID          DECIMAL(2) NOT NULL,
  fName        VARCHAR(20),
  lName        VARCHAR(20),
  username     VARCHAR(10) NOT NULL,
  password     VARCHAR(10) NOT NULL,
  dateAdded    VARCHAR(20),
  CONSTRAINT ValidUserPK PRIMARY KEY(vUID),
  CONSTRAINT ValidUserFK FOREIGN KEY(rID) REFERENCES Role(rID),
  CONSTRAINT ValidUserusernameUK UNIQUE(username)) \p;
-- display description of attributes of tables
DESC Role \p;
DESC ValidUser \p;
```

Transfer the two scripts to the server. Logon to the server and start mysql. At the mysql prompt, mysql>, execute the two scripts as follows. Do not type a semicolon (;) at the end of a source command.

```
mysql> source ct_db
mysql> source ct_security
```

13.4.4 Population of Tables

The database is also populated by two scripts, one script for the school objects and one for the security objects.

The script to populate the school objects tables is in_db.sql in Chapter 5. The script is reproduced below.

Script 13.3 Populate database school tables: in_db.sql

```
-- file name: in_db.sql
-- populate the six tables of the database
/* Two mysql built-in commands are used in this script.
print, \p: display the command, that is, all text to left of \p
clear, \c: terminate the command
The two commands together, \p\c, displays all text before the
commands,
which corresponds to command, echo, in a shell script.
*/
*********************************
***POPULATION OF SCHOOL TABLES***
***Script: in_db.sql         ***
********************************* \p\c
-- delete the contents of child tables first, then parent tables
DELETE FROM EnrolledIn \p;      -- child table
DELETE FROM IsPrerequisite \p;  -- child table
DELETE FROM Course \p;          -- child table, parent table
DELETE FROM Department \p;      -- child table, parent table
DELETE FROM Student \p;         -- parent table
DELETE FROM Professor \p;       -- parent table
-- populate parent tables first, than child tables
-- Student - parent table
INSERT INTO Student VALUES
     (1111222, 'Jerry', 'm', 'COMP'),
     (2222333, 'Elaine', 'f', 'PHYS'),
     (3333444, 'George', 'm', 'CHEM'),
     (4444555, 'Kramer', 'm', null) \p;
-- Professor - parent table
INSERT INTO Professor VALUES
     (123456, 'Jean', 'Charest', '1958-06-24'),
     (234567, 'Bernard', 'Landry', '1937-03-09'),
     (345678, 'Lucien', 'Bouchard', '1938-12-22'),
     (456789, 'Jacques', 'Parizeau', '1930-08-09'),
     (567890, 'Daniel', 'Johnson, Jr', null) \p;
-- Department - parent table and child table
INSERT INTO Department VALUES
     (10, 123456, 'Computer Science', 'Library Building'),
     (11, 345678, 'Physics', 'Science Building') \p;
-- Course - parent table and child table
INSERT INTO Course VALUES
     ('COMP228', 10, 'System Hardware', 3),
     ('COMP229', 10, 'System Software', 3),
     ('PHYS245', 11, 'Mechanics', 3),
     ('PHYS253', 11, 'Electricity & Magnetism', 3) \p;
-- EnrolledIn - child table
INSERT INTO EnrolledIn VALUES
     (1111222, 'COMP228', 'C+'),
     (1111222, 'COMP229', null),
     (4444555, 'COMP228', 'A+'),
     (2222333, 'PHYS245', 'B') \p;
-- IsPrerequisite - child table
INSERT INTO IsPrerequisite VALUES
     ('COMP229', 'COMP228'),
     ('PHYS245', 'COMP228'),
     ('PHYS253', 'PHYS245') \p;
-- display the contents of the tables
SELECT * FROM Course \p;
SELECT * FROM Department \p;
SELECT * FROM Student \p;
SELECT * FROM Professor \p;
SELECT * FROM EnrolledIn \p;
SELECT * FROM IsPrerequisite \p;
```

The script to populate the security tables is in_security.sql. The script is given below.

Script 13.4 Populate database security tables: ct_security.sql

```
-- file name: in_security.sql
-- populate two tables of the database
/* Two mysql built-in commands are used in this script.
print, \p: display the command, that is, all text to left of \p
clear, \c: terminate the command
The two commands together, \p\c, displays all text before the
commands,
which corresponds to command, echo, in a shell script.
*/
***********************************
***POPULATION OF SECURITY TABLES***
***Script: in_security.sql      ***
*********************************** \p\c
-- delete the contents of child tables first, then parent tables
```

```
DELETE FROM ValidUser \p;       -- child table
DELETE FROM Role \p;            -- parent table
-- populate parent tables first, than child tables
-- Role - parent table
INSERT INTO Role VALUES
   (1, 'dbadmin', 'no limitation on access to database'),
   (2, 'professor', 'display all tables except security tables,
change two tables: Student and EnrolledIn'),
   (3, 'student', 'display parts of some tables') \p;
-- ValidUser - child table
INSERT INTO ValidUser VALUES
   (7, 1, 'James', null, 'james', 'cat', null),
   (123456, 2, 'Jean', 'Charest', 'jean', 'catnap', null),
   (234567, 2, 'Bernard', 'Landry', 'bernard', 'dog', null),
   (345678, 2, 'Lucien', 'Bouchard', 'lucien', 'doghouse', null),
   (456789, 2, 'Jacques', 'Parizeau', 'jacques', 'dogdays',
null),
   (567890, 2, 'Daniel', 'Johnson, Jr', 'daniel', 'doggone',
null),
   (1111222, 3, 'Jerry', null, 'jerry', 'ela', null),
   (2222333, 3, 'Elaine', null, 'elaine', 'geo', null),
   (3333444, 3, 'George', null, 'george', 'kra', null),
   (4444555, 3, 'Kramer', null, 'kramer', 'jer', null) \p;
-- display the contents of the tables
SELECT * FROM Role \p;
SELECT * FROM ValidUser \p;
```

Transfer the two scripts to the server. Logon to the server and start mysql. Create the database tables as described above, if this has not been done already. At the mysql prompt, mysql>, execute the two scripts to populate the database tables as follows. Do not type a semicolon (;) at the end of a source command.

```
     mysql> @ in_db
     mysql> @ in_security
```

13.5 Roles

There are three roles (categories of users) for the database: database administrator (usually one user), professor (many users) and student (many users). There are professors and students in the database tables, Professor and Student, but anybody can have the role of professor or student since the table, ValidUser, is not related to tables, Professor and Student, and therefore there is no restriction on who can be a valid user.

The three roles are specified as follows.

13.5.1 Database administrator

The role, database administrator, has complete access to the database.

All SQL commands can be executed. So the contents of all tables can be changed.
To simplify the application, the DDL commands (SQL commands to change the structure of tables) are not included. The structure of a table must be changed by executing the DDL commands using the MySQL shell, mysql.

13.5.2 Professor

The role, professor, has limited access to the database.
Display contents of all tables except the following.
    ValidUser, Role - do not display
    Professor - do not display pID
Change (add, modify, remove) data in tables, Student and EnrolledIn.

An obvious extension of the database is to add the relationship, WorksIn, between Department and Professor. Assuming that each professor works in only one department, the relationship is many-to-one since a department employs many professors. This illustrates that the same two entities can be related in more than one way, ChairedBy and WorksIn in this case.

Another extension of the database is to add the relationship, TaughtBy, between Course and Professor. This relationship assumes that each course has only one section. The relationship is many-to-one since each course is given by one professor (only one section of each course) and each professor gives many courses (more than one course in the same academic term). With this extension, the role of professor would be further limited to changing Student and EnrolledIn only for the courses taught by the professor.

13.5.3 Student

The role, student, has limited access to the database.
Display parts of contents of some tables as follows.
    ValidUser - do not display
    Role - do not display
    Student - do not display
    EnrolledIn - display only rows for the student who logs on.
    IsPrerequisite - display all of table
    Course - display with dCode replaced by dName
    Department - display without dCode
    Professor - display only pFirstName and pLastName
Change data in no tables.

13.6 Web Pages

The application consists of 23 web pages which are assumed to be in the same directory.
The web pages are divided into six categories: framesets, menus, messages, images, functions and forms.
A list of the names of the web pages in each category follow.

(a) Framesets
    (1) index.html

- (2) afterlogon.php
- (3) thirdrowdisplay.php
- (4) thirdrowmodify.php
- (5) thirdrowmodify2.php

(b) Menus

- (6) menumain.php
- (7) menudisplay.php
- (8) menumodify.php

(c) Messages

- (9) title.php
- (10) welcome.htm
- (11) messagedisplay.htm
- (12) messagemodify.htm
- (13) messagemodify2.htm

(d) Images

- (14) erdiagram.jpg

(e) Functions

- (15) functions.php

(f) Forms

- (16) userlogon.php
- (17) password.php
- (18) displaycontents.php
- (19) displaystructure.php
- (20) displayconstraints.php
- (21) modifyadd.php
- (22) modifydelete.php
- (23) modifyupdate.php

A brief description of each web page follows. The order of the web pages which follows is the typical order in which the web pages appear on the screen as the application is used.

13.6.1 index.html (frameset)

This is the home page of the application which will be displayed when the application is started. This page contains a frameset with two frames. The first frame is the top row which contains the title of the application. The second frame is the bottom row contains the user logon form.

13.6.2 title.php (message)

This web page is in the first frame of index.html. This is the title of the application. It includes the date by using the PHP function, date().

13.6.3 userlogon.php (form)

This web page is in the second frame of index.html. This is the logon form. The user types the username and password, and the clicks, Submit. If the username or password or both are invalid, a message is displayed and the logon form is displayed again. If the username and password are valid, a submit button, Continue, is displayed which requests the web page, afterlogon.php.

13.6.4 afterlogon.php (frameset)

This page contains a frameset with three frames. The first frame is a row which contains the title of the application which is the web page, title.php; this is the same frame as the first frame in frameset, index.html. The second frame is a row which contains the main menu which is the web page, menumain.php. The third frame is a row which contains the web page, welcome.htm.

13.6.5 welcome.htm (message)

This is the first web page that appears in the third row frame, afterlogon.php, after clicking Continue in the logon form. The text is a brief description of the database. The image is the ER diagram of the database.

13.6.6 erdiagram.jpg (image)

This is an image of the ER diagram of the database which appears in the welcome.htm web page.

13.6.7 functions.php (functions)

Five functions are stored in this web page: connect(), selectdb(), query(), contents(), setrowpointer().

This page is included in web pages which access the database.

(1) connect()

The function, connect(), connects to the database, that is, establishes a communication path between the web server and the database server. It is necessary to connect to the database at the beginning of every web page which accesses the database because at the end of a script which accesses the database, disconnect from the database occurs automatically.

In contrast, the logon to the application (using the form in web page, userlogon.php) is done once and then the logon is maintained by the use of hidden variables or variables appended to URLs which are sent from one web page to the next.

(2) selectdb()

The function, selectdb(), selects a database.

(3) query()

The function, query(), executes a SQL command or other command.

(4) contents()

The function, contents(), displays the contents of a table.

(5) setrowpointer()

The function, setrowpointer(), sets the row number.

13.6.8 menumain.php (menu)

This page displays a menu with five items. Each item is the submit button of a form.

The two menu items on the left request the two submenus, the display menu and modify menu, which are displayed in the third row frame.

The third menu item, Logoff, requests the home page, index.html.

The fourth menu item, Change Password, requests the web page, password.php, and displays it in the third row frame.

The fifth menu item, Welcome, requests web page, welcome.htm, and displays it in the third row frame.

13.6.9 thirdrowdisplay.php (frameset)

This page is a frameset with two frames which are columns that are displayed in the third row. This page is displayed when Display is clicked in the main menu.

13.6.10 menudisplay.php (menu)

This page displays a menu with a maximum of three items: Contents, Structure, Constraints. This menu is displayed in the first column of the third row after Display is clicked in the main menu.

All items are displayed when the database administrator logs on. One item, Contents, is displayed when a professor or student logs on on.

13.6.11 messagedisplay.htm (message)

This page is a message that is displayed in the second column of the third row after Display is clicked in the main menu.

13.6.12 thirdrowmodify.php (frameset)

This page is a frameset with two frames which are columns that are displayed in the third row. This page is displayed when when a student has not logged on and Modify is clicked in the main menu.

13.6.13 thirdrowmodify2.php (frameset)

This page is a frameset with two frames which are columns that are displayed in the third row. This page is displayed when a student has logged on and Modify is clicked in the main menu.

13.6.14 menumodify.php (menu)

This page displays a menu with a maximum of three items: Add Row, Delete Row, Update Row.

All items are displayed when the database administrator logs on. One item, Contents, is displayed when a professor logs on on. No items are displayed when a student logs on and in this case a message informs the user that he or she is not authorized to modify tables.

13.6.15 messagemodify.htm (message)

This page is a message that is displayed in the second column of the third row after Modify is clicked in the main menu when a student is not logged on.

13.6.16 messagemodify2.htm (message)

This page is a message that is displayed in the second column of the third row after Modify is clicked in the main menu when a student is logged on.

13.6.17 password.php (form)

This page displays a form to change the password for the user who is logged on.
The name of the user is displayed. The user enters the new password twice and if both entries are the same, the password is changed in the database.

13.6.18 displaycontents.php (form)

The contents of one table is displayed by using a form. The entire contents of all tables are displayed for the database administrator. Partial contents of some tables are displayed for professors and students.

13.6.19 displaystructure.php (form)

The structure of one table is displayed by using a form.
Only the database administrator can display the structure of tables.

13.6.20 displayconstraints.php (form)

The constraints in one table are displayed by using a form.
Only the database administrator can display the constraints of tables

13.6.21 modifyadd.php (form)

One row is added to one table by using a form. The database administrator can add rows to all tables. A professor can add rows to only two of the tables. Students cannot add rows to any tables.

13.6.22 modifydelete.php (form)

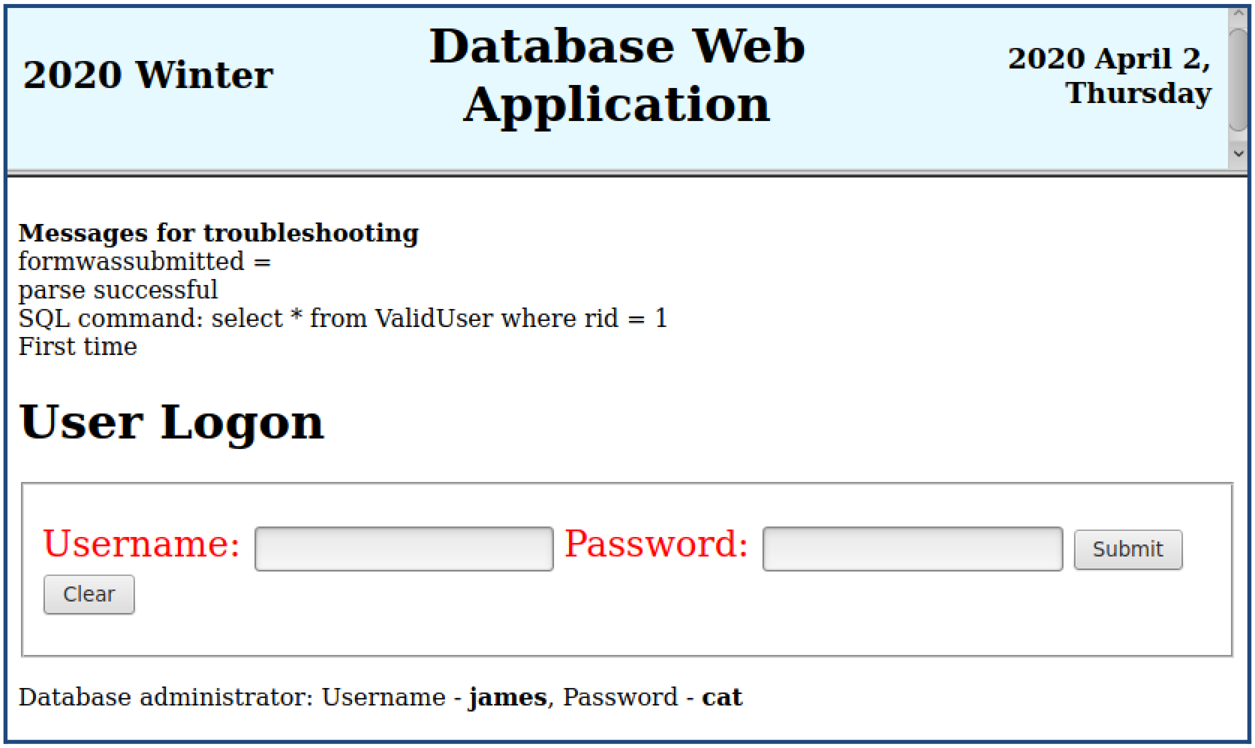
One row is deleted from one table by using a form. The database administrator can delete rows from all tables. A professor can delete rows from only two of the tables. Students cannot delete rows from any tables.

13.6.23 modifyupdate.php (form)

One or more rows is modified in one table by using a form. The database administrator can modify rows in all tables. A professor can modify rows from only two of the tables. Students cannot modify rows in any tables.

13.7 Programming Demonstrations

The first screen displayed after starting the application is shown in Figure 13.2. The screen is divided into two frames, which are the top row and bottom row. The title of the application is displayed in the top row and the logon form is displayed in the bottom row.



2020 Winter

Database Web Application

2020 April 2, Thursday

Messages for troubleshooting
formwassubmitted =
parse successful
SQL command: select * from ValidUser where rid = 1
First time

User Logon

Database administrator: Username - **james**, Password - **cat**

Figure 13.2 First screen after starting application

Type the username, james, and password, dog, and then click, Submit. The username, james, is the name of the database administrator but the password is incorrect. If the username or password or both are not valid, the form is displayed again with an error message.

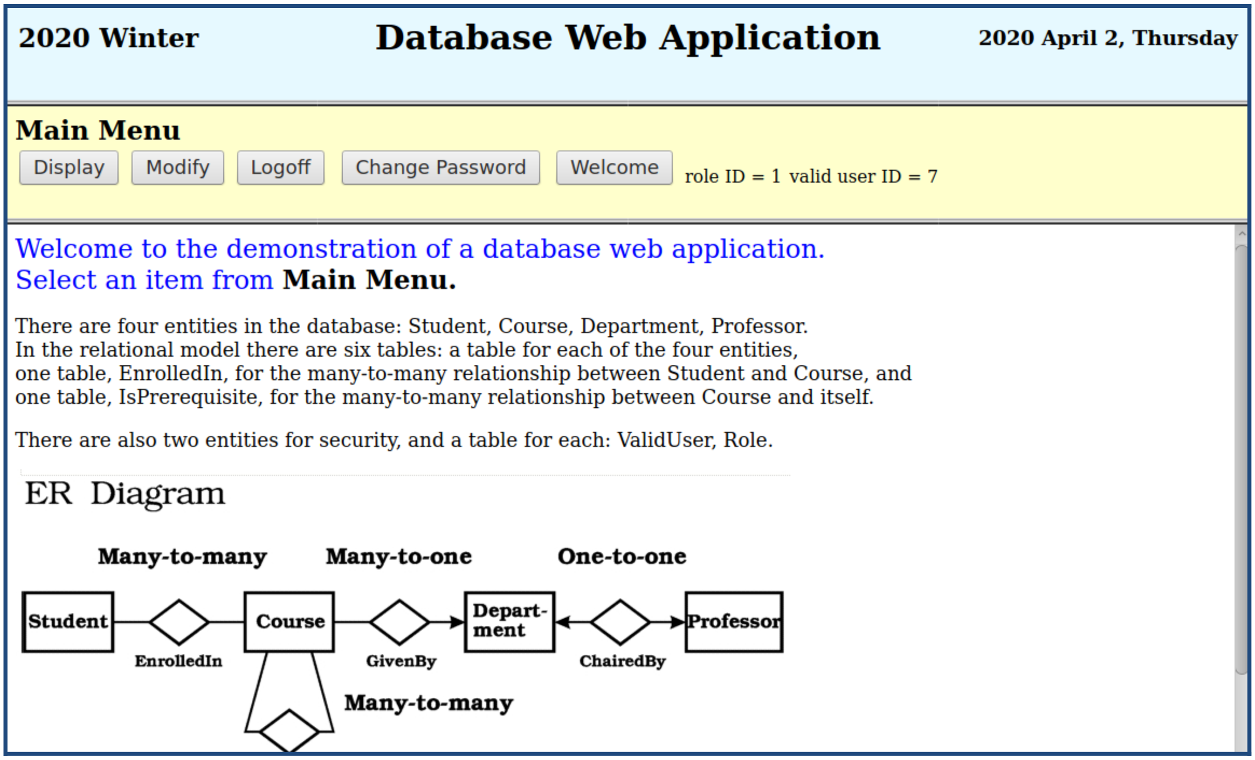
Type the username, james, and password, cat, and then click, Submit. The username, james, is the name of the database administrator and the password, cat, is correct. If both the username and password are valid, a message is displayed, and below the message a button, Continue.

After clicking Continue on the logon form, the first screen after logon appears which is shown in Figure 13.3. The screen is divided into three frames: top row (Database . . .), middle row (Main Menu) and bottom row (Welcome . . .).

Do all of the demonstrations by logging on as james, who has complete access to the database. Then logoff by clicking, Logoff, in the main menu. Logon as a professor and then a student and verify the restrictions for these roles which are given in Section 13.5. One of the professors has username, jean, and password, catnap. One of the students has username, jerry, and password, ela.

13.7.1 Display Contents

In the Main Menu click Display. The Display Menu is shown in Figure 13.4.



2020 Winter

Database Web Application

2020 April 2, Thursday

Main Menu

role ID = 1 valid user ID = 7

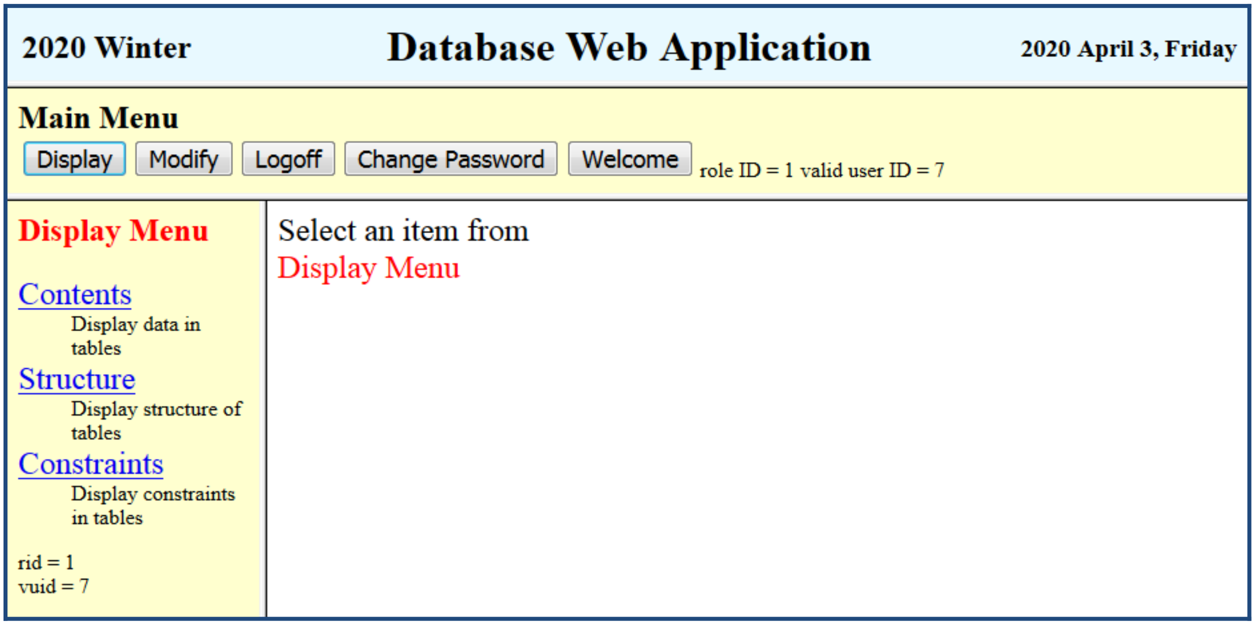
Welcome to the demonstration of a database web application.
Select an item from **Main Menu.**

There are four entities in the database: Student, Course, Department, Professor.
In the relational model there are six tables: a table for each of the four entities,
one table, EnrolledIn, for the many-to-many relationship between Student and Course, and
one table, IsPrerequisite, for the many-to-many relationship between Course and itself.

There are also two entities for security, and a table for each: ValidUser, Role.

ER Diagram

Figure 13.3 Part of first screen after logon



2020 Winter

Database Web Application

2020 April 3, Friday

Main Menu

role ID = 1 valid user ID = 7

Display Menu

Contents
Display data in tables

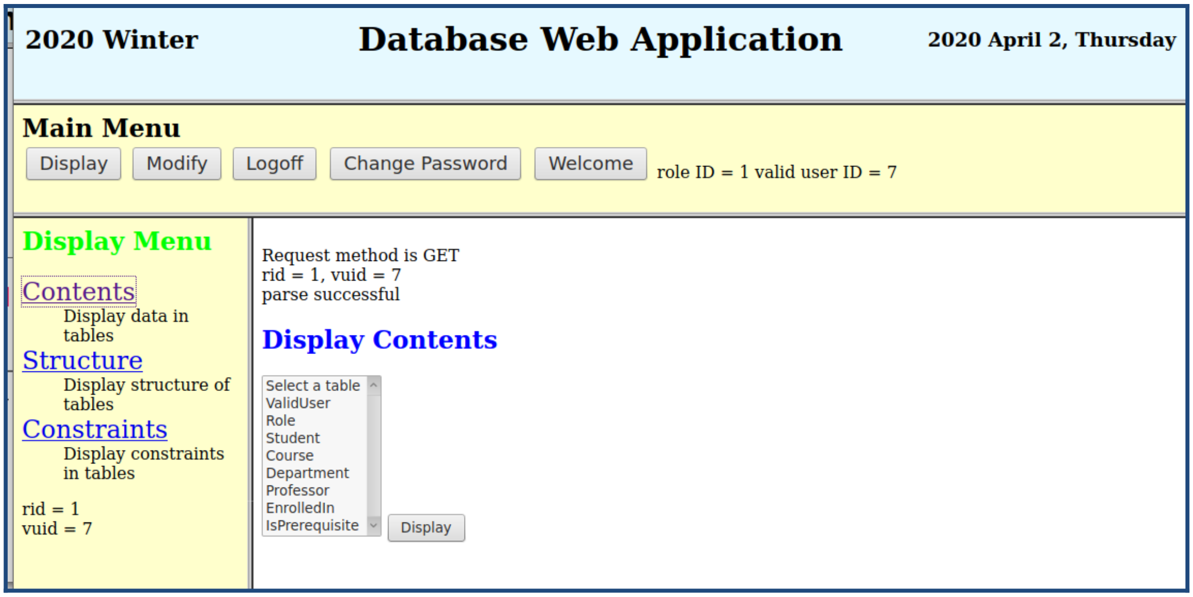
Structure
Display structure of tables

Constraints
Display constraints in tables

rid = 1
vuid = 7

Select an item from
Display Menu

Figure 13.4 Click: Display, in Main Menu



2020 Winter

Database Web Application

2020 April 2, Thursday

Main Menu

role ID = 1 valid user ID = 7

Display Menu

Request method is GET
rid = 1, vuid = 7
parse successful

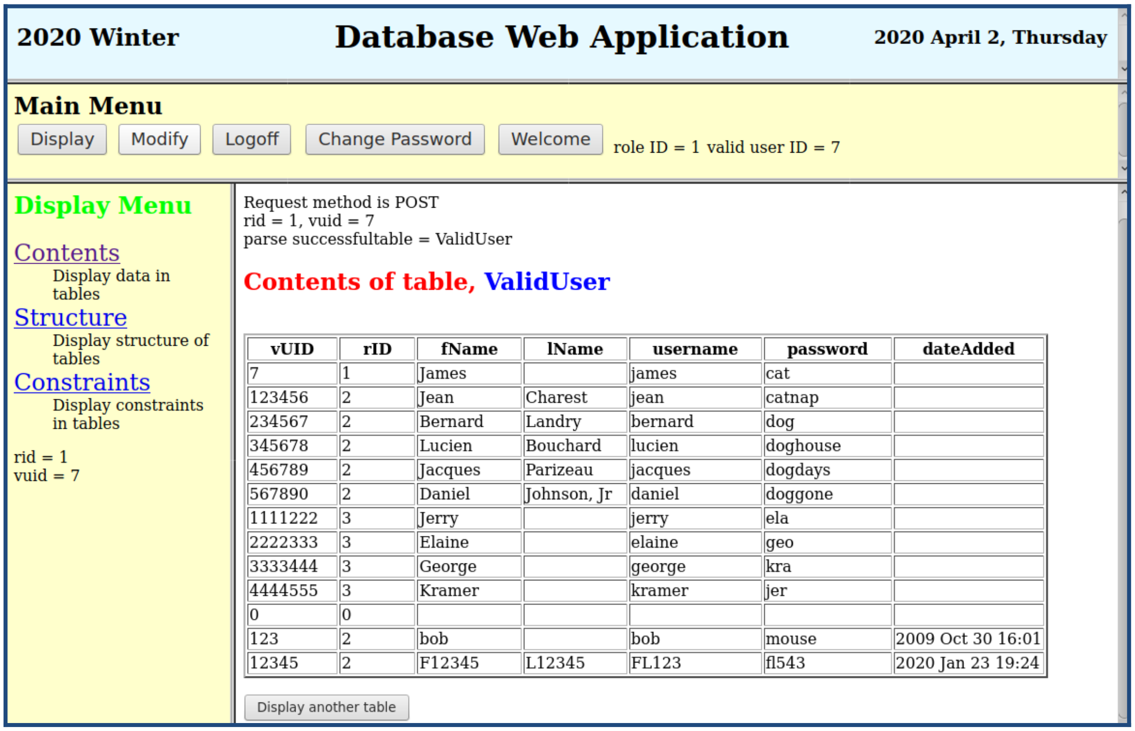
Display Contents

rid = 1
vuid = 7

Figure 13.5 Click: Contents, in Display Menu

In the Display Menu click Contents. The form for Contents is shown in Figure 13.5. Note that the Display Menu is unchanged.

In the form, Display Contents, click ValidUser and then click Display. The contents of table, ValidUser, is displayed as shown in Figure 13.6. This table contains the usernames and passwords of all valid users of the database.



2020 Winter

Database Web Application

2020 April 2, Thursday

Main Menu

role ID = 1 valid user ID = 7

Display Menu

Request method is POST
rid = 1, vuid = 7
parse successfultable = ValidUser

Contents of table, ValidUser

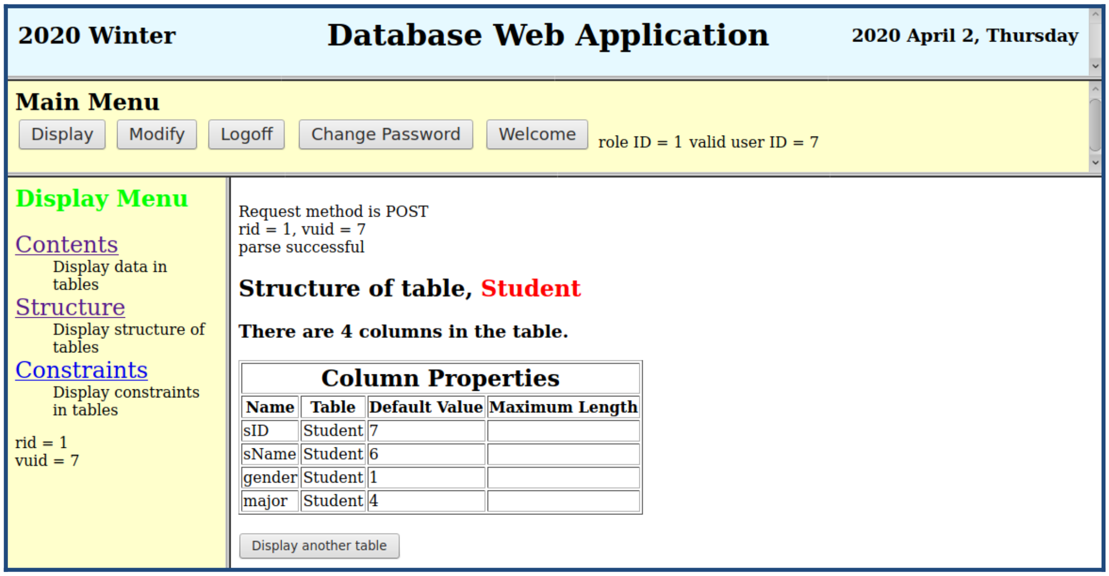
vUID	rID	fName	lName	username	password	dateAdded
7	1	James		james	cat	
123456	2	Jean	Charest	jean	catnap	
234567	2	Bernard	Landry	bernard	dog	
345678	2	Lucien	Bouchard	lucien	doghouse	
456789	2	Jacques	Parizeau	jacques	dogdays	
567890	2	Daniel	Johnson, Jr	daniel	doggone	
1111222	3	Jerry		jerry	ela	
2222333	3	Elaine		elaine	geo	
3333444	3	George		george	kra	
4444555	3	Kramer		kramer	jer	
0	0					
123	2	bob		bob	mouse	2009 Oct 30 16:01
12345	2	F12345	L12345	FL123	fl543	2020 Jan 23 19:24

rid = 1
vuid = 7

Figure 13.6 Click:, ValidUser, →Display in the Display Contents form

13.7.2 Display Structure

In the Display Menu click Structure. The form for Display Structure is the same as the form for Display Contents in Figure 13.5. In the form for Display Structure click Student and then click Display. The structure of table, Student, is displayed which is shown in Figure 13.7.



Request method is POST
rid = 1, vuid = 7
parse successful

Structure of table, Student

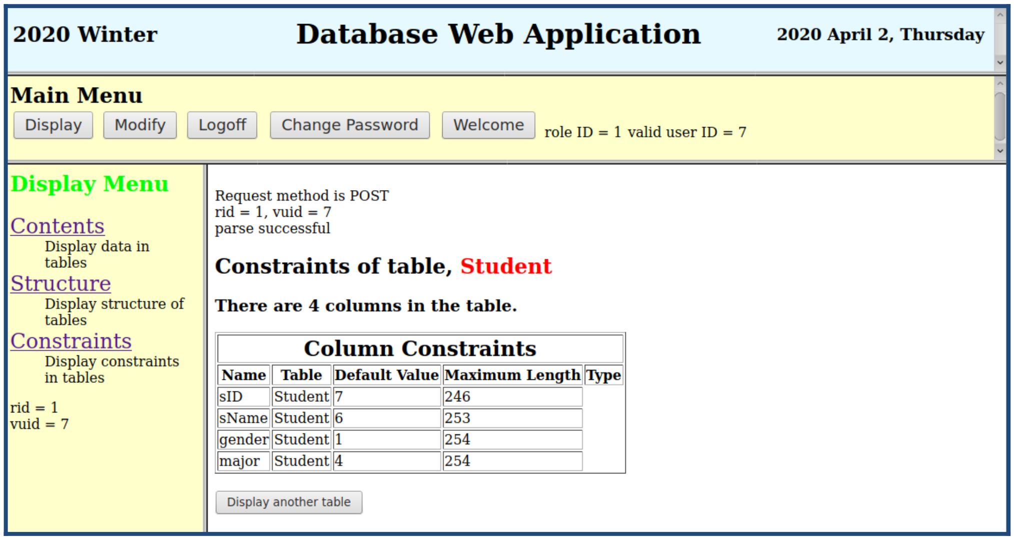
There are 4 columns in the table.

Column Properties			
Name	**Table**	**Default Value**	**Maximum Length**
sID	Student	7	
sName	Student	6	
gender	Student	1	
major	Student	4	

Figure 13.7 Click: Student→Display in the Display Structure form

13.7.3 Display Constraints

In the Display Menu click Constraints. The form for Display Constraints is the same as the form for Display Contents in Figure 13.5. In the form for Display Constraints click Student and then click Display. The constraints of table, Student, is displayed which is shown in Figure 13.8.



Request method is POST
rid = 1, vuid = 7
parse successful

Constraints of table, Student

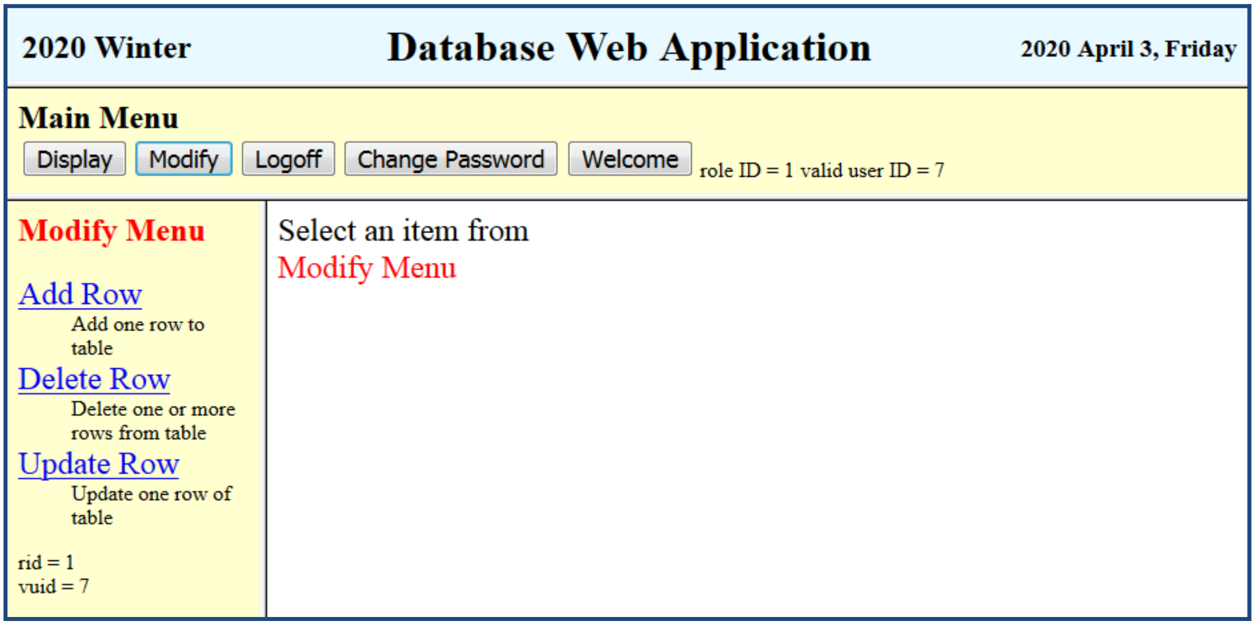
There are 4 columns in the table.

Column Constraints				
Name	**Table**	**Default Value**	**Maximum Length**	**Type**
sID	Student	7	246	
sName	Student	6	253	
gender	Student	1	254	
major	Student	4	254	

Figure 13.8 Click:Student→ Display in the Display Constraints form

13.7.4 Add Row

In the Main Menu click Modify. The Modify Menu is shown in Figure 13.9.



2020 Winter **Database Web Application** 2020 April 3, Friday

Main Menu

role ID = 1 valid user ID = 7

Modify Menu

Add Row
Add one row to table

Delete Row
Delete one or more rows from table

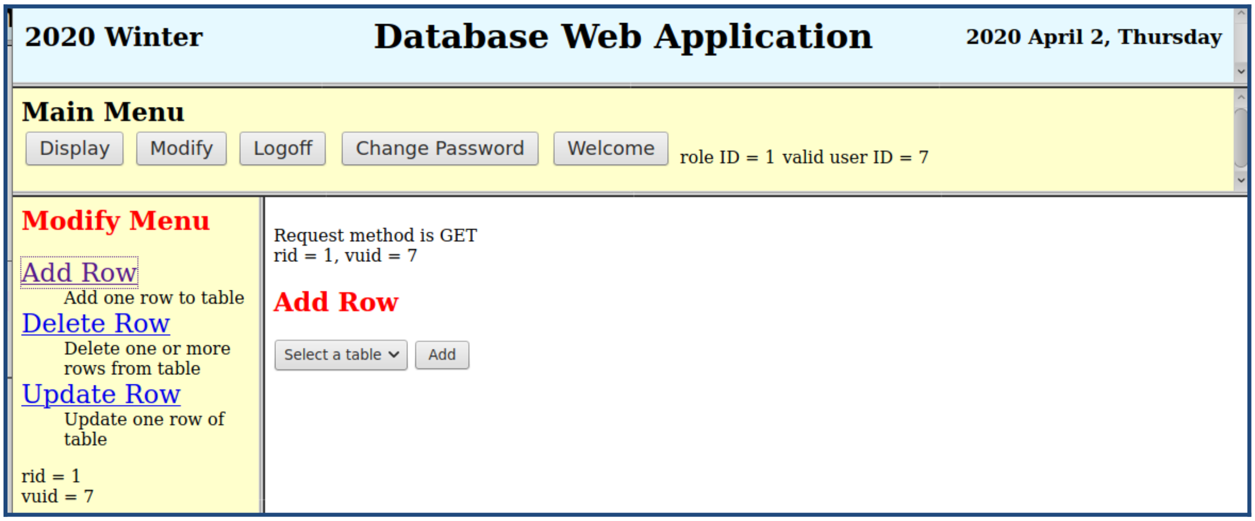
Update Row
Update one row of table

rid = 1
vuid = 7

Select an item from
Modify Menu

Figure 13.9 Screen after clicking, Modify, in Main Menu

In the Modify Menu click Add Row. The form for Add Row is shown in Figure 13.10. Note that the Modify Menu is unchanged.



2020 Winter **Database Web Application** 2020 April 2, Thursday

Main Menu

role ID = 1 valid user ID = 7

Modify Menu

Add Row
Add one row to table

Delete Row
Delete one or more rows from table

Update Row
Update one row of table

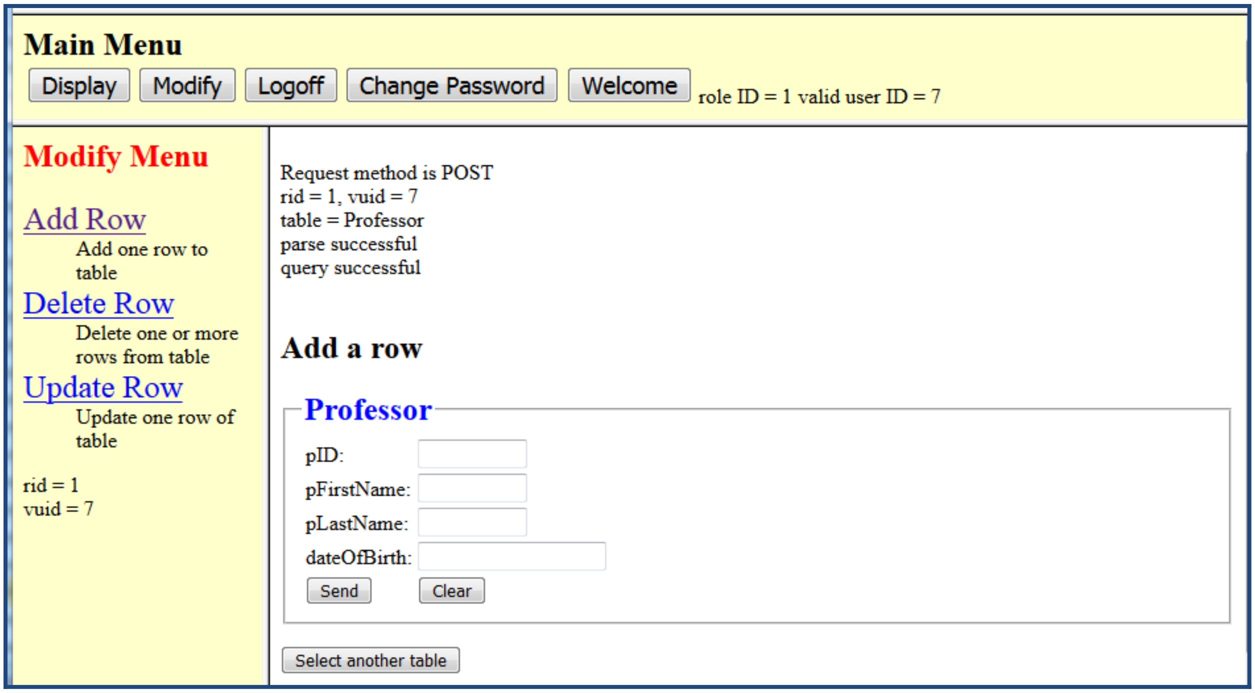
rid = 1
vuid = 7

Request method is GET
rid = 1, vuid = 7

Add Row

Figure 13.10 Screen after clicking, Add Row, in Modify Menu

In the form, Add Row, click the pull-down arrowhead to show all of the tables which can be selected. Click, Professor, and then click, Add. A form with all of the columns for table, Professor, is displayed which is shown in Figure 13.11.



role ID = 1 valid user ID = 7

rid = 1
vuid = 7

Request method is POST
rid = 1, vuid = 7
table = Professor
parse successful
query successful

Figure 13.11 Click: Professor→Add in the Add Row form

There are four new professors. The ID, first name and last name are the following. The date of birth is unknown.

112233 Randy Jackson
223344 Paula Abdul
334455 Simon Cowell
445566 Ryan Seacrest

First add Randy Jackson to the table, Professor. Type the data for Randy and then click, Send, to make the addition. Then click, Insert next row. Add the other three new professors. Then click, Select another table.

There are six new students. The ID, name and gender are the following.

123 Ross m, 234 Rachel f, 345 Joey m, 456 Monica f, 567 Chandler m, 678 Phoebe f

Add the six new students to the table, Student.

There is one new department. The code, name and location are the following, where Science Building is the location.

12 Chemistry Science Building

Simon Cowell is the chair of the Chemistry department.
Add the new department to the table, Department.
There are two new courses. The number, name and credits are the following

CHEM217 Analytical Chemistry 3
CHEM221 Physical Chemistry 3 (The name is incorrect intentionally; it should be Organic Chemistry)

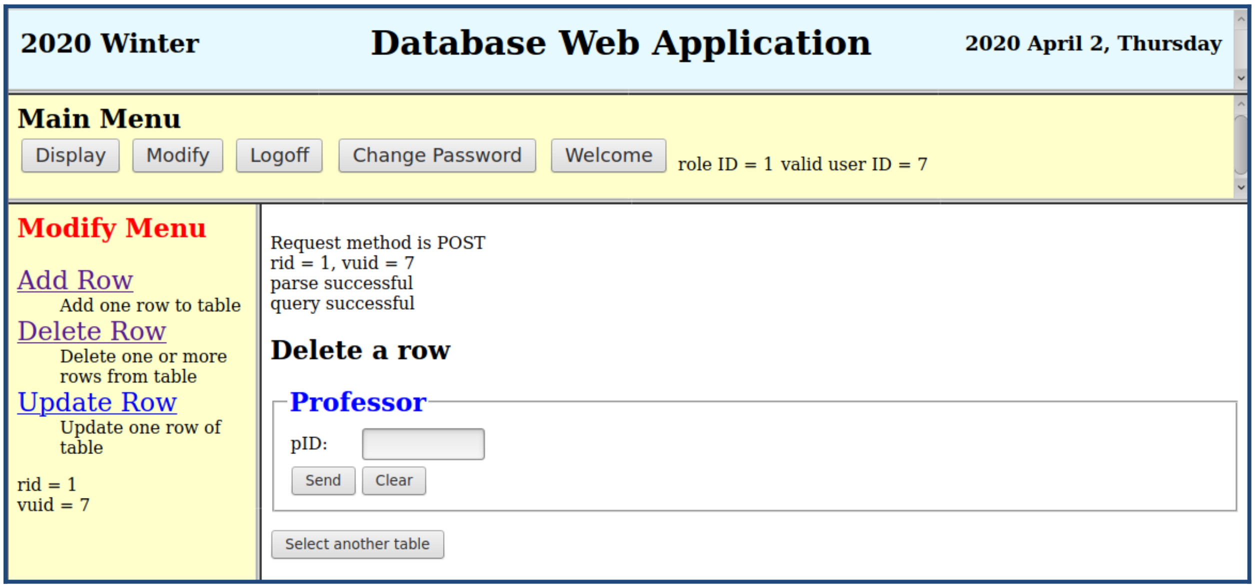
Both courses are in the Chemistry department.
Add the new courses to the table, Course.
CHEM217 is a prerequisite for CHEM221. Add the prerequisite to table, IsPrerequisite.

Two of the new students are enrolled in courses.
  Ross is in CHEM217 and received the grade, C
  Rachel is in CHEM221 and has not received a grade
Add the enrollment of Ross and Rachel to the table, EnrolledIn.

Verify the additions of data to the tables by displaying the contents of the tables: Professor, Student, Department, Course, EnrolledIn and IsPrerequisite.

Add all of the new professors and students to table, ValidUser.
The role ID of professors is 2 and is 3 for students. The username is the first name of the professor or student, in lowercase. The username must be different for every user. Choose a password for each new valid user. The password can be the same for more than one user.
Note that the date and time in the column, dateAdded, is automatically entered. This is done by using the PHP function, date().

Verify the additions of data to the table, ValidUser, by displaying the contents of the table.



role ID = 1 valid user ID = 7

rid = 1
vuid = 7

```
Request method is POST
rid = 1, vuid = 7
parse successful
query successful
```

Figure 13.12 Click: Professor→Delete in Delete Row form

13.7.5 Delete Row

In the Modify Menu click Delete Row. The form for Delete Row is the same as the form for Add Row in Figure 13.10. In the form for Delete Row click Professor and then click Delete.

A form for table, Professor, with the column, pID, is displayed which is shown in Figure 13.12.

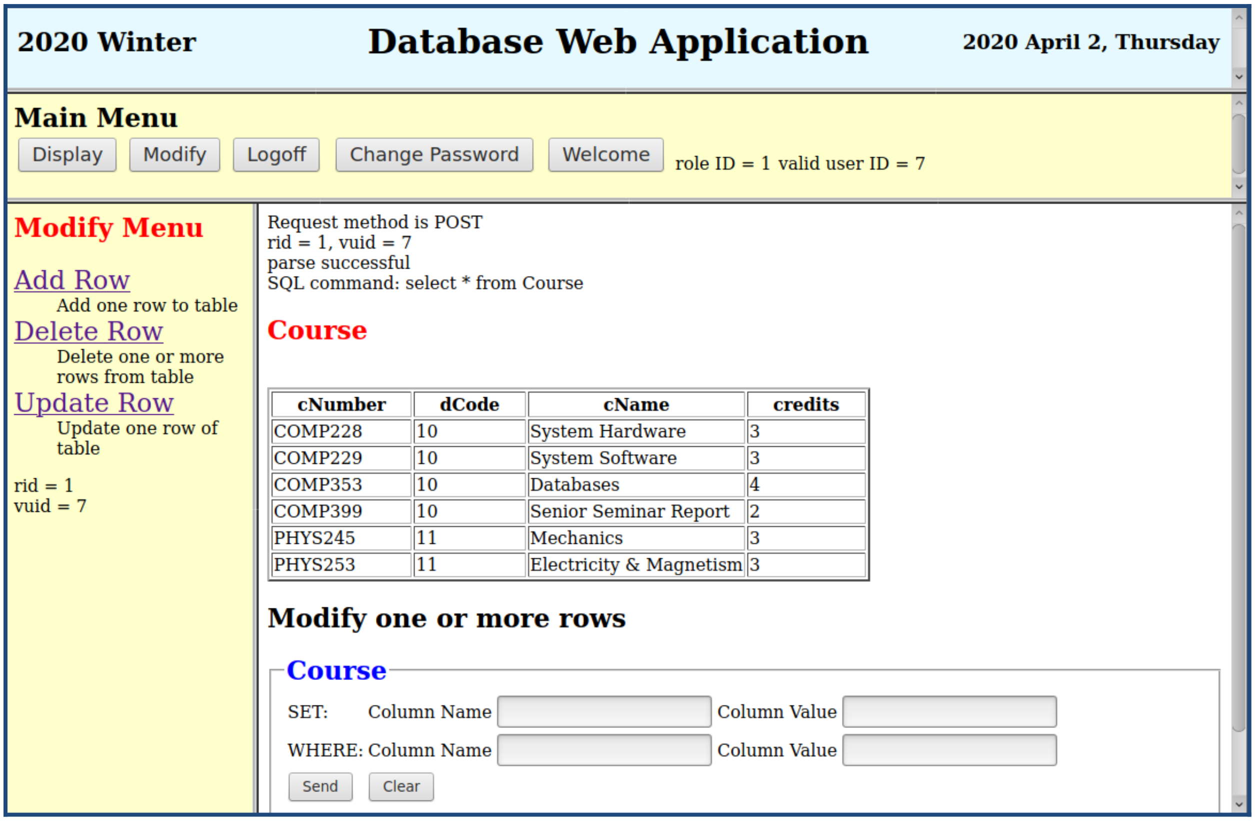
It has been discovered that Ryan Seacrest is not a professor. He is the host of a TV show.
So delete Ryan from the table, Professor. Type the pid for Ryan and then click, Send, to make the deletion.

To verify that Ryan was deleted, display the contents of table, Professor.

13.7.6 Update Row

In the Modify Menu click Update Row. The form for Update Row is the same as the form for Add Row in Figure 13.10. In the form for Update Row click Course and then click Update.

A form for table, Course, with four text elements is displayed which is shown in Figure 13.13. The contents of the table, Course, is displayed before the form. Also, after the form is completed and the user clicks, Send, the contents of the table, Course, is shown again with the modification that was made.



role ID = 1 valid user ID = 7

rid = 1
vuid = 7

```
Request method is POST
rid = 1, vuid = 7
parse successful
SQL command: select * from Course
```

Course

cNumber	dCode	cName	credits
COMP228	10	System Hardware	3
COMP229	10	System Software	3
COMP353	10	Databases	4
COMP399	10	Senior Seminar Report	2
PHYS245	11	Mechanics	3
PHYS253	11	Electricity & Magnetism	3

Modify one or more rows

Course

SET:	Column Name		Column Value	
WHERE:	Column Name		Column Value	

Figure 13.13 Click: Course→Update in the Update Row form

The SQL command to modify one or more rows in a table is

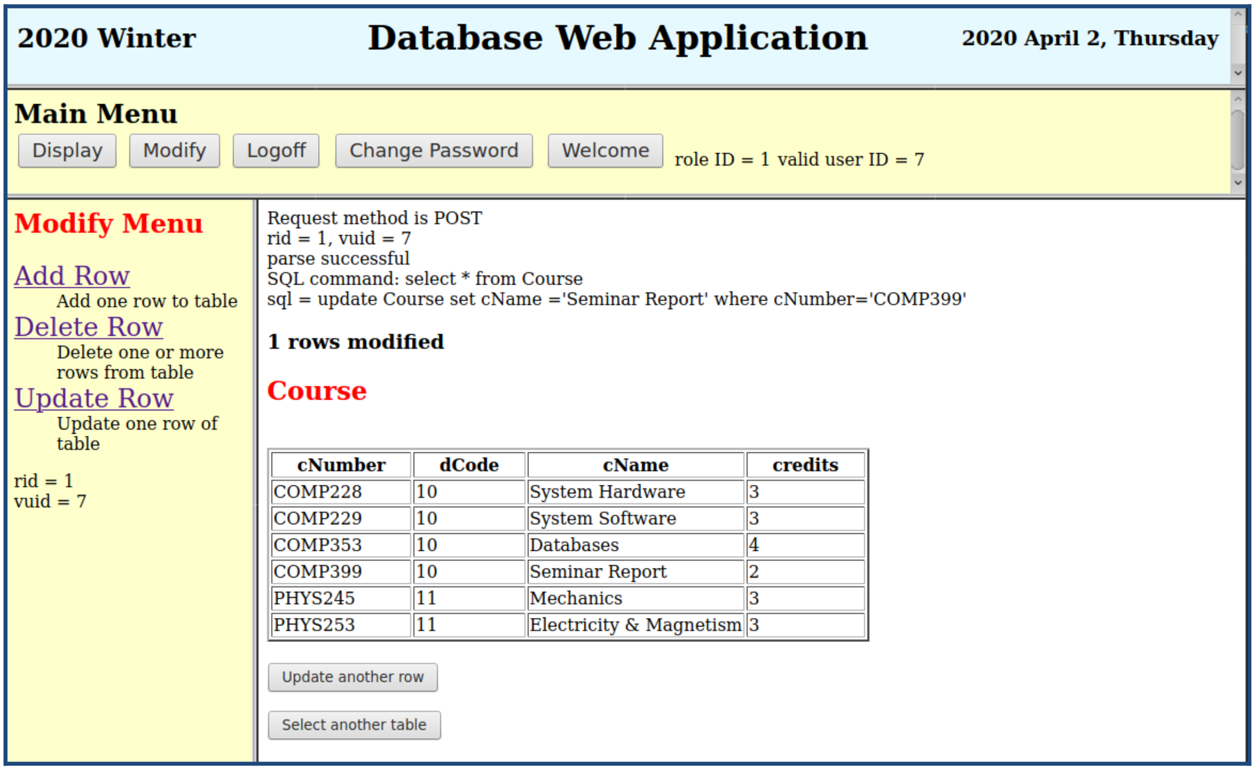
UPDATE *tablename*
SET *columnName* = *columnValue*
WHERE *columnName* = *columnValue*;

In the menu, *tablename* is chosen. In the form in Figure 13.13, the other four quantities are typed by the user.

The name of the course with number, COMP221, is Organic Chemistry and not Physical Chemistry.

Change the name by typing the appropriate quantities in the form. Click, Send, to make the modification.

The result is shown in Figure 13.14.



role ID = 1 valid user ID = 7

Add one row to table

Delete one or more rows from table

Update one row of table

rid = 1
vuid = 7

```
Request method is POST
rid = 1, vuid = 7
parse successful
SQL command: select * from Course
sql = update Course set cName ='Seminar Report' where cNumber='COMP399'
```

1 rows modified

Course

cNumber	dCode	cName	credits
COMP228	10	System Hardware	3
COMP229	10	System Software	3
COMP353	10	Databases	4
COMP399	10	Seminar Report	2
PHYS245	11	Mechanics	3
PHYS253	11	Electricity & Magnetism	3

Figure 13.14 Screen after updating Course

14 World Wide Web, A Hacker's Heaven

The World Wide Web (web for short) and the internet are covered in Chapter 1. The World Wide Web was introduced to the world, formally in May 1994, at a conference called "World Wide Web I". The web and Mosaic, a graphical browser, which was announced soon after, has transformed the internet to such an extent that for most people, the internet is the web. The web has given rise to a number of rich and powerful corporations which did not exist before its advent. The web browsers, after many generations, have become the graphical interface replacing interfaces such as motif and X-windows. The cell phone with its tiny screen has become the de-facto interface to all kinds of applications and has introduced new methods of communication and connections including to the internet via the web. The control of all this by a small number of monopolistic corporations, who have amassed large quantities of data on people, has created a situation which has become a web of betrayal of the promise of sharing and providing information, "freely".

14.1 The Original Ideas

While 1945 was a significant year in many ways, there were two events that have been little noticed but that have come to shape the world in ways almost as significant as the more celebrated ones. The main actor in both events is Vannevar Bush who is rather little known despite both his stature as an eminent scientist and the accuracy of his prophecy for the future. Over a number of years, he was director of the Office of Scientific Research and Development where he oversaw the work of thousands of scientists tasked with applying science to warfare. Trained as an engineer and having worked on the Manhattan project which developed the atomic bomb, he was a member of the committee which ultimately recommended bombing two Japanese cities, the only uses of hostile atomic power, to date. In 1945 Bush published the essay in The Atlantic magazine, As We may Think, in which he articulated a kind of network that was to become the precursor to the world wide web. The successful implementation of the ideas expressed therein would have to attend the development of more powerful and affordable computers and the networking of these computers. The initial impetus to the interconnection of computing devices was the need among scientists and academics to share resources and information. This in turn led to the development of the internet. The many possible designs were narrowed down to the packet protocols, the basic building block of the internet for the transmission of data in the form of packets. This idea is credited to Donald Davies along with Paul Baran. The internet, thus, was the culmination of this concept in interconnecting computers and the emergence of the TCP/IP communication protocol, the concept of network addresses and so on The idea expressed in Bush's essay was realized by the introduction of hypertext in the late 1980s. Hence, hypertext was around before the development of the web at Conseil Européen pour la Recherche Nucléaire (CERN). It is the mechanism by which one can easily peruse and navigate through electronic documents by providing links (pointers) to jump from one place to another rapidly with much greater accuracy than one could in a physical book. The original hypertext markup language (HTML), a scheme used to markup a document with links was very rudimentary and its weakness, in-spite of its many generations of standards is still evident. The introduction of the hypertext transport protocol (HTTP) required the transport level mechanism provided by TCP/IP, the common protocol to reliably transfer data packets from a source to a destination.

The idea expressed in Bush's essay was realized by the World Wide Web in the late 1980s. The web was formally introduced to a receptive audience at the first world wide web conference (WWW I.) held in May 1994 in Geneva and this meeting has sometimes been called the Woodstock of the web. It was a watershed moment and right after the announcement of the WWW1 conference, the National Center for Super-computing Applications (NCSA) announced the “Mosaic and the Web conference” in Chicago which, after some negotiations, was renamed the WWW II.

The emergence of the web and graphical browsers resulted in the skyrocketing of the share of internet bandwidth by web traffic. The web, in its turn has been dominated by social networks with a lot of traffic via mobile devices and all kinds of twitters twitting at all time of the day and night. The higher and higher speeds provided by the internet (and the mobile devices) resulted in the introduction of audio and video streaming services of various kind. This later addition to the internet use now dominates the traffic on the internet, the cost of which is borne by the end user who pays not only to the internet service provider but also to the streaming service provider. What has resulted is the emergence of the new breed of technology-driven robber barons and hackers. They all mine and expoit user data and expose them to privacy and security vulnerabilities.

14.2 Cookies

The HTTP protocol was designed to provide information corresponding to a request. It was designed to not remember nor keep track of any request, much less a history of requests. This was not only done to facilitate sharing information without strings attached but also to avoid storing this information from the users of the HTTP server who may or may not return. Also, permanent storage was limited a resource. HTTP protocol is idempotent since the same URL (request) would be served the same web content.

To overcome this inherent feature of the protocol and to off-load the storage costs to the clients, cookies were proposed to maintain the record of the users browsing activity on the client site. Cookies contain the server specified arbitrary state relevant information and stored on the client system’s permanent storage. When a HTTP request from a client is received at a server in the form of a simple URL served by the server, the server responds with the content corresponding to the request and requires the browser to save a cookie which contains the server specific data including a random identifier. This cookie is sent along with any subsequent request to the server. In this way, the stateless nature of the HTTP protocol is no longer stateless. The server can then log all the activity of the user who is identified by the IP address and the browser and hardware/ software details of the system making the request.

The server gets a free ride ignoring the privacy issues of the users. This has allowed the exploitation of the user’s information by interested commercial organizations. The browsers have a limit of the number cookies for each server (domain) visited by the browser.

The cookie contains information to overcome the memory-less feature of the HTTP protocol such as identifiers, to record the user's browsing activity and additional arbitrary pieces of information which the server may need to verify the user's future requests and to record the recent log of the user's requests which could be used to tailor the contents of the subsequent pages sent from the server. At the same time, this history can also be used to track the user's visits. While cookies are stored locally the browser is required to allow access of the cookie to the server that had set it.

Cookies can be classified into two groups: first-party cookies and third-party cookies; there are no second-party cookies. According to the IETF/RFCs, a cookie, is a small amount of data that the server sends to the client to be stored in the client's permanent storage. A first-party cookie is from the website that the user is currently visiting. A third-party cookie is from a website other than the one that the user is currently visiting. The information in a cookie is in the form of an attribute-value pair (to identify the user via a user name and password) and is accessible to the server site which created the cookie. In this way the server can determine the history of previous transactions with the client. If first-party cookies were disabled, a server could not keep track of the client's past activity which was the original motivation of the web and the HTTP protocol. Cookies were cooked up to support applications such as shopping. Also, some sites would refuse to serve web pages if cookies for the site were not allowed.

A server creates and requires the user's browser to store cookies in the client's permanent storage by sending a "set-cookies" header, with predefined and formatted attribute-value pairs (expire-date, domain, path and secure). Cookies could also be set and got by using JavaScript embedded in the web content sent by a server.

14.3 Cookies and Privacy

Cookies are used by most web applications created by web service providers to implement the missing state maintenance feature of the HTTP protocol. There are few if any web servers, at the time of writing, which provides service without trying to collect information about the users of their service. A case in point is the on-line banking application put in place by banks. When their customers use the on-line banking application, it allows these banks to reduce the personal service at their branches and thereby reduce expenses. However, not satisfied with this saving, much like the tech-giants of this century, they are also tracking their customers and use the data collected to advertise products and services to them. If one looks at the on-line banking web page of any bank, one sees the number of third-party services being used. Some of these would be tracking the users and others may be advertisers that these banks include in their contents. The banks use some of these to track the user interaction during their visit to the site as well as their partner sites. In spite of the rampant use of cookies, as mentioned above, only a small number of web users know about the intent of cookies and a very small number of these explore ways to protect their privacy. The following section classifies cookies based on their content and usage.

14.3.1 Classification of Cookies

At the present, the following classification is generally accepted.

14.3.1.1 HTTP Only cookie

A cookie which has a HTTP Only flag set can't be accessed by client side scripts (for example, Javascript) and hence avoids being used in cross-site scripting.

14.3.1.2 Persistent cookie

A persistent cookie is stored in permanent storage and has an expiration date or “life-time” set by a server and accessible to it when the user visits the server’s web site or accesses a resource associated with the server. Hence it is also called a tracking cookie and used to record a user’s web behaviour.

14.3.1.3 Secure cookie

A secure cookie can be transmitted only over a secure connection. It has the secure flag set.

14.3.1.4 Session cookie

A session cookie is stored in non-permanent storage and does not have an expiration date. It is deleted when the browser process terminates. It is also called an in-memory cookie, transient cookie or non-persistent cookie.

14.3.1.5 Super cookie

A non-super cookie is associated with a specific domain whereas a super-cookie is associated with a top level domain such as, .com, .org, .ca. It is a potential security concern and is therefore often blocked by web browsers.

14.3.1.6 Third-party cookie

A cookie set by the server site being visited is called a first-party cookie. If the visited site includes contents from some other site, not explicitly visited, such a non- explicitly visited site could set a cookie called a third-party cookie. This is used for tracking and used to push products and services to the unsuspecting user. The user is the second-party in this scenario, but there are no second-party cookies. It is worth noting that in everyday transactions the first party (party of the first part) is the entity initiating a transaction with another entity, which is the second party (party of the second part). The third party is independent of these two parties. In the web scenario, the so called third party is often brought in by the first party, for example the hidden pixels used as trackers piggy-back in the content from the visited site. Hence, in the web scenario the third party is not really independent. A user can use the preference option of the browser to disallow third party cookies.

Cookies are used not only to overcome the stateless nature of the HTTP protocol and thereby keep a user logged in and, for example, allow a record of shopping cart content. Cookies also track a user’s web browsing habits and hence compromise their privacy. This can also be done to some extent by using the IP address of the computer requesting the page or the referrer field of the HTTP request header, but cookies allow for greater precision. Consider the first access of some user to a web site. Such a request would not have a cookie associated with it and the server concludes that this is the first visit by the user. So to keep track of the user and provide a continuity of the visit, the server creates a unique identifier (usually a string of random letters and numbers) and sends it as a cookie to the browser together with the requested page. The browser saves the cookie in the permanent storage of the user. The browser would also include the cookie, automatically, for each subsequent request to the server. The server not only sends the requested page as usual but using the data in the cookie to classify the user, stores the addresses (URLs) of the requested pages and the dates/times of the requests in a log file. By analyzing this log file, it is then possible to find out which pages the user has visited, in what sequence, and for how long. Cookies allow the tracking of users by the first party (visited site, for example, an on-line social network) or a third party, The parties collect information about these users to be used to target them with tailored publicity for commercial or political purposes by others.

14.4 The HTTP Loophole

One of the features in the HTTP protocol is the inclusion of contents from third parties. While this is alluring, it has caused a large privacy and security issue for the users of the web; this may be considered a loophole. Using this feature, a server can incorporate in the web page served to the user, contents which could include tags to fetch images, scripts and obtain and store cookies not only from the visited site but also from third parties. This cocktail of contents, once considered as a great feature of the protocol, has morphed into one of the preferred methods to track users: it is called cross-site tracking. It has also led to organizers bent on cutting costs of their on-line web servers, and out-sourcing things such as measuring user satisfaction. This outsourcing also has the potential of making money by allowing other third parties to serve publicity and services to their users.

The HTTP protocol used in the current web has given birth to, among others, tracking business organisations. A tracker is company which collects information about users as they browse the web. The trackers scripts and tracking pixels are inserted into the contents provided by the website's servers to their users' requests. This is in exchange for some service or revenue provided by the tracking company. Trackers usually use some kind of user identifier in order to link information about the users as they visit different sites, and thus build a profile, based on the their browsing history.

In reality, no web site server requires third-party tracking. They do so to outsource, use economy of scale and save cost. At the same time the data generated from their users by the trackers would result in some revenue. The websites use these services so the feedback from the trackers would allow them to better classify their customers and hence target them so as to maximize the benefit to the website. Furthermore, in the majority of cases, trackers are invisible on the pages they appear. Often their presence is in the form of invisible pixels. There is little or no transparency about who is logging the user activity on the web. A common practice is to use tracking pixels in the contents served to the user. These third-party sites are used to track user's browser journey not only on line (on the visited site as well as off the visited site) but also off line. They collect not only anonymous data but also IP addresses, platform details (both hardware and software) browser signature, demographic data, data about interaction and page views, pseudo-anonymous data including IP Address, location, and click-stream data. Many commercial sites also use third parties to enable and deliver the publicity at the right time to the unsuspecting users and thereby collect revenues. The third-parties are free to sell/share the data to any other parties in perpetuity.

There is the possibility of including contents from third parties. The HTTP protocol allows scripts from third parties to run on the user's browser. The user could install browser add-ons which could control whether to run scripts such as JavaScript. However, the user needs to be aware of these add-ons and how to control them. Since most sites use scripts the contents from such sites may not function if the add-ons do not allow scripts or can distinguish first-party scripts from third-party ones. Electronic Frontier Foundation (EFF) has a research project called Panopticlick which is an effort to illustrate the problem with tracking techniques that users could use to do a test of their browser. Each browser has a unique fingerprint which could be used to identify the user's browser, one of the coordinates used in tracking.

One of the hacking practices common is to tempt users to open emailed malicious scripts to hack their data including cookies stored in their computer and thus allow the hacker to impersonate the user.

14.5 SQL Injection

With the acceptance of the current HTTP protocol the usual method of interaction by the end user to most applications is via a web based interface, usually a web form. When a web based form is used to provide an interface to a database, it is important to verify the legality of the user's input data. A popular attack method used to compromise the database is to insert a series of SQL statements which would reveal the structure and hence the contents of the database.

We have shown this in a detailed case study involving a web based application called CrsMgr. This is an application that started out as a PL/1 batch application which evolved into a X-Window based system. With the advent of the web and the introduction of PHP, and open source database MSQL, the application was ported for the web. It has gone through a number of versions including ones using PHP3, PHP4, PHP5. In the study reported in Zhu, it is shown that a malicious user or an intruder, after some experimentation, could discover a vulnerability. Thence, employing a sequence of statements such as show tables, describe table, select * from table, can gain access to a backend database. Without adequate verification of the user input, the malicious code would become part of the SQL query constructed by the PHP (or other) script, and would be executed when the query is sent for execution and the results would be shown on the result web page. Once the database structure and user IDs and passwords are compromised, the malicious user can then query the database, collect sensitive information, change and/or selectively destroy the contents of the database.

SQL injection is a web-based hacking technique which exploits the absence of input or parameter filtering in web form based applications. The date of discovery of the first attempts of SQL injection is not known, However, mention of it started to appear not long after the introduction of the web in 1994. Along with the dot.com craze, at the dawn of the 21st century, the cases of SQL injection became more and more common. Within a few years, over 90 percent of data breaches were due to SQL injection. SQL has become the de-facto language to query the backend database used in web application to render it data-driven and dynamic. Using one of the SQL injection methods, malicious users can insert SQL scripts into places where textual input is expected in a designated user input area. If no precaution is taken, such inserted SQL scripts could be executed by the server and the result would be displayed to this intruder. Hence a hacking attack with a series of appropriate SQL scripts could be made by a hacker and the result of these would enable the intruder to gain knowledge, otherwise not accessible to this user. A SQL injection attack could consist of a series of SQL scripts inserted in a data input area and executed by the system.

In general, SQL injection can be categorized as follows.

First Order Attack: when the malicious code is injected and executed immediately.
Second Order Attack: when the malicious code is saved in persistent storage, considered as credible data, and is executed by another query.
Lateral Injection: when the malicious code changes the value of one or more environment variable or implicit functions.

As the problem of SQL injection has grown, the methods used against SQL injection have been introduced and can be classified into three general methods to protect against SQL injection:
(1) Sanitize the input data by keyword filtering, escaping special character and type checking.
(2) Parameterize the input data. The query statement is prepared and care is taken before binding the user supplied data to the parameters of the statement.
(3) Use the least privilege for SQL account used for each query

A simple example of SQL injection by an intruder would be, for example, through a website's login page containing a PHP script segment such as the following.

```
$user=$_POST['userName'];
$pw=$_POST['password'];
$sql = “SELECT * from User where userid =”$user” and password=”$pw””;
```

If the intruder enters data two random values “xyx” and “123” “or 1<>2” for the user name and password where the string such as “ or 1<>2” is appended after the password the query presented to the database becomes the following.

```
“SELECT * from User where userid =”xyz“and password=”123 or 1<>2”;
```

Since 1<>2 is true, and if there are no features in the PHP script to monitor and sanitize the user input to the query it would be executed and the intruder would be logged in. Once the intruder discovers this, he would tailor other inputs with other SQL codes to discover the structure of the database and thence try more queries to get sensitive information from the database. The sequence of queries could be appended to the data inputs discovered to be not monitored for correct type or sanitized. This way the intruder would be able to access almost any data from the database including the administrative user name and password. The intruder could also compromise the database.

A detailed case study of a system, built before the emergence and general knowledge of SQL injection attack techniques is reported in Zhu. It also shows the technique used to harden the system. One of the features built in for any form-based input to any database is to validate the input and ensure that it does not contain any compromising access to the database. If user's input matches the keywords such as use, drop, create, and, or, where, limit, group by, select, insert, update, delete, union, into, load_file, putfile and exec, and these keywords are followed by a non-English character such as white-space, the user input is considered as a suspicious input. Furthermore, if user's input contains special characters such as ' , /*, *, ../ and ./ then the input is considered as suspicious and special characters are replaced by the encoded characters, for example, "< "replaced by its HTML entity encoding, "&lt;".

14.6 Cross Site Scripting

Cross Site Scripting (XSS) is an attacking technique wherein scripts, for example JavaScript or JSP, is injected into a user input text area. Such scripts, if no precaution to monitor the user input is in place, would be accepted by the web server and executed, typically, by the host web browser and thus bypass access controls, and access and/or transmit sensitive data. One type of data that could be transmitted is the user's persistent cookies which could then be used to impersonate the user from another site. This would be possible if the server that set the cookie provides "Remember me /keep me signed in" option and the user had, for convenience opted for it.

Cross Site Scripting can be divided into the following types.

1. Store XSS. The malicious code is stored by the attacker on the server side, typically in persistent storage (database). When the server handles the victim's request, the malicious code is used to construct a web page for the victim.
2. Reflect XSS. This type of Cross Site Scripting is executed immediately. The malicious code is sent to the server and carried back to the browser.
3. Document Object Model (DOM) based XSS. The malicious code does not come from the server. Instead, it utilizes the DOM interface to hijack the request of the victim.

A recent development to avoid hackers and cross site scripts transmitted via emails is to scan email by the user mail agent for hackers or scripts before it is handed to the outgoing mail system. However, it is always possible to bypass such a mail agent.

To defend against Cross Site Scripting, the following rules should be observed.

- Since any of the contents of an HTML page can be inserted from a third-party site or could be a script, is it recommended that all untrusted third-party or user-supplied content must be escaped before inclusion in the page. Even data from clients persistence storage, may contain malicious script.
- The attribute data in HTML tags, such as font size and width, should be sanitized.
- Any data going into JavaScript or CSS code should be sanitized.
- The browser should use Http Only property for cookies to prevent them from being stolen and disallow third-party cookies.

14.6.1 Cascading Style Sheet (CSS) Injection

The CSS injection problem arises if the web application allows style scripts supplied by the user. Such styles could override existing styles. Since the styles sheet is rendered on the browser, it may take up processing resources and could even freeze the display. User supplied CSS could be used for XSS when scripts could be used to make requests which, if allowed to be executed, could compromise the backend database.

Allowing users the freedom to personalize any web application by using their own CSS, loaded from anywhere, is an invitation for injection to be definitely included in any real application!

14.7 Generating Safe Web Applications

Software, regardless of where it comes from and how much it costs, is often infected with bugs. Many years of software engineering has not completely improved the state of software. So creating any software application is a work in progress. For web database applications, the general principles are the following.

- Do not trust the users input
- Scan and sanitize user input
- Give minimum privilege for each application
- Honour the privacy of the users, customers
- Ensure the security of the information under trust

For example, to allow a user to download source code, do not give access to the main database. An alternative would be to create an ad-hoc database or non-editable database views.

Since software keeps changing, the applications have to keep up-to-date with newer versions of software. As new bugs and security breach methods are discovered, the existing applications should be updated and attack points eliminated or hardened.

Appendix A - References

A1 Introduction

A1.1 Books

Bipin C. Desai
An Introduction to Database Systems, West Publishing, 1990
Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom
Database Systems: The Complete Book, Pearson, 2009

A1.2 On-line

Time-sharing
https://en.wikipedia.org/wiki/Time-sharing
Cloud computing
https://en.wikipedia.org/wiki/Cloud_computing

A2 MySQL Shell

The URL for the MySQL 8.0 Reference Manual in English
https://dev.mysql.com/doc/refman/8.0/en/

The reference manual for version 8.is not applicable for older versions of MySQL. There are other reference manuals for older versions of MySQL.

The URLs for the documentation and knowledgebase in English for MariaDB
https://mariadb.org/documentation/
https://mariadb.com/kb/en/

A3 SQL Basics

A3.1 W3Schools

The W3Schools site map lists all tutorials available
http://www.w3schools.com/sitemap.asp
Under Server Scripting, click SQL Tutorial
http://www.w3schools.com/sql/default.asp

A3.2 SQL.org

An SQL reference which includes tutorials and reference material for many different database programs, including MySQL and Oracle
http://www.sql.org/
Click, Beginner Tutorials, under Online Resources.
On the next web page, there is a list of links to six different tutorials.
Click, A Gentle Introduction to SQL
http://www.sqlzoo.net/
Note that database program can be chosen, for example MySQL or Oracle, since for some commands there is a syntax difference among the database programs.

A3.3 MariaDB/MySQL resources

The MySQL Reference Manual includes chapters which document SQL.
The internet address for the manual in English
https://dev.mysql.com/doc/
The URL for MariaDB tutorials
https://mariadb.com/kb/en/training-tutorials/
Every user of MariaDB/MySQL should become familiar with the contents of these references.

A4 SQL Constraints

A4.1 Application Developer's Guide

Database Resources
https://mariadb.com/resources/

A4.2 SQL Reference

Complete list of SQL statements for data definition, data manipulation
https://mariadb.com/kb/en/sql-statements/

A5 SQL Relationships

MySQL documents
https://dev.mysql.com/doc/
MariaDB
https://mariadb.org/documentation/

A6 SQL/PSM

A6.1 Books

A comprehensive book about PL/SQL
Steven Feuerstein with Bill Pribyl,
Oracle PL/SPL Programming, 2014, O'Reilly

A6.2 On-line

MySQL Reference manual
https://dev.mysql.com/doc/refman/8.0/en/
MariaDB Server documentation
https://mariadb.com/kb/en/documentation/
MariaDB Knowledge Base
https://mariadb.com/kb/en/
Oracle documentation
https://docs.oracle.com/en/database/oracle/oracle-database/index.html

A7 HTML

A7.1 Books

There are many books which introduce HTML. One which is suitable for beginners without any knowledge of HTML
Elizabeth Castro,
A HTML 4 for the World Wide Web@, Fourth Edition, 2002, Peachpit

An appendix in this book has a list of the HTML tags and attributes with the page number in the book where the tags and attributes are demonstrated. As with any book in Computer Science, look at the book thoroughly in a bookstore before deciding to buy the book.

A book which is suitable for professional programmers as well as beginners
Chuck Musciano and Bill Kennedy,
HTML & XHTML The Definitive Guide, 5th Edition, 2002, O'Reilly.

A7.2 On-line

A short account of the history of the web is the following, which includes many links to other pages with details of the development of the web
http://www.hitmill.com/internet/web_history.html
One of the most popular tutorials for HTML on the web
http://www.w3schools.com/html/html_primary.asp
The W3C specification for HTML 4.01
http://www.w3.org/TR/html4/
HTML Living Standard
https://html.spec.whatwg.org/
History of hypertext
https://en.wikipedia.org/wiki/History_of_hypertext

A8 CSS

A8.1 Books

A book which is suitable for professional programmers as well as beginners
Chuck Musciano and Bill Kennedy,
HTML & XHTML The Definitive Guide, 5th Edition, 2002, O'Reilly.
One chapter each is devoted to CSS, XML and XHTML.

A8.2 On-line

A8.2.1 CSS

A popular tutorial for CSS is by W3Schools
http://www.w3schools.com/css/default.asp
An introduction to CSS, called CSS Primer
http://www.w3c.rl.ac.uk/primers/css/cssprimer.htm
The W3C (World Wide Web Consortium) CSS 2.1 (Cascading Style Sheets level 2 revision 1) specification
http://www.w3.org/TR/CSS21/

A8.2.2 XHTML

The W3Schools tutorial for XHTML
http://www.w3schools.com/xhtml/default.asp
The first version of XHMTL is 1.0 which was released in 2000. The W3C specification of XHTML 1.0
http://www.w3.org/TR/xhtml1/
The latest version of XHTML is 2.0 and was released in 2003

The W3C specification of XHTML 2.0
http://www.w3.org/TR/2003/WD-xhtml2-20030506/
The W3C specification of HTML5
http://en.wikiesia.org/wiki/HTML5

A9 PHP

A9.1 Books

Rasmus Lerdorf, the creator of PHP, has coauthored a book about PHP which every serious PHP programmer should at least look at.

Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre,
Programming PHP Paperback 2006 , O'Reilly

A9.2 On-line

The PHP manual is on the internet, in English
http://www.php.net/manual/en/
as well as in other languages.
A good online tutorial for beginners for PHP is offered by W3Schools
http://www.w3schools.com/php/default.asp

A10 JavaScript

A10.1 Books

An authoritative book about JavaScript which contains complete references for both core and client-side JavaScript:

David Flanagan,
JavaScript The Definitive Guide, Fifth edition, 2006, O'Reilly.

A10.2 On-line

The ECMA (European Computer Manufacturers Association) has published a standard for the EcmaScript language. The latest is 9th Edition / June 2018)
https://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
Many simple demonstrations of JavaScript are given on the web
https://en.wikipedia.org/wiki/Regular_expression
https://en.wikipedia.org/wiki/JavaScript
http://www.w3schools.com/js/js_intro.asp

A11 State

A11.1 Cookie

https://phpbuilder.com/category/articles
https://tools.ietf.org/html/rfc6265
https://www.w3schools.com/js/js_cookies.asp

A11.2 Session

https://www.w3schools.com/php/php_sessions.asp

A12 MYSQL API

A12.1 PHP MySQL functions

The PHP Manual contains a chapter, Function Reference, which includes the section, MySQL. The internet address for the MySQL Functions in English

http://www.php.net/manual/en/book.mysql.php

A12.2 MySQL Reference Manual

The MySQL Reference Manual in English

https://dev.mysql.com/doc/refman/8.0/en/
https://mariadb.org/documentation/

A13 Applications

A13.1 PHP manual

The table of contents of the PHP manual in English

https://www.php.net/manual/en/book.mysqli.php

A13.2 MySQL functions

The MySQLi functions

http://www.php.net/manual/en/book.mysql.php

A13.3 Date and time functions

The PHP manual contains a chapter, Function Reference, which includes the section,
Date and Time Functions

https://www.php.net/manual/en/refs.calendar.php

A14 World Wide Web, A Hacker's Heaven

A14.1 Books

Billy Hoffman, Bryan Sullivan,
Ajax Security, Addison-Wesley, 2008

A14.2 On-line

A Brief History of the Internet
https://www.internetsociety.org/resources/doc/2017/brief-history-internet/

Paul Baran
https://en.wikipedia.org/wiki/Paul_Baran

Tim Berners-Lee
Information Management: A Proposal, March 1989, May 1990,
CERN, available from
https://www.w3.org/History/1989/proposal.html

Maude Boneenfant, Bipin C. Desai, Drew Desai, Benjamin C. M. Fung, M. Tamer Ozsu, Jeffrey D. Ullman
Panel: The State of Data: Invited Paper from panelists
IDEAS '16: Proceedings of the 20th International Database Engineering & Applications Symposium, July 2016, Pages 2–11
https://doi.org/10.1145/2938503.2939572

Vannevar Bush

As we may think, The Atlantic, July 1945, https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/

Carole Cadwalladr, Emma Graham-Harrison
Revealed: 50 million Facebook profiles ,harvested for Cambridge Analytica in major data breach, https://www.theguardian.com/news/series/cambridge-analytica-files

Cross Site Scripting
https://en.wikipedia.org/wiki/Cross-site_scripting

Donald Davies
https://en.wikipedia.org/wiki/Donald_Davies

David M. Kristol
HTTP Cookies: Standards, privacy, and politics, ACM Transactions on Internet Technology (TOIT) 1.2 (2001): 151-198, https://dl.acm.org/doi/10.1145/502152.502153

HTTP cookie
https://en.wikipedia.org/wiki/HTTP_cookie

Internet protocol suite
https://en.wikipedia.org/wiki/Internet_protocol_suite

David Streitfeld
As Amazon Rises, So Does the Opposition, NY Times, April 2020 https://www.nytimes.com/2020/04/18/technology/athena-mitchell-amazon.html,

Jianhui Zhu, Bipin C. Desai
User Agent and Privacy Compromise , IDEAS15, 2015 Yokohama [Japan], http://dx.doi.org/10.1145/2790798.2790803

Jianhui Zhu
Secure CrsMgr: a course manager system, Master's thesis, Concordia University, 2016, https://spectrum.library.concordia.ca/981879/

PHP Group, Prepared Statements
http://php.net/manual/en/mysqli.quickstart.prepared-statements.php

RFC2109, HTTP State Management Mechanism, Feb. 1997
http://tools.ietf.org/html/rfc2109

RFC2965, HTTP State Management Mechanism, Oct. 2000
http://tools.ietf.org/html/rfc2965

SQL Injection
https://en.wikipedia.org/wiki/SQL_injection

Third-Party Cookies
http://www.pcmag.com/encyclopedia/term/52849/third-party cookie

OWASP Top Ten
https://owasp.org/www-project-top-ten/

Testing for CSS Injection
https://wiki.owasp.org/index.php/Testing_for_CSS_Injection_(OTG-CLIENT-005)

Appendix B – Supplemental Material

The source codes used for examples and exercise in this text are available from the first author's home pages. They would also be available in the result of a search on the following repository:

https://spectrum.library.concordia.ca/

ISBN 978-1-988392-04-2

20100 >

9 781988 392042