Application Development with Oracle & PHP on Linux

for Beginners

Application Development with

Oracle & PHP on Linux

for Beginners

Ivan Bayross
Sharanam Shah

SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD.
Mumbai *Bangalore* *Chennai* *Kolkata* *New Delhi*

Application Development with Oracle & PHP on Linux for Beginners
by Ivan Bayross, Sharanam Shah

Series Editor: Ivan Bayross

First Edition: August 2005

ISBN: 81-7366-684-9

Published by **Shroff Publishers & Distributors Pvt. Ltd.** C-103, T.T.C. Industrial Area, M.I.D.C., Pawane, Navi Mumbai - 400 701. Tel.: (91-22) 2763 4290 Fax: (91-22) 2768 3337 E-mail: spdorders@shroffpublishers.com Printed at Decora Printers, Andheri, Mumbai.

Foreword

In the early Linux days there was always a scarcity of tried and tested, production strength, database software for Linux. This issue was answered to a very large extent by two database products, Postgres and MySQL.

Today there are a number of commercial database products officially supported on Linux, including industry heavyweights Oracle and DB2. Oracle is an excellent choice to keep data consistent. Since Oracle supports the use of database transactions, it provides major enhancements over MySQL.

One of the most common reasons people choose databases such as MySQL over Oracle is price. And the price of Oracle isn't necessarily a stumbling block, either. A free copy of Oracle for development and learning purpose is freely available.

Oracle, is the world's largest selling RDBMS product. It is estimated that sales of Oracle database products account for around 80% of the RDBMSystems sold worldwide. These products are constantly undergoing change and evolving. The current, version, of this product is Oracle 10g.

Oracle, combined with the Apache Web server and PHP, makes Linux a very attractive platform for developing and running Web-enabled database applications of all sizes. All three are very flexible tools, and in practice the combination is remarkably easy to use.

This material has been written to provide commercial application developers an insight into how to use Oracle as the data store with PHP as the programming environment of choice all run on Linux.

Linux being a very popular, multi user, multi tasking and multi threaded operating system from the open source stable of products.

Apache is a Web server that runs best on Linux and is used by almost 85% of web site hosting companies to deliver HTML content to client browsers.

To be to able to drive dynamic HTML page content, bound to user demand, an efficient, slick, simple to use yet powerful database is an absolute must. Oracle plays this role flawlessly.

This is the programming framework on which almost all commercial web sites run.

Having a framework is just great, however, without a proper programming environment that can exploit the power and speed of such a framework efficiently the framework itself is largely useless.

PHP is the most widely used programming language for this framework. PHP as a Server side scripting language is especially crafted to use the power and speed of such a framework. Programs written in PHP run smoothly on Linux and Apache with Oracle as the data store, delivering dynamic web content, 24/7.

Any programmer that acquires strong development skills using this framework will always be in demand and is sure of always being able to make a good living selling these skills to the highest bidder.

Ivan Bayross & Sharanam Shah

Table Of Contents

SECTION I: SETTING UP THE FRAMEWORK

Understanding The Framework

Programmers the world over are always looking for stable, tried and trusted, programming environments, that run on a standard O/s, network, framework to craft commercial applications with. A stable, tried and tested, programming environment that works well with an O/s, is runnable on a network, always goes a long way in ensuring an application that is scalable and secure.

These kind of stable, scaleable, environments are often called production environments. This means good quality, production level, code can be developed using such an environment.

Another vital ingredient of any commercial application today, is a **D**ata**B**ase **M**anagement **S**ystem (i.e. DBMS). A DBMS is always used to store valid business data for future reference so that business managers can be empowered to make informed business decisions.

The Internet is the largest, heterogeneous, stable, network in existence today. Hence, it makes the most sense to craft commercial applications that can run successfully on the Internet.

This will permit any business entity to have worldwide reach in capturing, validating and storing its business data, irrespective of which part of the world the business model is being run in and which part of the world is being used to control and monitor the business processes of the business entity.

To recap, programmers the world over are really looking for the best programming environment, O/s and DBMS from which to craft Internet enabled commercial applications to make use of the worlds biggest and most stable network.

PHP, Linux, Apache Web Server and Oracle, are four critical ingredients that permit Internet based, commercial applications, to run smoothly and successfully.

There cannot be a better choice of **R**elational-**DBMS** (i.e. RDBMS) than Oracle. This is an RDBMS perfectly suited for a commercial application developer's needs. This is an RDBMS that has stood the test of time and has matured into production strength, product over the years. It is robust, secure, scalable relatively easy to install and maintain on both M.S. Windows and Linux and several other O/s as well.

M.S. Windows and Linux are **O**perating **s**ystems (O/s), which are used by almost 90% of the world's computer population. Both are stable, secure, multi user, multi tasking and definitely scalable.

PHP, as a programming environment is completely stable, has fully functional, built-in programming libraries from which scaleable, object oriented commercial applications can be crafted. Commercial applications crafted from PHP work perfectly well when used with **1** user as with **100** or a **1000** users. PHP has built in, native, connectivity to Oracle. Finally, it is available completely free of cost. This is the icing on the cake. Additionally, there are several excellent **I**ntegrated **D**evelopment **E**nvironments available on the Internet for free download and use today for both M.S. Windows and Linux.

All commercial applications are forms based. A form is designed in some kind of graphical editor using standard components, such as Text Boxes, Text Areas, Radio Buttons, Check Boxes and so on. These are the components via which the data entry operator captures business data, generally using a keyboard.

Any form so created, always requires a Client side, runtime environment, to execute in. That simply increases the cost of deploying the commercial application. Each Client computer must have a special runtime environment installed, which definitely has costs involved. Sometimes, due to the large numbers of client computers on which the application runs, run time environment costs can become quite prohibitive.

PHP resolves the need of any client side, runtime environment. This is because with PHP, HTML is used as the forms creation environment. HTML is a very lightweight, forms creation environment. Forms crafted in HTML run in a Browser. Almost every O/s worth its salt comes bundled with a free Browser. Thus commercial application deployment costs are kept to the very minimum (i.e. zero).

Since data capture forms, in PHP based commercial applications are crafted from HTML, a Web Server is required to deliver the appropriate form to the Web Browser that requested its use, (i.e. essentially as an HTML page). Here, the Web Server used in 95% of all web sites across the globe to deliver web site material is **Apache**. It's a **free**, industrial strength, Web Server, from the Open Source stable, that's completely flexible and very extensible. Apache is being used as the Web Server of choice in this material.

PHP and Apache are products from the Open Source stable, completely free of cost, very flexible, scaleable and quite secure. PHP and Apache are Open Source products that are available for the M.S. Windows framework as well as Linux which has gone a very long way in their universal adoption for developing, industrial strength, Internet / Intranet mountable, commercial applications which do not cost an exorbitant price to develop, debug, and deploy.

Oracle has always extended its support openly to the Open Source movement. So much so there are several Oracle DBMS engines available on the Oracle home site, available for free download (for development purposes only **NOT deployment**) that have been written to run on perfectly on Linux.

Linux is a Server O/s, that is standard, stable, scaleable, secure, multi user and multi tasking. Often available for free download from a number of web sites and mirror sites, worldwide **or** for very nominal costs, when compared to other O/s available in the market today.

Hence, complete industrial strength, commercial applications can be crafted in a programming environment and O/s that costs **almost nothing** to **develop** and **deploy**. This I believe is the whole truth for PHP, Apache, Linux and Oracle being chosen as the programming environment & framework of choice for such a multitude of programmers worldwide.

To conclude the following framework will be built:

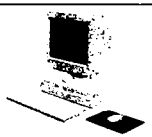
- **Red Hat, Enterprise Advanced Server** (i.e. a specific Linux distribution) as the **Operating System**
- **Oracle Database 10g** as the **RDBMS**
- **Apache Web Server** as a **Web Server**
- **PHP** as the scripting language (installed as an Apache module)

The framework in this material will include a single server serving as an Application as well as a Database Server.

This simply means a **single computer** will function as an application and a database server:

- **Application Server** - Apache Web Server with PHP as a module in the Apache Web Server will be installed on the Red Hat Enterprise Linux Advance Server
- **Database Server** - Oracle Database 10g will also be installed on the Red Hat Enterprise Advanced Server (i.e. on the same computer)

Diagram 1.1 and diagram 1.2 depicts the physical framework that is going to be built.



Node Type	Linux Server
Host Name	ROSE
IP	192.168.0.3
Processor	Athlon 1.7
RAM	512
O/S	Red Hat Enterprise Linux Advance Server 3 (Update 2)
Oracle Version	10g (Release 10.1.0.3)
Apache Version	2 (Release 2.0.53)
PHP Version	5 (Release 5.0.3)

Node Type	Clients (Windows / Linux based)
Host Name	Lily, Orchid or (Can also be the same machine for testing)
IP	192.168.0.1, 192.168.0.2 or 192.168.0.3
Processor	Athlon 1.0
RAM	512
O/S	Windows XP, Fedora Core I, Red Hat E.L. A.S. .

Diagram 1.1

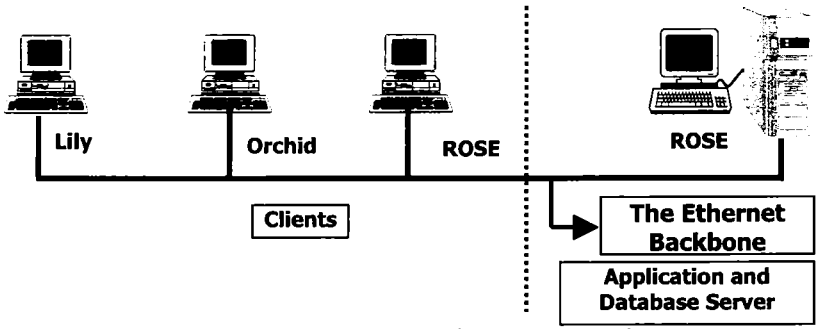


Diagram 1.2: The Complete Framework.

The Ethernet backbone could be cabled, fiber optic, wireless, infrared, with hub, without hub, with Switch, without Switch and so on depending on whether the network topology is **bus** or **star**.

In this type of architecture the **clients** and **server/s** are **always connected** to one another when switched on and active.

In this framework, the application must run in a client based application called a **Web browser**, (i.e. *Internet Explorer, Netscape Communicator, Opera are examples*). Another application must run on the Server appropriately called a Web server, *Apache Web Server being chosen in this case.*

It is these two applications that setup a link between them when a **client** and **server** talk to one another. The most common protocol (*language of communication*) used to set up this link is HTTP. The two applications **i.e.** Web browsers and Web servers communicate using a **Request / Response** paradigm. What this really means is that when a Web browser wishes to communicate with a Web server it broadcasts such a request blindly to **port 80** on which **all** Web servers listen. The broadcast will contain the **ip** address of the Web server with which the Web browser wishes to communicate. All Web servers hear the broadcast but only the Web server whose **ip** is contained within the broadcast, replies.

Once the Web server replies it immediately terminates the link setup between the Web browser and itself. In fact the Web server immediately forgets which Web browser contacted it, hence the term **Reply/Response communication paradigm**.

This really means that there is no permanent link setup between the client and server computers. Any link setup is valid **only for the duration** of the Request and Reply paradigm. Primarily this is done to decongest network bandwidth and permit the millions of Web browsers and Web servers to communicate freely and effectively on the Internet.

As can be seen, this really is a huge change from a standard wired Ethernet backbone used for commercial applications.

HOW The Framework works

The basic Client / Server framework on the Internet is as shown in diagram 1.3.

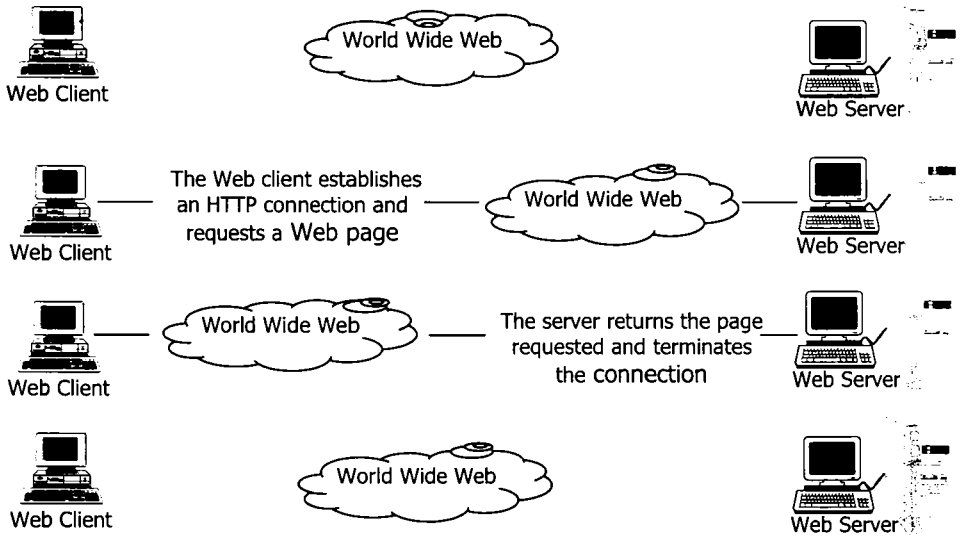


Diagram 1.3: The Request/Response Internet

Apache is the **Web server** responsible for responding to requests received from client browsers for information. **Oracle** is the **database** in which such information is stored. **PHP** is the **middleware**, programming environment of choice that can**:**

1. Respond to such information requests being processed by the Web Server Apache
2. Access the Oracle database tables where the information requested is stored
3. Convert this to HTML
4. Return this HTML to the client browser via Apache Web server

Thus servicing the client's request for information.

All this is because a Web server (*Apache2*) **cannot communicate** directly with a database management system (*Oracle*) hence the PHP program code plays the role of mediator.

Decomposing The Server Side Architecture

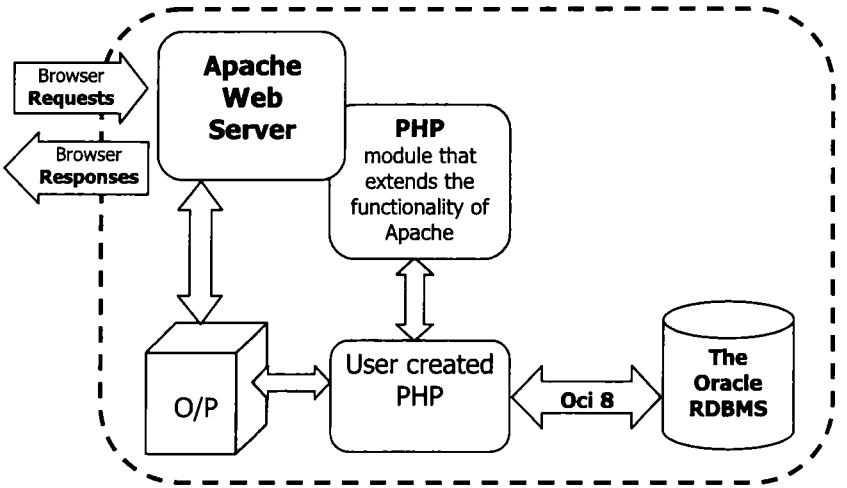


Diagram 1.4: The Server Side Internal Architecture.

Diagram 1.4 decomposes the Web server side framework that works together in harmony to respond to a client browser's request for information.

Now that the Request/ Response paradigm of the Internet and the framework on which this paradigm can be implemented is known, it is necessary to actually Setup / Install such a framework on a Linux box to develop applications on.

This will involve the installation (*and configuration, where applicable*) of:

- Red Hat Enterprise Linux Advance Server 3 as an O/s
- Oracle Database 10g
- Apache Web Server 2
- PHP 5
- Configure connectivity b/w Apache, PHP and Oracle

On an appropriate hardware platform.

SECTION I: SETTING UP THE FRAMEWORK

Setting Up The Oracle 10g Database

Downloading & Obtaining The Product

Oracle Database 10g is available for download from the Oracle website **http://www.oracle.com/technology/software/products/database/oracle10g/htdocs/linuxsoft.html**. Refer diagram 2.0). This is how Oracle products being installed on Linux for the first time are obtained.

REMINDER

All software downloads from the Oracle website are free. Each comes with a development license that allows the use of full versions of the products only while developing and prototyping applications. For deployment and commercial use Oracle products with full-use licenses can be bought at any time from the online Oracle Store or from Oracle sales representatives.

Once the Db files are downloaded burn them onto a CDROM, the installation of Oracle Database 10g is then carried out from this CD-ROM. Specific details of this process are described later in this material. The installation process of Oracle Database 10g Server **depends** on the Operating System. Currently Oracle 10g installs on the following Linux operating system versions:

- **Red Hat Enterprise Linux 3 (Update 2)** (This will be the O.S. of choice in this material)
- **SuSE Linux Enterprise Server (SLES) 8** with service pack 3 or later
- **SuSE Linux Enterprise Server 9**

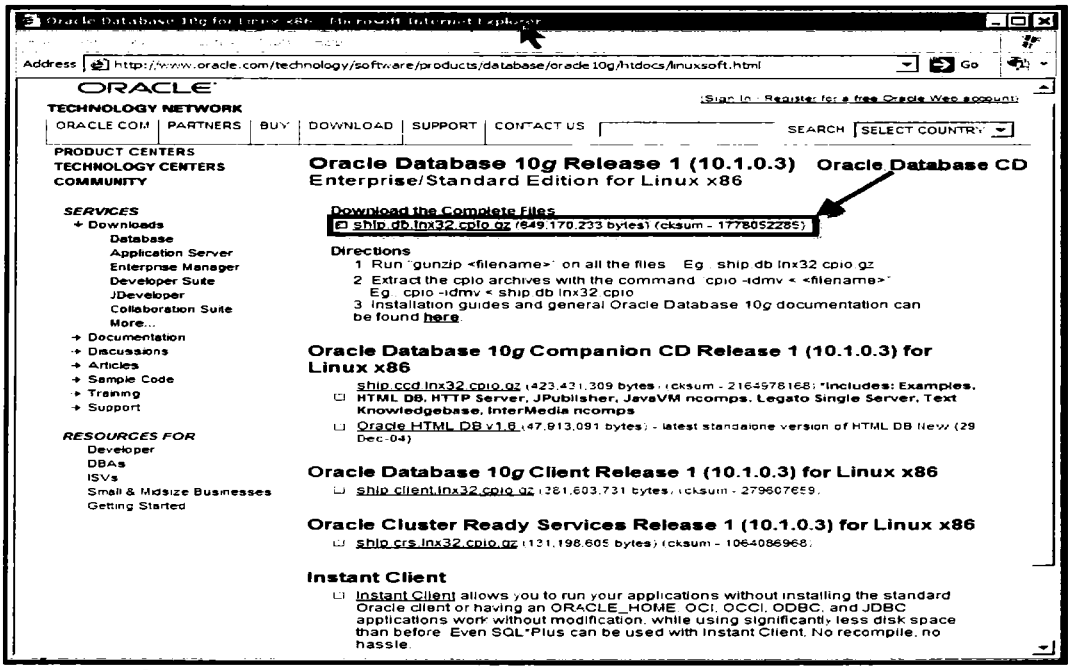


Diagram 2.0: Downloading Oracle Database 10g Release 1(10.1.0.3)

Basic System Requirements

- 512 MB of RAM (Minimum recommended on the mother board)
- 1 GB of SWAP space (Generally available on the hard disk drive)
- 400 MB disk space in /tmp
- 1.5 GB of free hard disk space for the software installation files
- 1.2 GB of free hard disk space for associated database files

Oracle 10g Database will only run and install in the following Linux kernel version (**or** a later version):

- Red Hat Enterprise Linux 3 (Update 2)
  - 2.4.21-15.EL (This is the default kernel version)
- SuSE Linux Enterprise Server 8 (x86-64)
  - 2.4.21-185-smp
- SuSE Linux Enterprise Server 9
  - 2.6.5-7.97

The following packages (**or** later versions) must be installed as part of the O/s:

- Red Hat Enterprise Linux 3 (Update 2):
  - **make**-3.79.1
  - **gcc**-3.2.3-34
  - **glibc**-2.3.2-95.20
  - **glibc-devel**-2.3.2-95.20
  - **glibc-devel**-2.3.2-95.20 (32 bit)
  - **compat-db**-4.0.14-5
  - **compat-gcc**-7.3-2.96.128
  - **compat-gcc-c++**-7.3-2.96.128
  - **compat-libstdc++**-7.3-2.96.128
  - **compat-libstdc++**-devel-7.3-2.96.128
  - **gnome-libs**-1.4.1.2.90-34.1 (32 bit)
  - **openmotif**21-2.1.30-8
  - **setarch**-1.3-1
  - **libaio**-0.3.96-3
  - **libaio-devel**-0.3.96-3
- SuSE Linux Enterprise Server 8 (x86-64):
  - **make**-3.79.1
  - **gcc**-3.3-43
  - **gcc-c++**-3.3.3-43
  - **glibc**-2.2.5-213
  - **glibc-32bit**-8.1-9
  - **glibc-devel-32bit**-8.1-9
  - **openmotif**-2.2.2-124
  - **libaio**-0.3.96-3
  - **libaio-devel**-0.3.96-3
- SuSE Linux Enterprise Server 9:
  - **gcc**-3.3.3-43
  - **gcc-c++**-3.3.3-43
  - **glibc**-2.3.3-98
  - **libaio**-0.3.98-18
  - **libaio-devel**-0.3.98-18
  - **make**-3.80
  - **openmotif-libs**-2.2.2-519.1

REMINDER

Do not attempt an install of Oracle Database 10g if there is **256 Mb** of RAM (or less) on the mother board as the installer will crash at link time.

Note down the **checksum** of the file being downloaded. This checksum will be used to check the **validity** of the downloaded file.

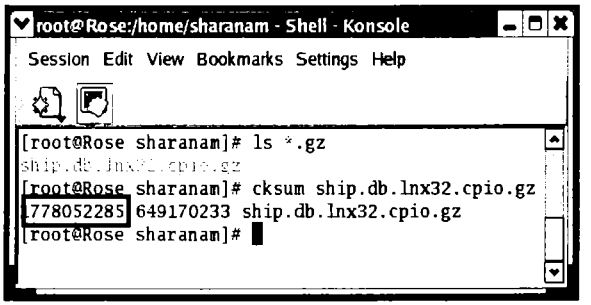


Diagram 2.1.1: Verification of the file using the **cksum** command.

Uncompressing Oracle Database 10g Software

Verify The File downloaded

To check whether the downloaded file is the same as the one on the Oracle Server for download, and is free of any corruption, verify its checksum by using the following:

<System Prompt> cksum ship.db.lnx32.cpio.gz (Refer diagram 2.1.1)

Compare this number with the one (noted and) specified on the website from where the file was downloaded. If both the numbers match exactly, the file has been downloaded correctly else the file has been corrupted and needs to be downloaded again.

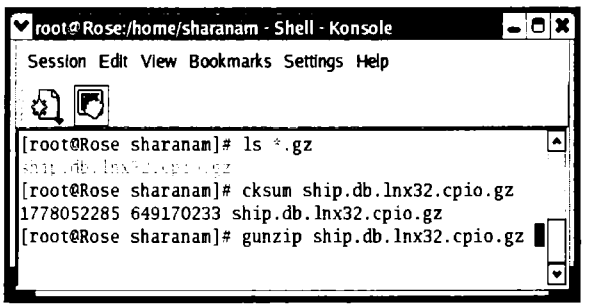


Diagram 2.1.2: Extracting the contents held in the .gz file

If the checksum matches, proceed with the following steps:

Uncompress the file using the following command

<System Prompt> gunzip ship.db.lnx32.cpio.gz (Refer diagram 2.1.2)

The next step is to unpack the file. To do so, use the following:

<System Prompt> cpio -idmv < ship.db.lnx32.cpio

*(The above command unpacks the file's contents into a folder named **Disk1**)* (Refer diagram 2.1.3)

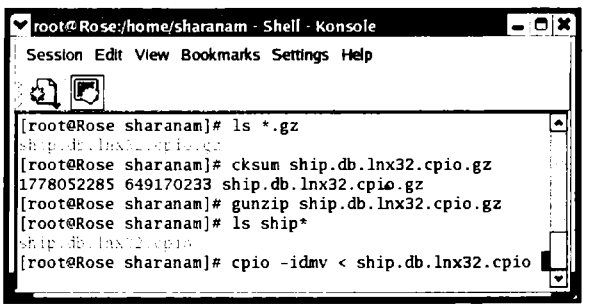


Diagram 2.1.3: Extracting the contents held in the .cpio file

Burning Oracle Database 10g CD

Using a command based utility **mkisofs** burn the Oracle Database 10g directory (Disk1) on the CDROM. To do this the device number of the CD-Writer being used must be known. Get this information using: (Refer diagram 2.2)

<System Prompt> cdrecord -scanbus

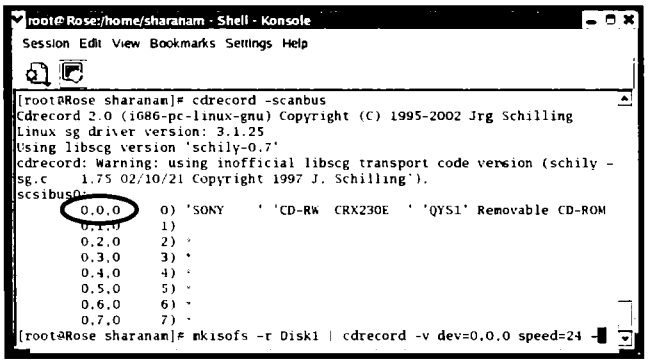


Diagram 2.2: Burning the Oracle Database 10g Directory.

This command provides a listing of the scanbus details, which includes the **dev** number of the CDROM drive. Note this down and use the following: (Refer diagram 2.2)

<System Prompt> mkisofs -r Disk1 | cdrecord -v dev=**0,0,0** speed=24 -

This command is responsible to do the actual CD-ROM burning process. **–r** indicates recursive that is all the contents available under all the levels of the directory structure. **Disk1** is the directory to be burned on the CD-ROM. **cdrecord** indicates the driver that will be used in the entire burning process. **–v** switches the mode to verbose which depicts each and every details of the entire process. **dev** indicates which device to use for burning the CD-ROM. **speed** is the rate at which the burning process will take place.

After typing in this command, and pressing Enter, a screen similar to the one in diagram 2.3 will appear. This shows the CD's burning progress.

Once completed, the system prompt will be seen as in diagram 2.3.

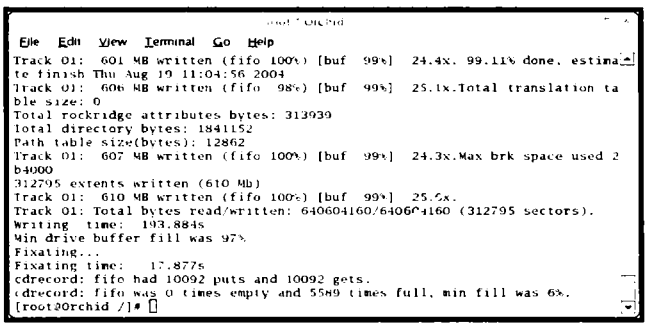


Diagram 2.3: The Process of Burning the CDROM.

PRE-INSTALLATION Tasks For The Oracle 10g Database

Before starting the Oracle Database 10g installation, several parameters must be checked.

To perform these checks login in as **root** since some of these parameters can be checked only if the user logged in is **root** or **s**witch **u**ser to **root** using command **su - root.**

Checking Memory And Swap Space

For a successful installation of Oracle Database 10g, the system is required to have at least 512MB of RAM and 1GB of swap space. Verify this as: (Refer diagram 2.4)

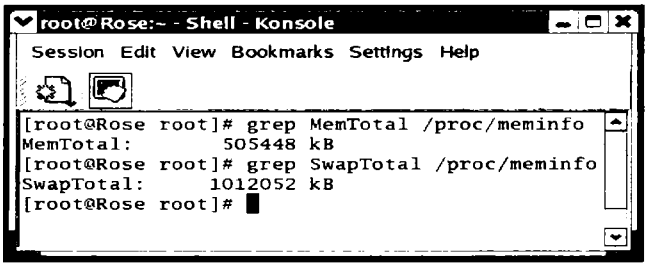


Diagram 2.4: Checking Memory and Swap space.

```
<System Prompt> grep MemTotal /proc/meminfo
<System Prompt> grep SwapTotal /proc/meminfo
```

Checking /tmp Space

The Oracle Universal Installer requires 400MB of free space in the /tmp directory on the hard disk. The command to check this is as follows: (Refer diagram 2.5)

```
<System Prompt> df /tmp
```

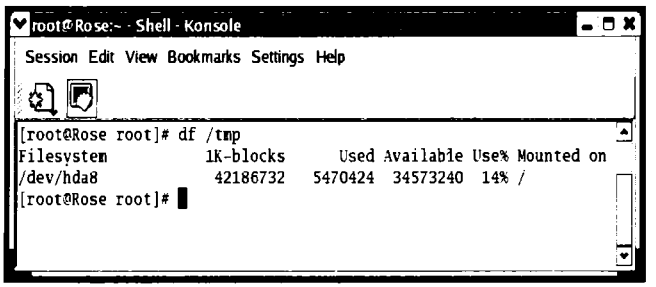


Diagram 2.5: Checking /tmp space

HINT

Normally, / (i.e. root) is the mount point of the entire Linux installation tree, while the **/tmp** directory is under the / partition. This means that **/tmp** may not be limited to a specific size, but can grow to occupy any free space on the HDD.

Checking The Kernel Version

The kernel version required to install Oracle Database 10g is **2.4.21-15.EL or higher**. The command to find out the current version of the Linux kernel is as follows: (Refer diagram 2.6)

```
<System Prompt> uname -a
```

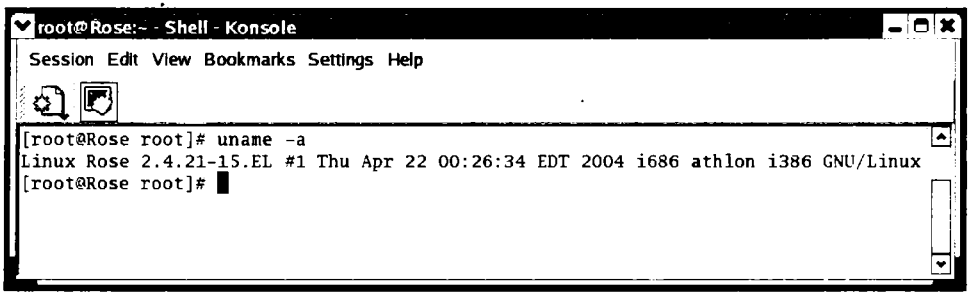


Diagram 2.6: Checking kernel version

Checking The Existence Of gcc, glibc-devel, glibc-headers, glibc-kernheaders,cpp, binutils, openmotif, setarch RPMs

Before starting the installation of Oracle Database 10g, the system needs to be checked for the existence of the above software.

WARNING

If the ***Custom → Everything*** option was selected while installing Linux, all these RPMs will already exist on the system. If Linux was a custom install, then any missing RPMS will have to be manually installed **prior** the actual Oracle Database 10g installation.

Issue the following command at the system prompt, to check the existence of the required RPMs **:** (Refer diagram 2.7)

```
<System Prompt> rpm -q make gcc glibc glibc-devel glibc-devel compat-db
                compat-gcc compat-gcc-c++ compat-libstdc++ compat-libstdc++
                gnome-libs openmotif setarch libaio libaio-devel
```

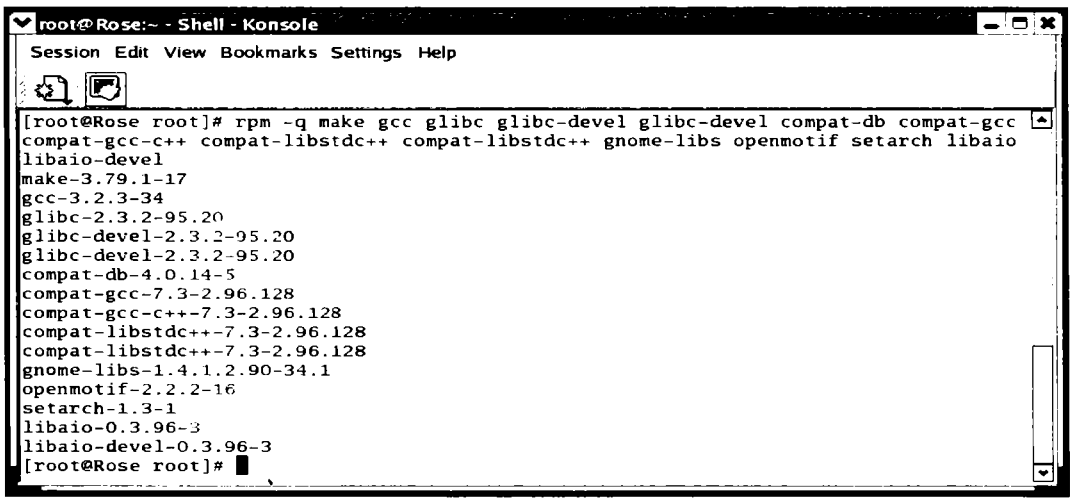


Diagram 2.7: Checking version of packages

REMINDER

Oracle Database 10g while being installed, detects the Linux distribution via a file called **redhat-release**. Not all Red Hat Linux distributions support installation of Oracle Database 10g. Oracle 10g Database can be installed only on Red Hat Enterprise Linux Advance Server 2.1 and 3.

Verify that the redhat-release RPM is installed on the Red Hat system as: (Refer diagram 2.8.1)

<System Prompt> rpm -q redhat-release

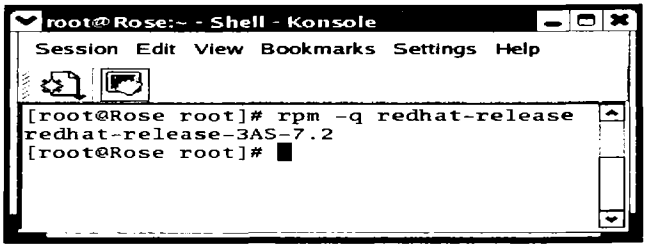


Diagram 2.8.1: Verifying the redhat-release rpm.

This RPM is important because the Oracle installer will check this file for Red Hat Enterprise Linux AS 3 **or** Red Hat Enterprise Linux AS 2.1 as these are the only Linux operating systems on which Oracle 10g will install. Without this RPM, Oracle10g will not be able to recognize the O.S. **and refuse to install.**

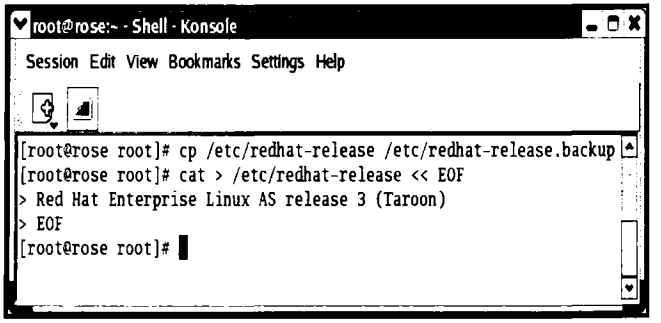


Diagram 2.8.2: Editing **/etc/redhat-release** file.

WARNING

To install Oracle Database 10g on an unsupported Linux distribution such as **Fedora Core**, a file **/etc/redhat-release** needs to be edited to make Oracle Database 10g believe it is running on a appropriate Linux distribution (Since it does not support Fedora Core). To accomplish this, type in the following command at the prompt:

<System Prompt> cp /etc/redhat-release /etc/redhat-release.backup

The above command creates a copy of the **redhat-release** file on the HDD named redhat-release.backup. This will be used to restore the original copy of redhat-release later. This must be done only **after** the successful installation of Oracle Database 10g. The command for this is as follows:

<System Prompt> cp /etc/redhat-release.backup /etc/redhat-release

Remember execute the above command only after Oracle 10g installation is complete.

Next, type in the following command at the prompt to **Change The Current Distribution's Identity:** (Refer diagram 2.8.2)

<System Prompt>cat > /etc/redhat-release << EOF
Red Hat Enterprise Linux AS release 3 (Taroon) EOF

Checking kernel parameters

To see these kernel parameters, issue the commands shown in table 2.1at the prompt:

Command	Parameter and expected values
cat /proc/sys/kernel/shmmax	shmmax = 2147483648
cat /proc/sys/kernel/shmmni	shmmni = 4096
cat /proc/sys/kernel/shmall	shmall = 2097152
ipcs –lm | grep "min seg size"	shmmin = 1
cat /proc/sys/kernel/sem | awk '{print $1}'	semmsl = 250
cat /proc/sys/kernel/sem | awk '{print $2}'	semmns = 32000
cat /proc/sys/kernel/sem | awk '{print $3}'	semopm = 100
cat /proc/sys/kernel/sem | awk '{print $4}'	semmni = 128
cat /proc/sys/fs/file-max	file-max = 65536
cat /proc/sys/net/ipv4/ip_local_port_range	ip_local_port_range 1024 65000

Table 2.1

The above commands creates the output as seen in diagram 2.9

Kernel parameters having values greater than or equal to the ones listed in table 2.1 should be left unaltered. To meet the requirement specification, the **/etc/sysctl.conf** file needs to be altered. This file is primarily used during the boot process.

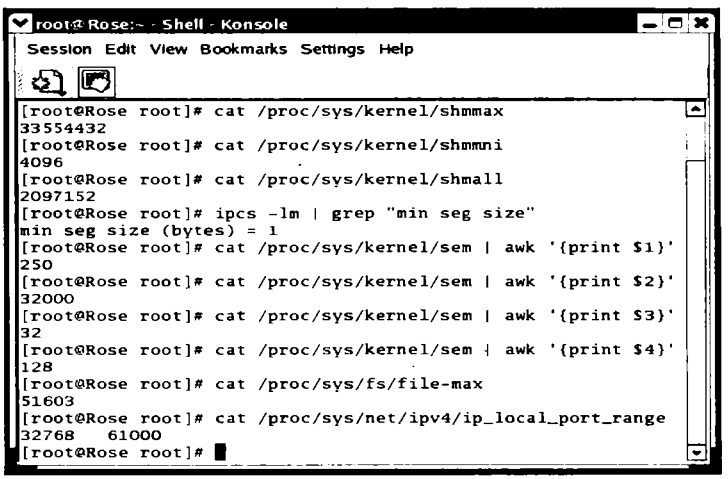


Diagram 2.9: Checking Kernel parameters

Open the above file in any text editor (eg. Kate) (Refer diagram 2.10) and carefully type in the following lines (Refer diagram 2.11):

```
kernel.shmmax = 2147483648
kernel.sem = 250 32000 100 128
fs.file-max = 65536
net.ipv4.ip_local_port_range = 1024 65000
```

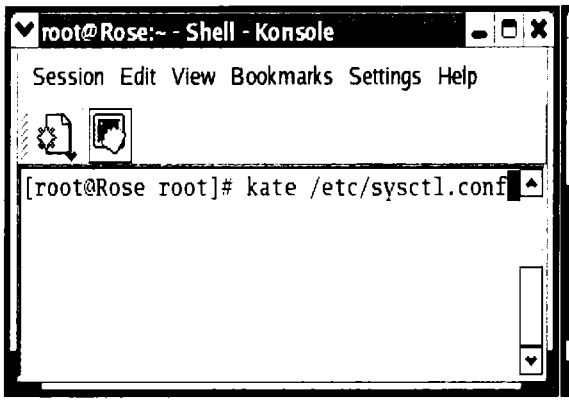


Diagram 2.10: Opening file using Kate.

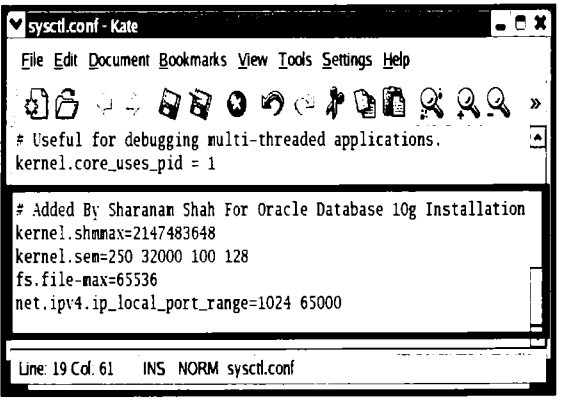


Diagram 2.11: Modification in etc/sysctl.conf file.

This modification in the file will cause the system to change the kernel parameters after each boot so as to make the system compatible for running Oracle 10g. For these changes to be effective immediately execute the following command:

```
<System Prompt> sysctl -p
```

An output as shown in diagram 2.12 will be seen.

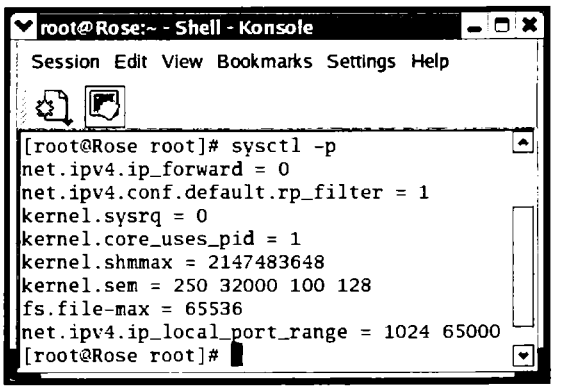


Diagram 2.12: The **sysctl –p** command.

Creating Oracle User Accounts

The next step of installation is to create the Oracle accounts and groups.
The new User/Groups that have to be created are:

- User – **oracle** (*provide any password of choice appropriately*)
- Group – **oinstall** and **dba**

Do this as follows: **(Only root can do this)**

```
(Switch to the root user if currently not logged in as root)
<System Prompt> # su -root
<System Prompt> # groupadd dba
<System Prompt> # groupadd oinstall
<System Prompt> # useradd -c "Oracle software owner" -g oinstall -G dba oracle
<System Prompt> # passwd oracle
```

The last command will prompt for a password and its confirmation as shown in diagram 2.13. Key in an appropriate password (note this down properly) and continue.

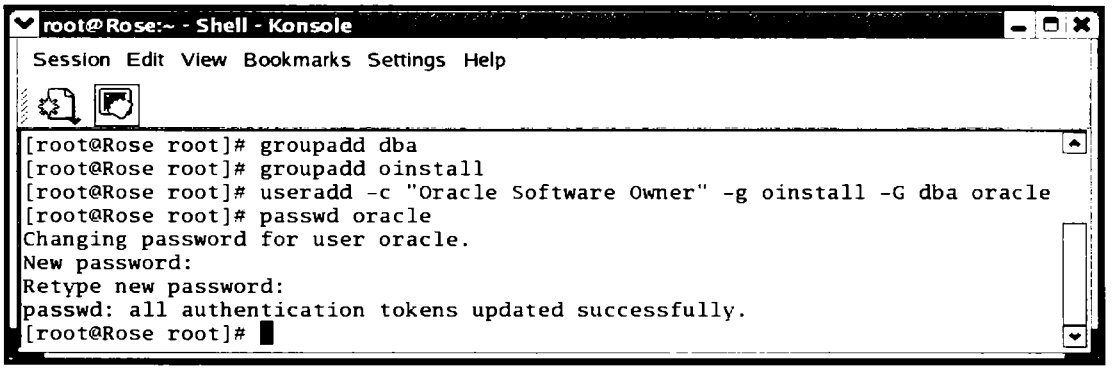


Diagram 2.13: Creating oracle account and the required groups.

Creating Oracle Directories

To install Oracle Database 10g, the following directory structure needs to be created. Refer diagram 2.14.

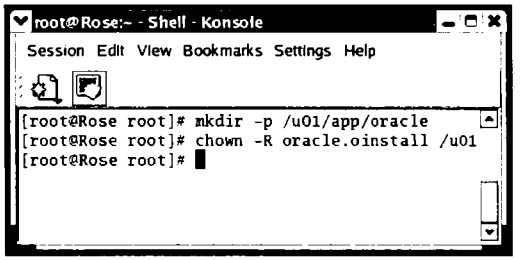


Diagram 2.14: Creating Oracle directory.

Do this as follows:

```
<System Prompt> mkdir -p
                /u01/app/oracle
```

Change the owner and group of the directories created to oracle and oinstall respectively as:

```
<System Prompt> chown -R oracle.oinstall /u01
```

Setting Oracle Environment

Since the Oracle installer namely **runinstaller** is run using the Oracle account, some environment variables of the oracle user needs to be configured and set.

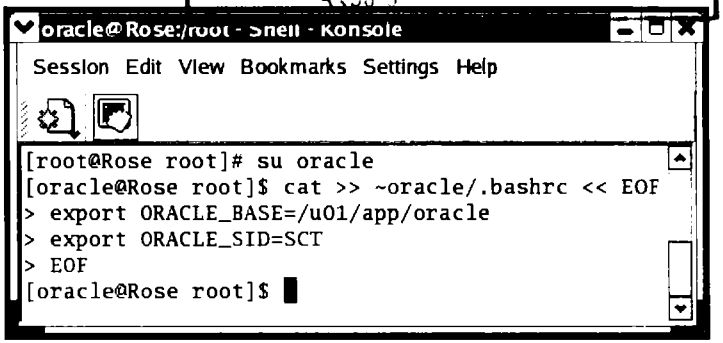


Diagram 2.15: Editing **~oracle/.bashrc** file.

To set these environment variables automatically each time the user Oracle logs in, issue the following command to enter these setting in the Oracle user's bash shell. Refer diagram 2.15

Login as Oracle:

```
<System Prompt> su oracle
[oracle@Rose oracle] cat >> ~oracle/.bashrc << EOF
  export ORACLE_BASE=/u01/app/oracle
  export ORACLE_SID=SCT
  EOF
```

The next step is to **unset** the **ORACLE_HOME** and **TNS_ADMIN** variables. Do this as follows: (Refer diagram 2.16)

```
[oracle@Rose oracle] unset ORACLE_HOME
[oracle@Rose oracle] unset TNS_ADMIN
```

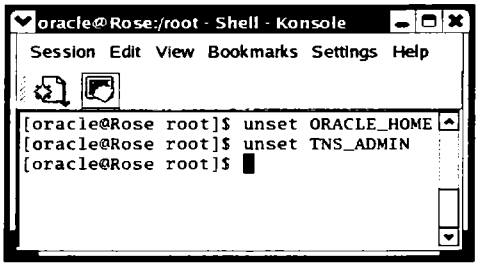


Diagram 2.16: Unsetting variables.

Once the variables are unset, change the user back to root by typing the command at the prompt (The user oracle will be logged out by doing this):

```
<System Prompt> exit
```

This will pass control back to the root user. (Refer diagram 2.17)

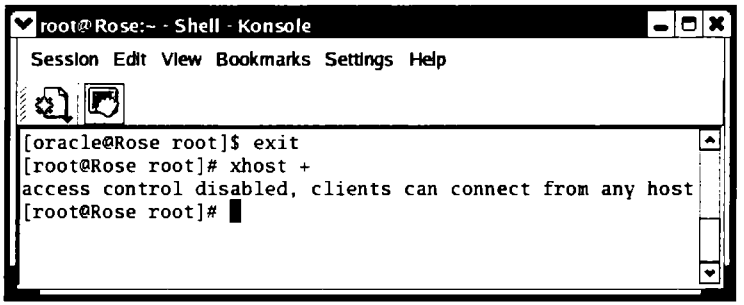


Diagram 2.17: Running xhost command.

Next the **xhost** command needs to be executed as shown in diagram 2.17. This is done to disable the access control mechanism of the system. The command is:

```
<System Prompt> xhost +
```

Before starting the installation process, verify the environment variables set earlier. For this, log in as Oracle. These settings can be verified by running the set command as follows: Refer diagram 2.18.

```
<System Prompt> su oracle
[oracle@Rose oracle] set | grep
                     ORACLE_*
```

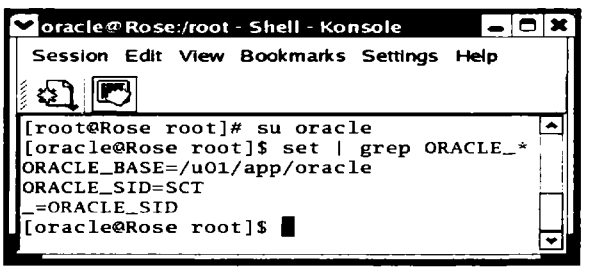


Diagram 2.18: Verifying Environment variables.

This command will allow verification of the environment variables **ORACLE_BASE** and **ORACLE_SID**.

This completes the Oracle 10g pre installation steps.

Installing The Oracle 10G Database

In order to begin the installation, mount the CDROM drive using the following command: (Refer diagram 2.19.1)

```
[root@Rose root] mount /mnt/cdrom
```

Switch to the Oracle user using **su oracle.** (Refer diagram 2.19.2)

```
[oracle@Rose oracle] /mnt/cdrom/runInstaller
```

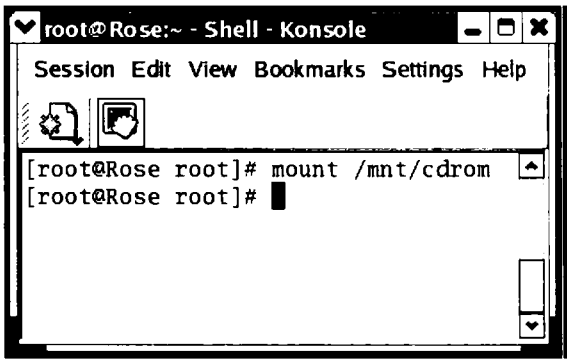


Diagram 2.19.1: Mounting the CDROM drive

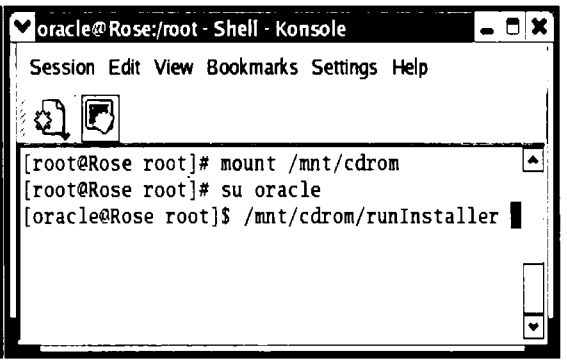


Diagram 2.19.2: Running the installer

After issuing these commands the output will be as seen in diagram 2.19.3

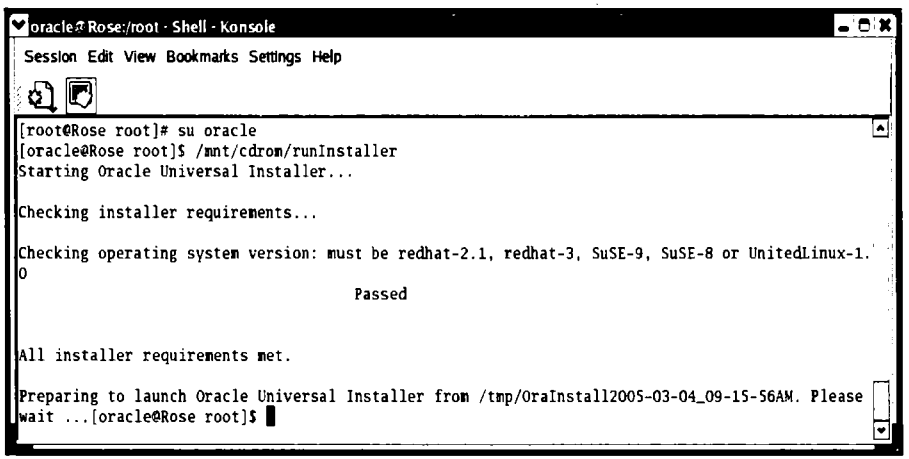


Diagram 2.19.3: Running of the Oracle Installer

After this completes, a new window similar to the one shown in diagram 2.20 appears. This screen allows two types of installation:

- Basic Installation
- Advanced Installation

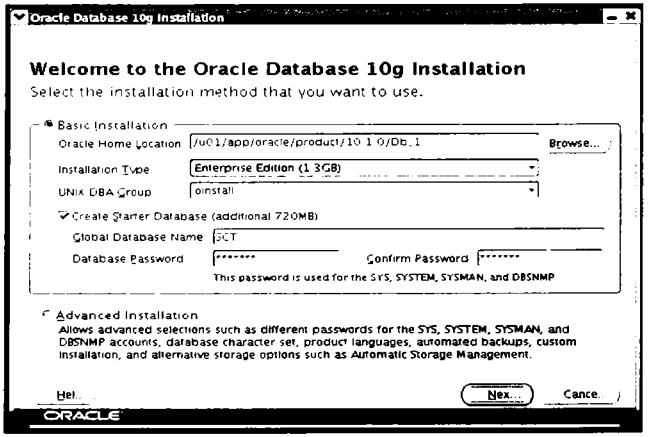


Diagram 2.20: The Welcome screen.

Select the **Basic Installation** option. This screen accepts information related to Oracle Home (**i.e.** the destination for the installation files), the Global Database Name and a common password for users such as SYS, SYSTEM, SYSMAN, and DBSNMP.

Using the **Advanced Installation** option permits different passwords for different user accounts.

REMINDER

While creating this material the **Global Database name** was set to **SCT**.

After specifying the **Global Database Name**, click Next to continue.

A progress bar is displayed as shown in diagram 2.21.0. This shows the progress of the installation process.

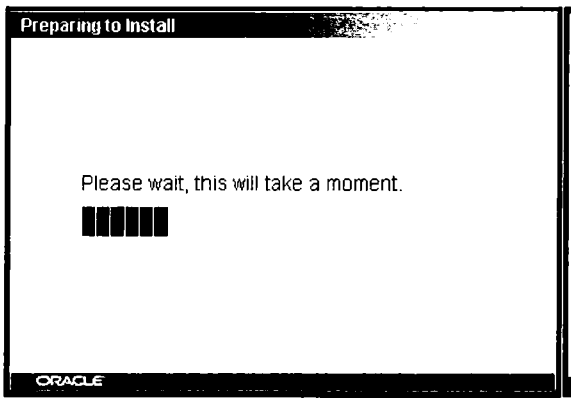


Diagram 2.21.0: Progress bar for the loading process.

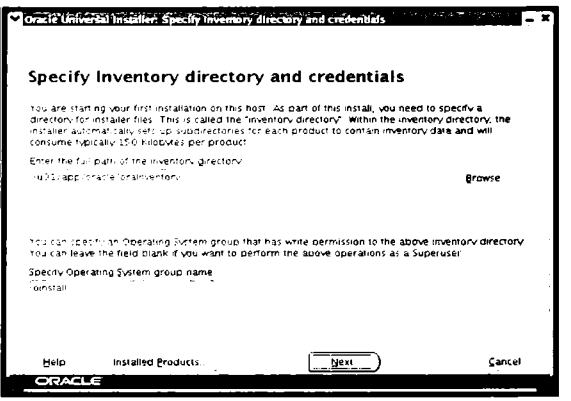


Diagram 2.21.1: The Inventory Location screen.

After the loading process completes, the screen as shown in diagram 2.21.1 appears. Keep the contents of the Inventory Location dialog box (**i.e.** make no changes). Click **Next** to continue.

The installer requests that a specific file **/u01/app/oracle/oraInventory /orainstRoot.sh** is run, as shown in diagram 2.21.2.

Do not disturb the existing console window in which the installer is running. **Open a new console window** and type in the following command. An output as seen in diagram 2.21.3 will appear.

```
<System Prompt> /u01/app/oracle/oraInventory/orainstRoot.sh
```

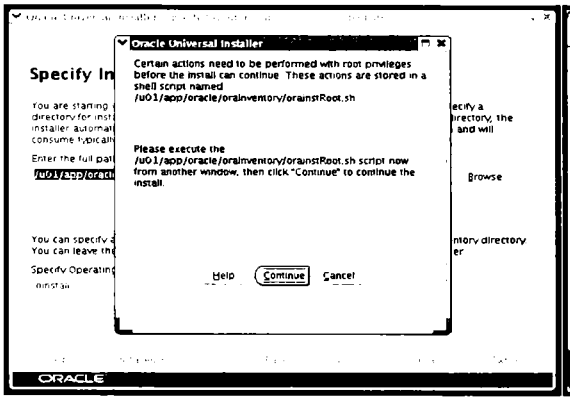


Diagram 2.21.2: Message to execute the orainstRoot.sh file.

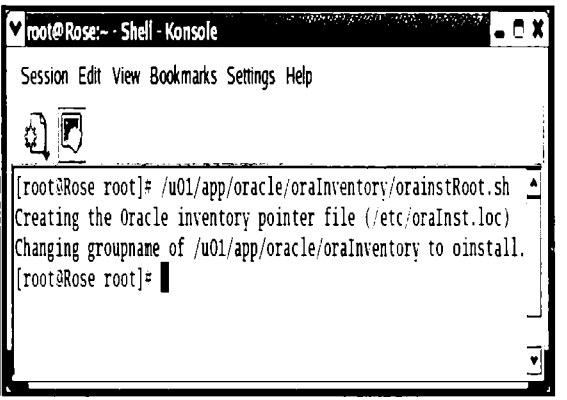


Diagram 2.21.3: Execution of the orainstRoot.sh.

Close this console window by using **exit**. Click **Continue** to proceed (Refer diagram 2.21.2). The screen, as shown in diagram 2.21.4 appears. Further loading processes occur. A progress bar, of the loading operation appears at the top right corner of the screen. The Pre-requisites checks begin as seen in diagram 2.22.1.

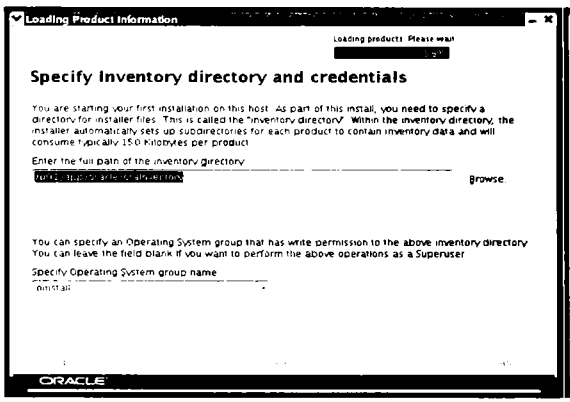


Diagram 2.21.4: Further Loading process.

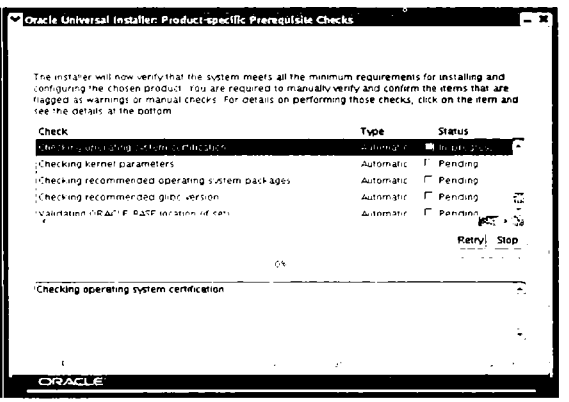


Diagram 2.22.1: Checking product specific requirements.

During this process, the Installer checks for product specific, pre-requisite checks. If successful an appropriate message is displayed as shown in diagram 2.22.2.

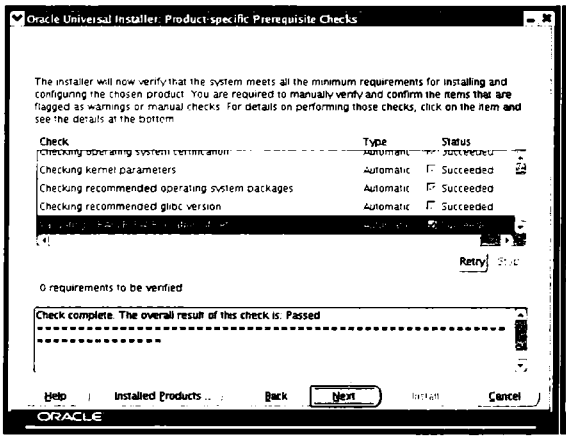


Diagram 2.22.2: Messages on checks made.

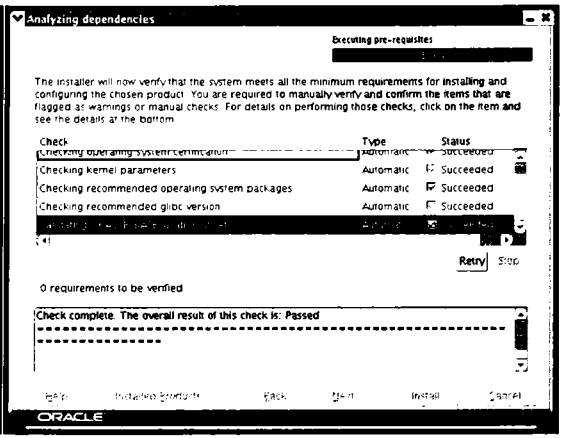


Diagram 2.22.3: Further Loading process.

Click **Next** to proceed. A progress bar appears on the top right corner as seen in **diagram 2.22.3**. The screen as shown in diagram 2.23 summarizes the settings of the installation that will be carried out now.

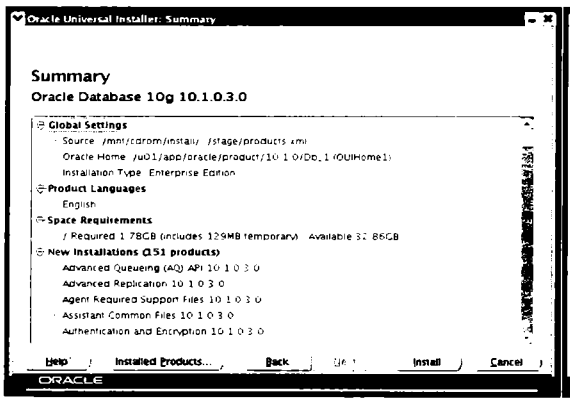


Diagram 2.23: Summary of the Installation.

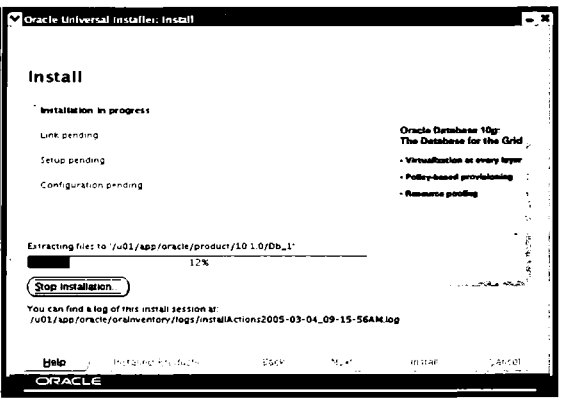


Diagram 2.24.1: Beginning of the installation process.

Click **Install** to begin the installation process. The installation process begins as shown in diagram 2.24.1 - diagram 2.24.4.

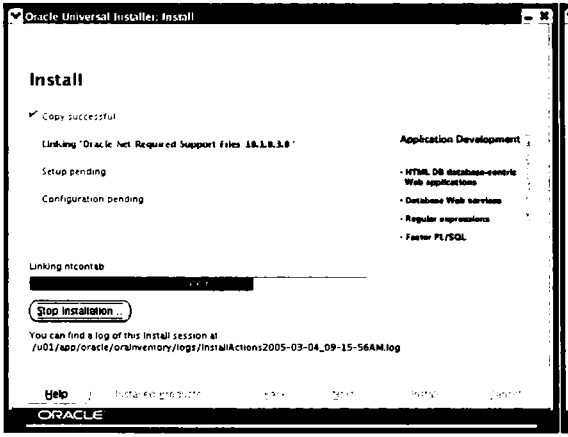


Diagram 2.24.2: Linking Progress.

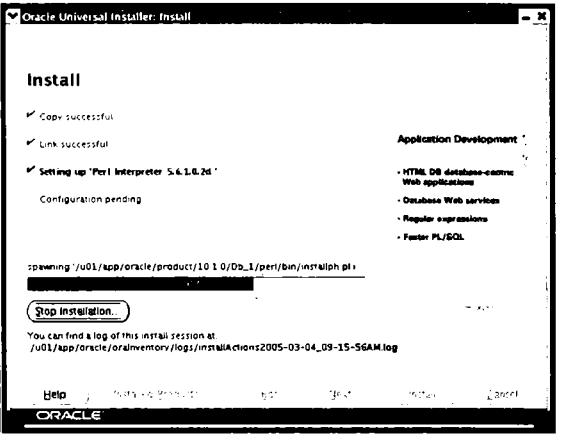


Diagram 2.24.3: Setting Up progress.

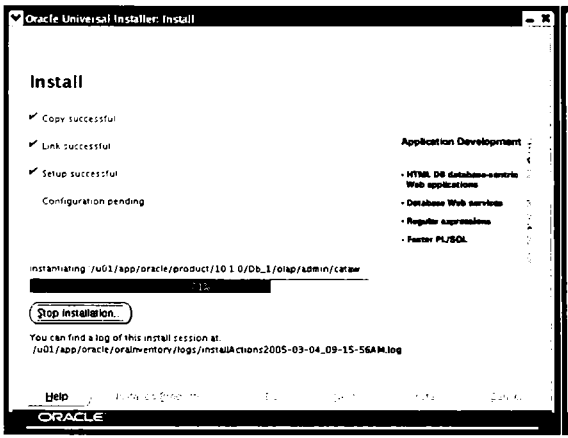


Diagram 2.24.4: Setup Procedure completed.

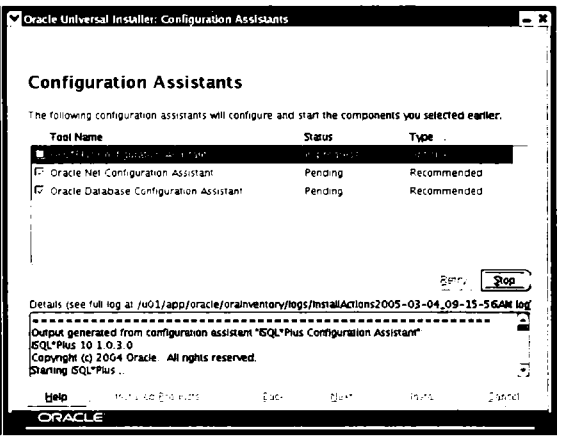


Diagram 2.25.1: The configuration process.

After the copying of files to their destination is complete, the installation process automatically proceeds to the configuration of all installed tools. Refer diagram 2.25.1.

The auto-configuration process will configure the following tools:

- iSQL*Plus Configuration Assistant (Refer diagram 2.25.2)
- Oracle Net Configuration Assistant (Refer diagram 2.25.3)
- Oracle Database Configuration Assistant (Refer diagram 2.26.1 - diagram 2.26.3)

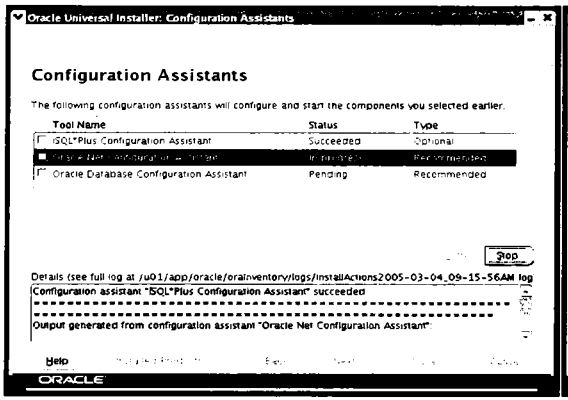


Diagram 2.25.2: iSQL*Plus Configuration Assistant completed.

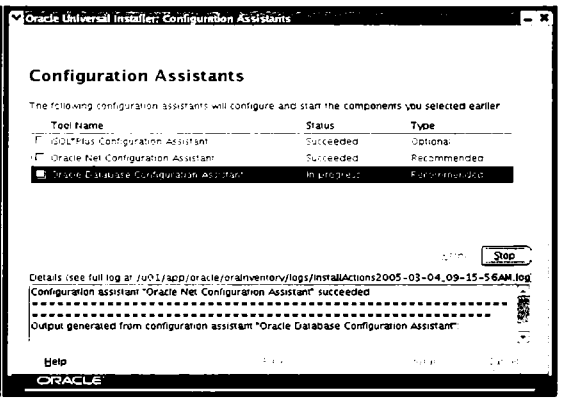


Diagram 2.25.3: Oracle Net Configuration Assistant completed.

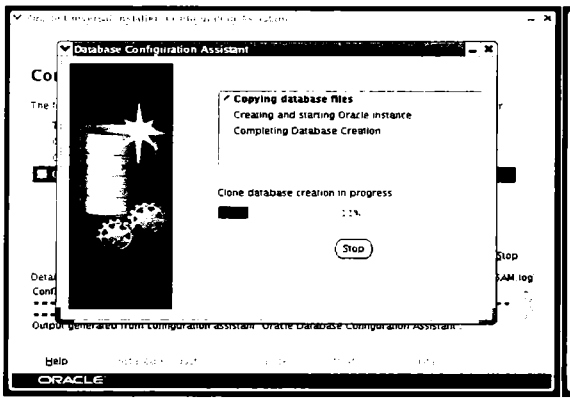


Diagram 2.26.1: Oracle Database Configuration Assistant – Copying Process.

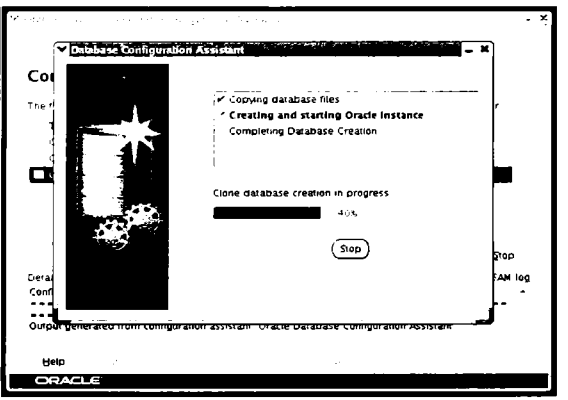


Diagram 2.26.2: Oracle Database Configuration Assistant – Creation & Starting Process.

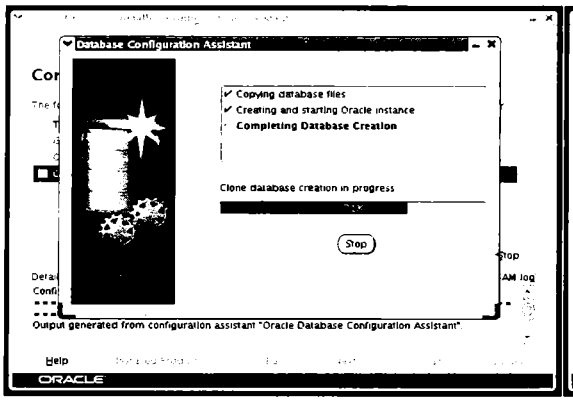


Diagram 2.26.3: Oracle Database Configuration Assistant – Completion Process.

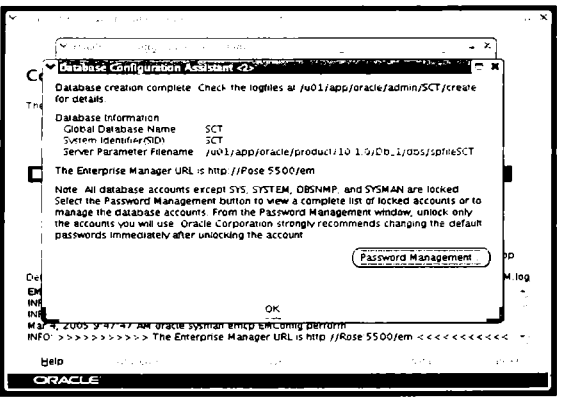


Diagram 2.27.1: The DB Configuration Assistant process completed.

2. SETTING UP THE ORACLE 10g DATABASE

This process is responsible for actual database creation on the hard disk. Once complete, a screen displaying a summary and the option to start the Password Management tool appears as in diagram 2.27.1.

Click the **Password Management** button present on the lower right corner and **unlock** the following accounts:

- **SCOTT** - Employee Management System
- **PM** - Product media, used for multimedia datatypes
- **IX** - Queued shipping, shows advanced queuing
- **SH** - Sales history, large amount of data, analytic processing
- **OE** - Order entry, intermediate topics, various datatypes
- **HR** - Human resources, basic topics, supports Oracle Internet Directory

By removing the tick mark as shown in diagram 2.27.3. Provide the desired passwords for the unlocked account. Make a note of the same.

HINT

These accounts are linked to **tablespaces,** which have helpful example exercises.

Click **OK** and the install process will return to the screen as seen in diagram 2.27.4. Now, click **OK** again and a screen as shown in diagram 2.28 appears. Click **Next**.

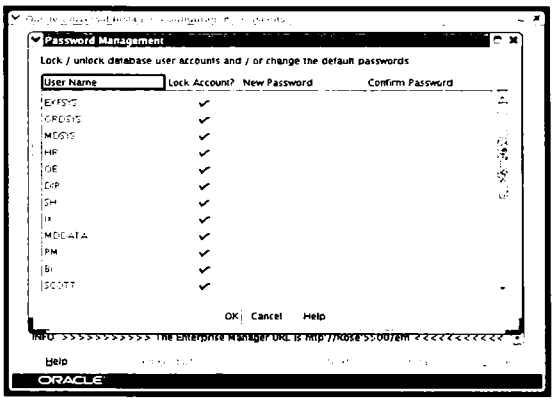


Diagram 2.27.2: The Password Management tool.

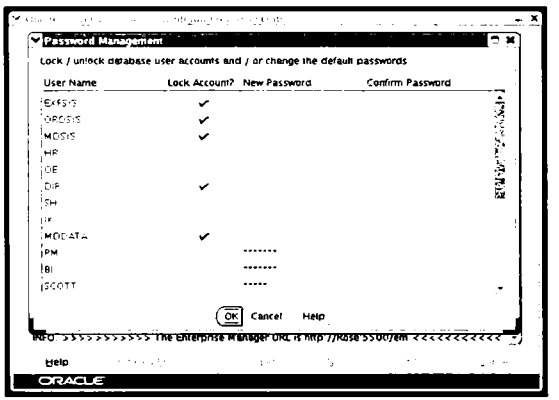


Diagram 2.27.3: Unlocking And Changing Passwords.

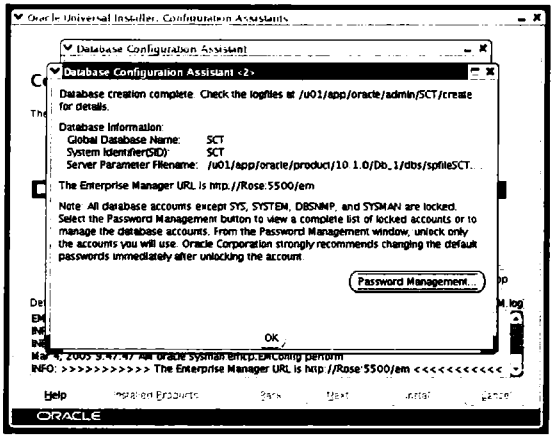


Diagram 2.27.4: The Password Management tool.

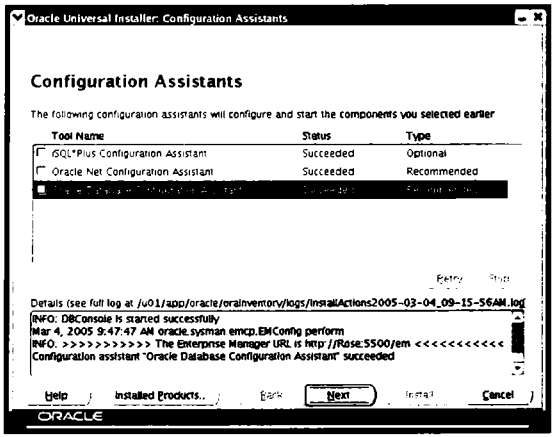


Diagram 2.28: The Configuration Process completes.

Once the configuration processes are completed a screen as shown in diagram 2.29 appears.

The installer expects the root user to run the **root.sh** file. Open a new console.

WARNING

Do not disturb the Linux console from which the oracle installer was invoked.

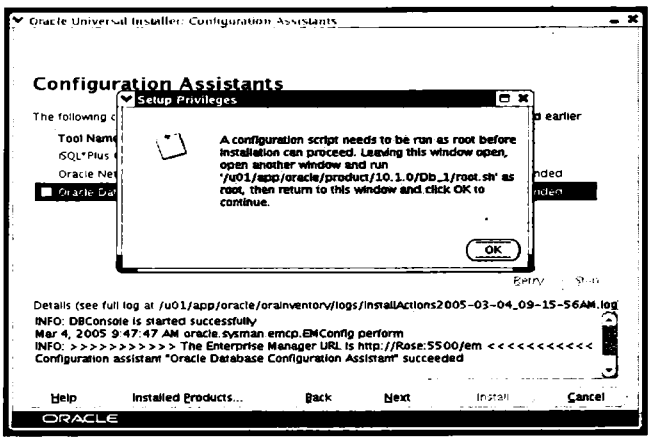


Diagram 2.29: Message to execute the **root.sh** file.

In the new console type in the following command:

```
<System Prompt> /u01/app/oracle/ product/10.1.0/db_1 /root.sh
```

A screen, as shown in diagram 2.30, appears. Press **ENTER**, when prompted for the bin directory path.

The screen scrolls with a lot of messages and finally returns to the prompt as shown in diagram 2.30.

Close this **console window** by using the **exit** command.

Come back to the screen as seen in diagram 2.29. Click **OK**.

That's the end of a successful installation. A message for the same appears as shown in diagram 2.31.

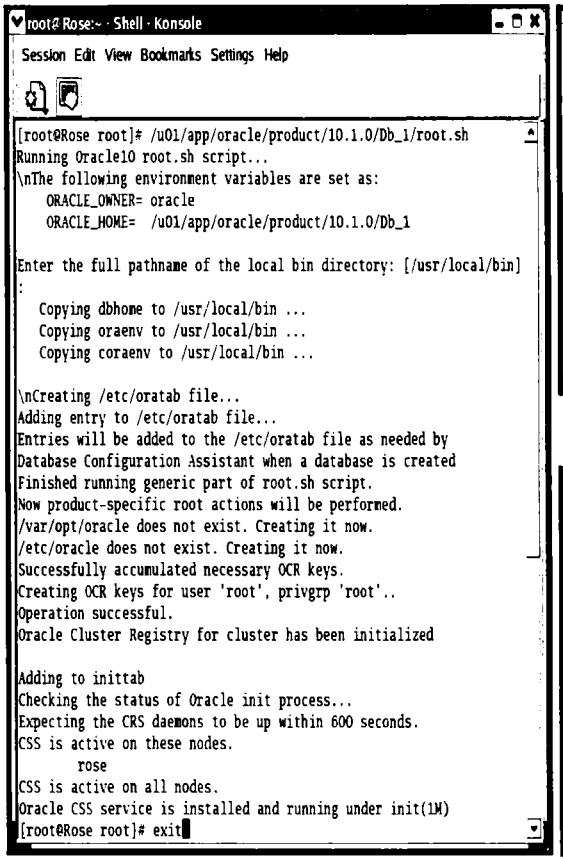


Diagram 2.30: Executing the root.sh file.

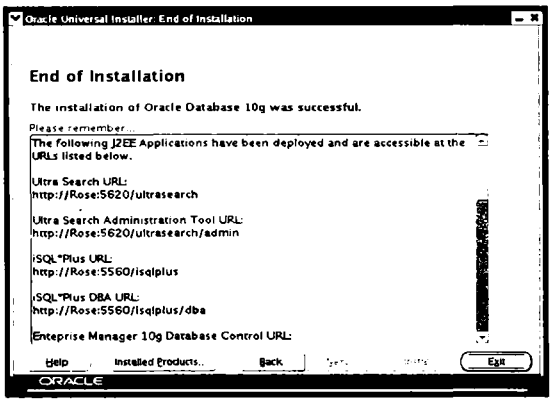


Diagram 2.31: End of Installation screen.

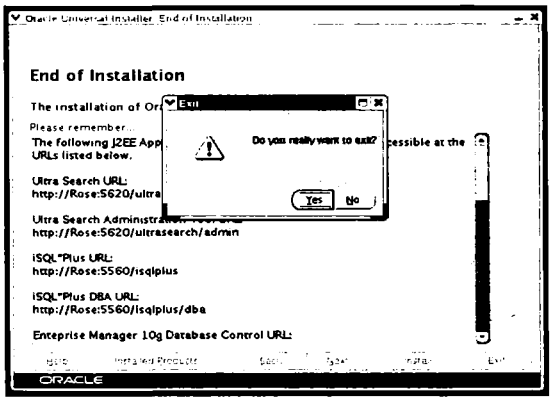


Diagram 2.32: Confirmation for exit.

Now click on **Exit**. A confirmation to exit the installer appears as shown in diagram 2.32. Click **Yes** and proceed.

The console that was running the oracle installer waits for a response from the user. Press **Enter.**

Post-Installation Tasks For Oracle 10g Database

Open the **/home/oracle/.bashrc** using any text editor as shown in diagram 2.33 by issuing the following command.

```
[root@Rose root] kate /home/oracle/.bashrc
```

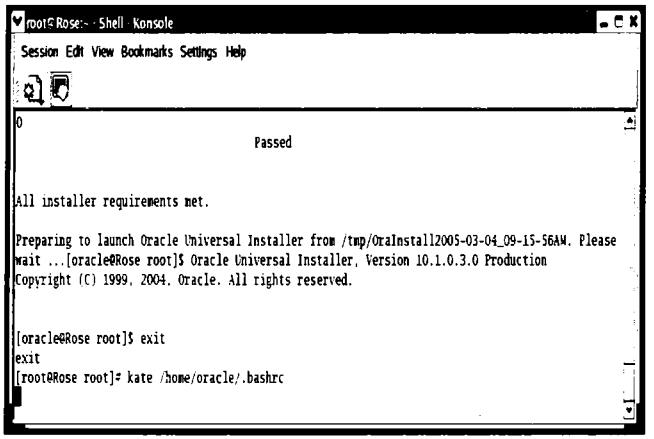


Diagram 2.33: Exiting the oracle user login & Opening the **/home/oracle/.bashrc** file.

Append the following lines - as shown in diagram 2.34 to the bottom of the file.

```
export ORACLE_HOME=$ORACLE_BASE/product/10.1.0/Db_1
export PATH=$PATH:$ORACLE_HOME/bin
```

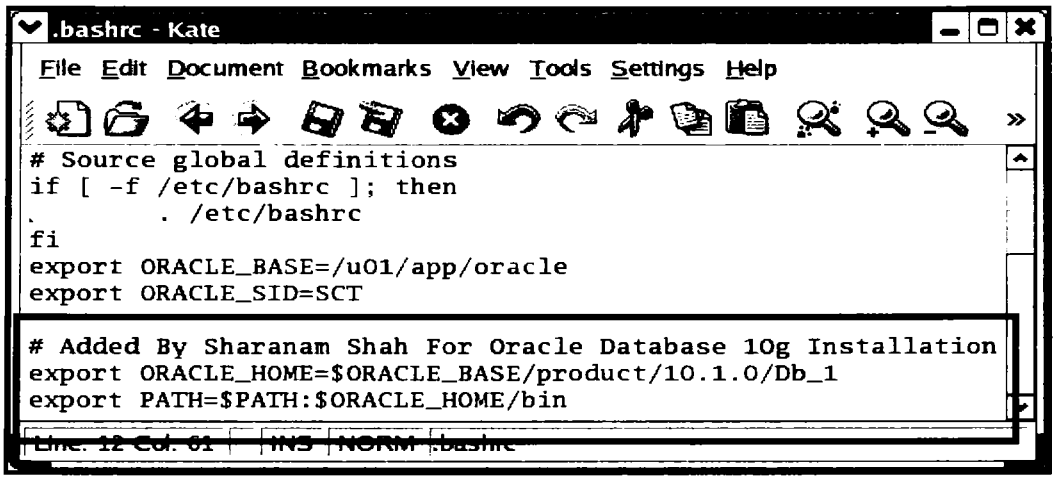


Diagram 2.34: Appending variables to .bashrc file

Testing The Installation

Logout from the console window using exit command and re-login as oracle to apply the changes made in the **.bashrc** file. After the logging in as oracle (i.e. switching the user to **oracle** from **root** in the console window), type in the command **sqlplus** to begin an interactive session with the oracle engine as shown in diagram 2.35

Use **scott** as the username and key in the appropriate password (tiger is default), a connection to the Oracle 10g Database engine is established.

To fire an SQL query, type in the following query at the SQL prompt: (Refer diagram 2.35)

SQL> SELECT * FROM tab;
SQL> SELECT * FROM dept;

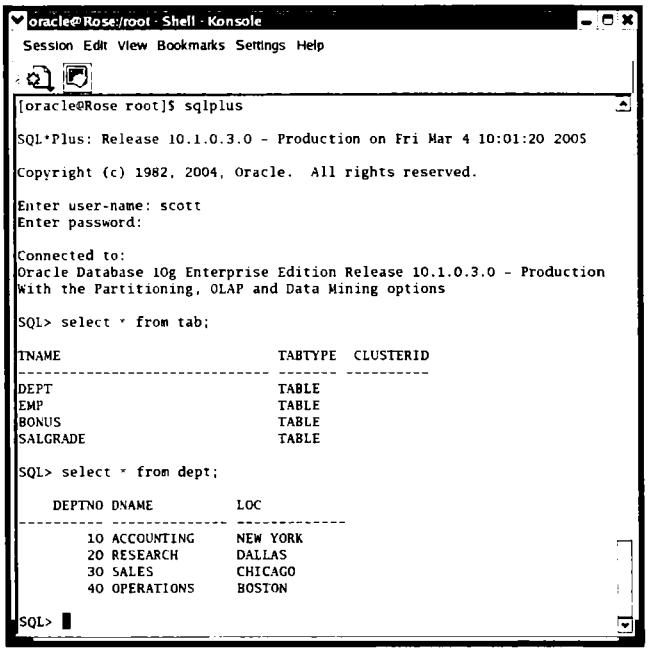


Diagram 2.35: Starting SQL *PLUS

The output appears on the screen indicating the successful installation of Oracle Database 10g as shown in diagram 2.35.

SECTION: SETTING UP THE FRAMEWORK

Setting Up Apache

The Birth of APACHE

When Rob McCool, who had developed NCSA HTTPd, left NCSA in 1994, the project fizzled out. Since the source code was publicly available, many people using it had developed their own bug fixes and additional features that they needed for their own sites. The patches were shared via Usenet, without any centralized place for collecting and distributing these patches.

Brian Behlendorf and Cliff Skolnick had put up a mailing list and Brain set up a CVS (Concurrent Versioning System) tree, now anyone who wanted to could contribute new features and bug fixes. This led to place where a group of developers could collect and distribute the patches and bug fixes. Thus came into existence - Apache.

An Introduction To APACHE

Since it was a **patchy** Web server, the name APACHE came from **A PA**t**CH**y s**E**rver. The Apache version 0.6.2 was released in April 1995. Currently there are two versions of Apache available the first is 1.3.X.XX-X the most popular, tried and tested and completely stable. Apache.org, the Apache website indicates that there will be no enhancements made to 1.3.X.XX-X on bug fixes done. The second is 2.0.X.XX, which is the latest version of Apache, which is being constantly being **bug fixed** and **enhanced** (when creating this material). Either version can be use as a full-fledged production Web Server when required.

Apache is the hottest Web server for the Internet. Apache is the world's largest used Web server. Apache is a freely available Web server. It can be downloaded from http://www.apache.org.

Getting Started

Download Apache2

Download Apache2 from **http://httpd.apache.org/download.cgi**. In the **Downloads** section choose the **latest stable version** of Apache available for download. At the time of creating this material the latest stable version of Apache was **httpd-2.0.53.tar.gz**. (Refer diagram 3.1.0)

HINT

Apache 2.0.53 Is The Best Available Version.

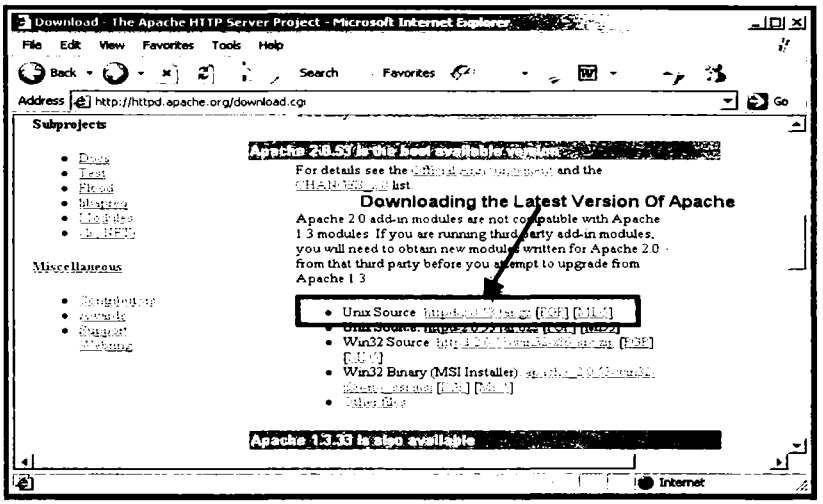


Diagram 3.1.0: Download Apache 2.

Apache 2.0 add-in modules are **not compatible** with Apache 1.3 modules. If third party add-in modules are currently being run using Apache 1.3, obtain new modules written for Apache 2.0 from that third party before attempting to upgrade from Apache 1.3 to Apache 2.0.

- Unix Source: httpd-2.0.53.tar.gz [PGP] [MD5]
- Unix Source: httpd-2.0.53.tar.bz2 [PGP] [MD5]
- Win32 Source: httpd-2.0.53-win32-src.zip [PGP] [MD5]
- Win32 Binary (MSI Installer): apache_2.0.53-win32-x86-no_ssl.msi [PGP] [MD5]
- Other files

The Apache 2 Installation Process

Erasing An Older Version

Check if any older version of apache exists. This can be done as follows:

<System Prompt> rpm –q httpd ↵

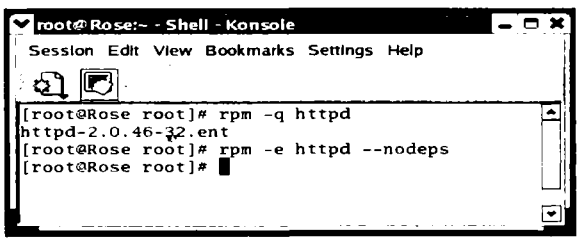


Diagram 3.1.1: Erasing the older apache version.

If an older version exists, erase it as follows: (Refer diagram 3.1.1)

<System Prompt> rpm –e httpd --nodeps ↵

HINT

There will be many dependencies reported. Erase it using **--nodeps**.

Installing The New Version

Login as **root**. (Refer diagram 3.1.2) Make a directory **/apache2**. Copy the downloaded file **httpd-2.0.53.tar.gz** there. (In this case, the setup files are on the accompanying CD-ROM, hence the CD-ROM is mounted to copy the file from). Once done, its time to unzip the file and extract its contents.

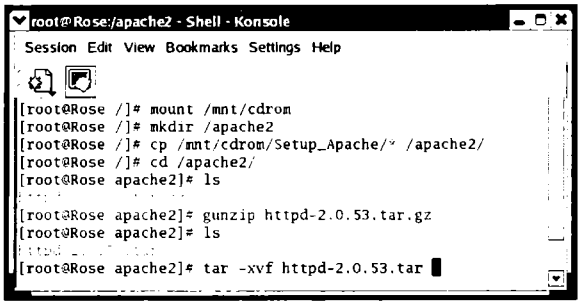


Diagram 3.1.2: Beginning the install process.

<System Prompt> gunzip httpd-2.0.53.tar.gz ↵

This will extract the file **httpd-2.0.53.tar** from the file **httpd-2.0.53.tar.gz**.

The **httpd-2.0.53.tar.gz** file gets replaced by **httpd-2.0.53.tar** in the same directory. The contents of **httpd-2.0.53.tar** must be extracted after which the actual install process can begin. To extract the content of **httpd-2.0.53.tar:** (Refer diagram 3.1.2)

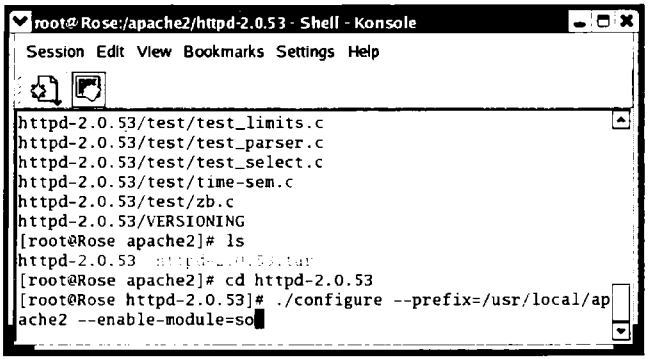


Diagram 3.2: The **./configure** command.

<System Prompt> tar -xvf httpd-2.0.53.tar ↵

The extraction process creates a directory called **httpd-2.0.53** into which the contents of the tar file are extracted. This is just one level below the **/apache2** subdirectory. Change to this sub-directory: (Refer diagram 3.2)

<System Prompt> cd httpd-2.0.53 ↵

To Begin The Apache2 Configuration

<System Prompt> **./**configure **--**prefix=<PREFIX>

REMINDER

Replace **<PREFIX>** with the file system path under which Apache should be installed. A typical installation might use **/usr/local/apache2** (*without the quotes*) for **<PREFIX>**.

The easiest way to find all of the configuration flags for Apache 2.0 is to run:
<System Prompt> **./**configure --help

Example: (Refer Diagram 3.2)

```
<System Prompt> ./configure --prefix=/usr/local/apache2 --enable-module=so ↵
```

In the above command:

1. **--prefix=/usr/local/apache2** line informs the **configure** command where Apache will be installed during the command **make install**
2. **--enable-module=so** line enables **Dynamic Module Loading** i.e. allows PHP (which will be installed later) to be compiled as a Dynamic Shared Object (DSO)

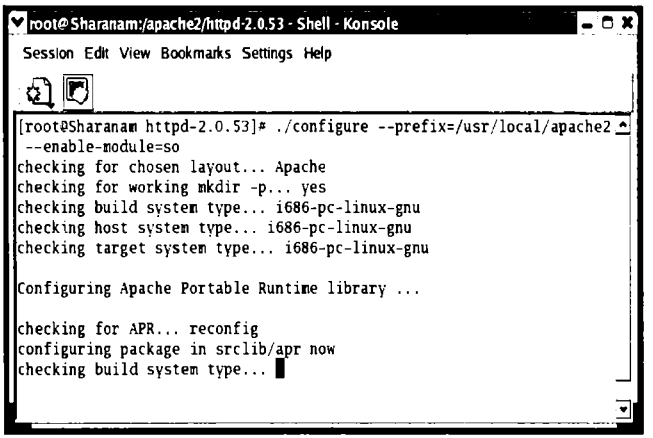


Diagram 3.3: The **./configure** processing.

The configuration routine commences (Refer diagram 3.3). The time taken depends upon the amount of memory available and the processor speed.

After the configuration runs successfully: (Refer diagram 3.4)

```
<System Prompt> make ↵
```

Finally run the: (Refer diagram 3.5)

```
<System Prompt> make install ↵
```

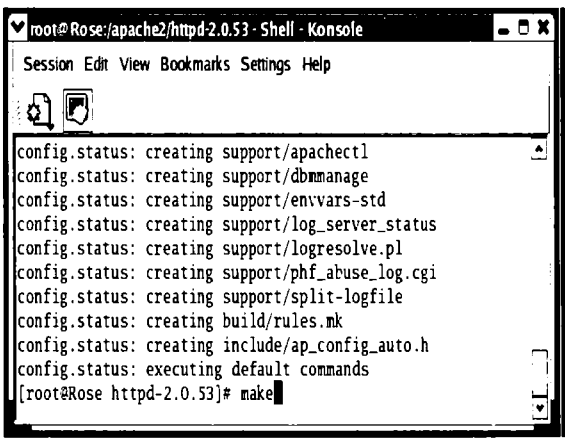


Diagram 3.4: The **make** command.

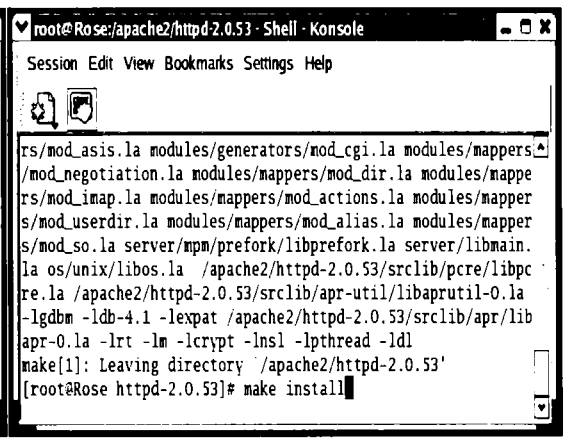


Diagram 3.5: The **make install** command.

This concludes the actual installation process on the computer's Hard Disk Drive. Apache now needs to be configured so that it knows each and every virtual domain created under it.

Testing Apache2

Once the installation is complete test it immediately. This can be done by starting the apache service as: (Refer Diagram 3.6)

<System Prompt> /usr/local/apache2 /bin/apachectl start

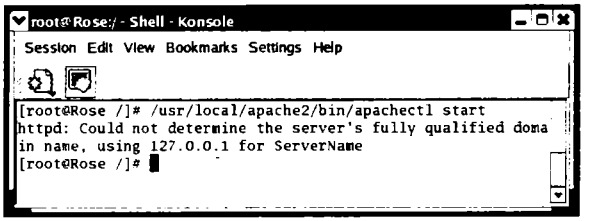


Diagram 3.6: Starting the Apache service

This is followed by pointing a web browser to the url http://127.0.0.1. If a screen as seen in diagram 3.7 is displayed then the installation was successful.

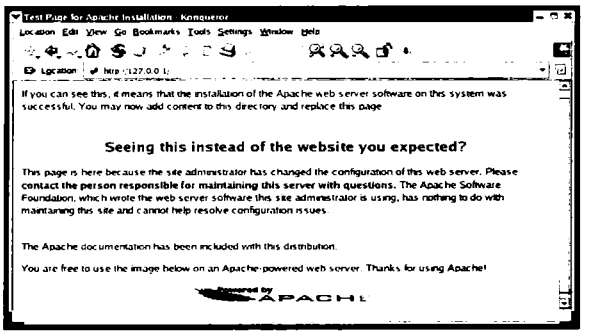


Diagram 3.7: Apache Testing

WARNING

When Apache2 is started for the very first time a warning as seen in diagram 3.6 appears. This is because the server name is not yet set in the apache configuration file (i.e. httpd.conf).

HINT

The server name will be set later in this chapter.

DIRECTORY TREE STRUCTURE OF APACHE

Once Apache is installed, its installation process creates several sub-directories under its root directory.

The Apache home directory created by the install process was **/usr/local/apache2,** therefore all Apache executables are placed here.

The sub-directories are as seen in diagram 3.8.

Once the Apache executables are installed and its directory tree structure has been created and populated through the install process, it is necessary to configure Apache to work on the computer on which it is installed. Specific entries must be made in several Apache configuration files to make Apache run successfully. Appropriate entries should be made in its **httpd.conf**, **access.conf**, **srm.conf** (i.e. **conf**-iguration) files.

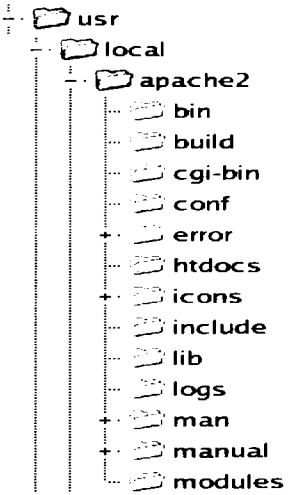


Diagram 3.8: Directory Structure.

REMINDER

In previous versions of Apache for example Apache 1.3.3 entries have to be made in **all the three** configuration files i.e. **httpd.conf**, **access.conf** and **srm.conf**. But later versions, for example Apache 1.3.6 onwards, entries have to be made in only **one** configuration file i.e. **http.conf**. This file includes the data of all the other files i.e. **access.conf** and **srm.conf**.

All these files are available in the **conf** sub-directory located under **/usr/local/apache2**

APACHE MODULES

Apache modules are code segments that are written to work with Apache because they are written using Apache's API specifications. They can be loaded into the Apache Web server executable by being added using appropriate instructions when compiling the Apache source files on Linux.

The tool called **apxs** (**AP**ache e**X**ten**s**ion) tool, allows the building and installation of an Apache extension module during its source compile time.

Alternatively, Apache modules can be dynamically loaded via Apache's httpd.conf configuration file, also known as **DSO** (**D**ynamic **S**hared **O**bject) loading, which is a fine feature of Apache.

DSO allows the extension of the features and capabilities of Apache by adding specific modules **on demand**, **without** recompiling Apache's binary executable. Use the **LoadModuie directive** placed within Apache's httpd.conf file to load a Dynamic Shared Object into Apache Web Server namespace at startup time.

For all this to work properly the O/s platform on which Apache is installed has to support DSO features and the Apache **httpd** binary has to be built (**i.e.** compiled from source) along with the **mod_so** module.

CONFIGURING APACHE WEB SERVER

When learning web page creation very often Apache server is loaded on the very same Linux box on which the web pages are designed. This facilitates testing of newly created pages. It also enables the web page creators to physically check to see what the web pages would look like and behave before being mounted on an Internet based server.

With this in mind, the following must be done on the computer.

Locate the **Hosts** file, available under **/etc**. This file is a pure ASCII file. This file will be used to hold the **IP address** and **name** of the local machine.

Open the *Hosts* file in **Kate** (i.e. a Linux based ASCII editor) and make the following entries: (Refer to diagram 3.9)

ip. Address Hostname
(A **Tab** separates these two entries)
127.0.0.1 www.ivanbayross.com
(i.e. any domain name of choice)

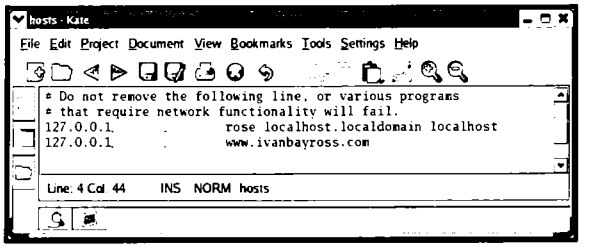


Diagram 3.9: Contents of **/etc/hosts** file.

From now on, if Apache is passed the IP address 127.0.0.1, (the name **localhost)** or **www.ivanbayross.com** Apache will understand that a reference is being made to the same computer on which it is installed, which has a virtual domain named **www.ivanbayross.com** hosted on it.

SETTINGS TO BE MADE IN THE HTTPD.CONF FILE

The main configuration file of Apache is **httpd.conf**, which contains directives written in plain text. The location of this file is set at compile-time. When required, directives within this file, may be overridden with the **–f** command line flag.

The **srm.conf** and **access.conf** files can also be used to configure Apache. Other configuration files may be added and their instructions passed to Apache by using the **Include** directive followed by the **path** that points to where the additional configuration files are.

Any directive may be placed in **any** of these configuration files. Apache is pretty flexible in this respect. Changes made to the main configuration files are only recognized by Apache when it is **started** or **restarted**. If any configuration file is actually a directory, Apache will enter that directory and parse any files (and sub-directories) found there as configuration files.

Apache **works best** if there is **only one** configuration file used and all its directives are placed in that file (**i.e. httpd.conf**). Apache also reads a file containing document mime types. This filename is set by the **TypesConfig** directive, which is **mime.types** by default.

Apache configuration files contain one directive per line. If a directive must continue onto the next line use back-slash (\) as the **last character** on the previous line.

Directives in configuration files are **case-insensitive**, but **arguments** to directives are **often case-sensitive**. Any line beginning with a hash (#) character is ignored. Blank lines and white spaces before a directive are ignored. Configuration files can be checked for syntax errors **without** starting the server by using **apachect1 configtest** or the **–t** command line option.

Understanding Some Important Entries In httpd.conf File

Global Settings

ServerType standalone
The two server types are **standalone** and **inetd**. Usually it is always **standalone**. Setting it to **inetd** causes a new server to be started to handle every incoming HTTP request, which will die as soon as the request is served. This makes things slower because of re-reading the configuration file and the overhead of server startup with every request.

ServerRoot /etc/httpd
This sets the absolute path to the server directory. Generally, the argument of **ServerRoot** should be the path to where Apache is installed.

ErrorLog logs/error_log
CustomLog logs/access_log common
The various directives that end in **Log control** indicate whether log files exist at all. These directives also indicate exactly where the log files exist in the Linux file system.

LockFile /var/run/httpd.lock
The LockFile directive sets the path to the lockfile used when Apache is compiled. This directive should normally be left at its default value.

PidFile /var/run/httpd.pid
It is a file in which the server should record its process identification number when it starts.

ScoreBoardFile logs/apache_runtime_status
It is a file used to store internal server process information. Not all architectures require this. But if local architecture does (This will be known because this file will be created when Apache is run) then **ensure** that no two invocations of Apache share the same Scoreboard file.

Timeout 300
Indicates the number of seconds before Apache receives and sends a time out.

KeepAlive On
Indicates whether or not to allow persistent connections (more than one request per connection). Set it to **Off** to deactivate.

MaxKeepAliveRequests 100
Indicates the maximum number of requests to allow during a persistent connection. Set it to **0** to allow an unlimited amount. It is recommended that this number is set high, for maximum performance.

KeepAliveTimeout 15
Indicates the number of seconds to wait for the next request from the same client on the same connection.

MinSpareServers 5
MaxSpareServers 20
Indicates the Server-pool size regulation. Rather than making a user guess how many server processes are required, Apache dynamically adapts to the load it sees — that is, it tries to maintain enough server processes to handle the current load, plus a few spare servers to handle transient load spikes (e.g., multiple simultaneous requests from a single Netscape browser).

It does this by periodically checking how many servers are waiting for a request. If there are fewer than MinSpareServers, it creates a new spare. If there are more than MaxSpareServers, some of the spares die off. The default values are probably OK for most sites.

Number of servers to start initially should be a reasonable ballpark figure.

StartServers 8
Port 80
This indicates the port on which the server should run on.

User and Group
The Web server's user and group values are denoted by the User and Group settings. This should be set to User ID and Group ID that the server will use to process requests. If the server runs as root, some hacker could exploit the privilege. Normally, people want to run Apache as an underprivileged user for security reasons. On Linux, this can be done by setting both to **nobody**.

ServerAdmin root@localhost
Accepts the Email address, where problems with the server should be e-mailed. This address appears on some server-generated pages, such as error documents.

ServerName localhost
This sets the hostname the server will return. Set the name of the server using the **ServerName** directive. This is especially useful when the computer has multiple names **or** IP addresses.

DocumentRoot /usr/local/apache2/htdocs
This indicates the absolute path of the document tree, which is the top directory from which Apache will serve files. The **DocumentRoot** is the root of the Web tree and it defaults to **/usr/local/apache2/htdocs**. Assuming that Apache is installed under **/usr/local/apache2/**, this can be changed if required.

DirectoryIndex index.html index.htm index.shtml index.php index.php4 index.php3 index.phtml index.cgi

Specifies the name of the file or files to use as a pre-written HTML. Separate multiple entries with **spaces**.

UserDir public_html

Indicates the name of the directory that is appended onto a user's home directory if a user request is received. This is directly relative to a local user's home directory where that user can put public HTML documents. Documents placed in that directory will be available as: http://servemame/~username

Alias /icons/ "/usr/local/apache2/icons/"
ScriptAlias /cgi-bin/ "/var/www/sct/phpproj/cgi-bin/"

These both map a URL path to a directory on the server.

Additional options can be added to directories using **<Directory>** and **Options** directives. <Directory> is a **container directive**. Container directives enclose multiple lines in the **config** file. They require a closing directive of the form </Directive>.

Aspects of Apache's behavior can be controlled on a per-directory or per-filename basis using <Directory> and related **<Files>** and **<FilesMatch>** directives. Directives placed between <Directory></Directory> apply to **sub-directories** as well. What options are allowed in a directory can be controlled using the **Options** directive. Possible values to an **Options directive** are given below with brief descriptions.

Option	Brief Explanation
ExecCGI	CGI scripts can be run from this directory tree
FollowSymLinks	The server will follow symbolic links in this directory
Includes	Server-side includes are permitted
IncludesNoEXEC	Server-side includes are permitted, but the #exec command and #include of CGI scripts are disabled
Indexes	If a URL, which maps to a directory is requested, and there is no **DirectoryIndex** (e.g, index.html) in that directory, then the server will return a formatted listing of the directory
MultiViews	Content negotiated **MultiViews** are allowed
SymLinksIfOwnerMatch	The server will only follow symbolic links for which the target file or directory is owned by the same user id as the link
All	Everything except for **MultiViews**. This is the default value

Apache allows for decentralized management of configuration via special files placed inside the web tree. The special files are usually called **.htaccess**, but **any name** can be specified in the **AccessFileName directive.**

Directives placed in **.htaccess** files apply to the directory where the file is placed and all sub-directories. .htaccess files follow the same syntax as main configuration files. Since .htaccess files are read on every request, **changes** made in these files **take immediate effect**. The directives that may be placed in .htaccess files can be controlled by configuring the **AllowOverride directive** in the main configuration file **httpd.conf**.

REMINDER

The **default settings** provided by Apache2 installation were maintained. No changes are made in this file content.

VIRTUAL HOSTS

The **VirtualHost** directive in the **httpd.conf** configuration file is used to set the values of **ServerName**, **DocumentRoot**, **ErrorLog** and **TransferLog** or **CustomLog** configuration directives to hold different values for **each** virtual host.

Multiple websites can be served from one computer, as long as they have different domain names. Each domain, served from the single computer that hosts them all, is referred to as a **virtual host**. There are **two** ways provided by Apache for setting up virtual hosts on a single computer, **IP based** and **Name based**.

IP based virtual hosts use the IP address of the connection to determine the correct virtual host. Hence, a **unique** IP address is required for each host.

With **name based** virtual hosting, Apache Web server relies on the client to deliver the hostname as part of the HTTP leaders sent to Apache. Using this technique, many different virtual hosts can share the **same IP address**.

WARNING

Older browsers do not support delivering a hostname with their HTTP headers. This is not part of their **HTTP 1.1** header encoding. Hence these browsers will only work with IP based virtual hosts.

Apache can be configured to support multiple virtual hosts either by running a **separate httpd** daemon for each hostname or by running a **single httpd** daemon, which supports all the virtual hosts.

If separate httpd daemons must be run for each host, separate installations of Apache for each virtual host have to be created. For each installation, use the **Listen directive** in the **httpd.conf** configuration file to select which IP address **or** virtual host that daemon services. For example, Listen 192.168.0.1:80.

A single http daemon can also be used to service to the main server and **all** its virtual hosts.

Configuring Name Based Virtual Hosts

Using the new name based virtual hosts is quite easy and superficially looks like the old method. The notable difference between IP-based and Name-based virtual host configuration is the **NameVirtualHost** directive, which specifies a single IP address that should be used as a target for name-based virtual hosts.

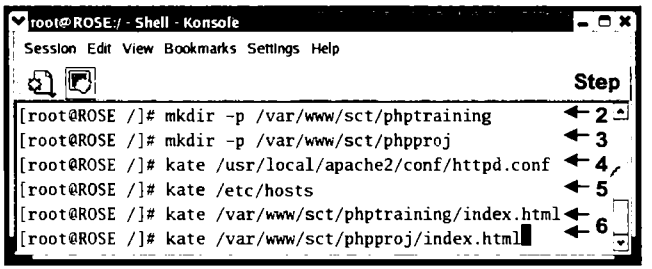


Diagram 3.10.1: Steps to create Virtual Hosts

For example, suppose that both domains sct.phptraining.com and sct.phpproj.com must be bound to the IP address 192.168.0.3. Then simply follow the following steps: (Refer diagram 3.10.1)

1. Create a directory called **sct** under **/var/www/**
2. Create a directory called **phptraining** under **/var/www/sct**
3. Create a directory called **phpproj** under **/var/www/sct**
4. Edit the **/usr/local/apache2/conf/httpd.conf** file and append the following towards the end of the file: (Refer diagram 3.10.2)

```
ServerName 192.168.0.3

NameVirtualHost 192.168.0.3
<VirtualHost 192.168.0.3>
   DocumentRoot
/var/www/sct/phptraining
   ServerName
sct.phptraining.com
</VirtualHost>

<VirtualHost 192.168.0.3>
   DocumentRoot
/var/www/sct/phpproj
   ServerName sct.phpproj.com
</VirtualHost>
```

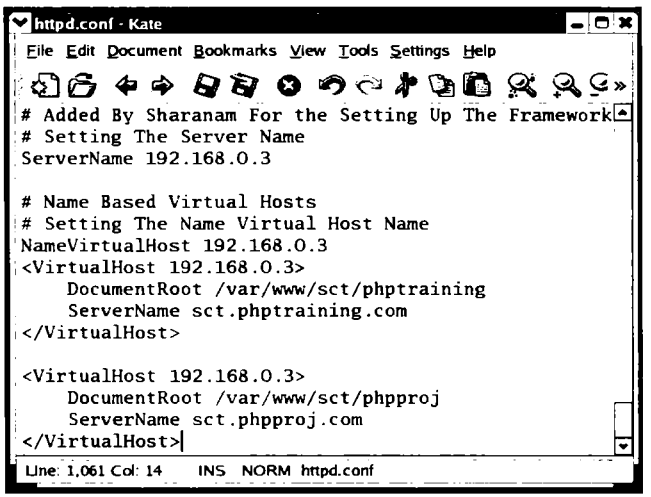


Diagram 3.10.2: The httpd.conf file with name based virtual hosting.

5. Edit the **/etc/hosts** file and append the following entries: (Refer diagram 3.10.3)

```
192.168.0.3   sct.phptraining.com
192.168.0.3   sct.phpproj.com
```

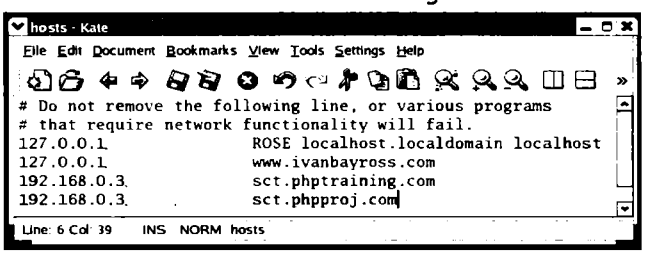


Diagram 3.10.3: The **/etc/hosts** file for name based virtual hosting.

6. To test the name based virtual host settings create an index file in the root of each host as: (Refer diagram 3.10.4 and diagram 3.10.5)

index.html at /var/www/sct/phptraining/

```
<HTML>
  <HEAD><TITLE>SCT PHP Training</TITLE></HEAD>
  <BODY>This is a Training Domain</BODY>
</HTML>
```

index.html at /var/www/sct/phpproj/

```
<HTML>
  <HEAD><TITLE>SCT PHP Project</TITLE></HEAD>
  <BODY>This is a PHP Project</BODY>
</HTML>
```

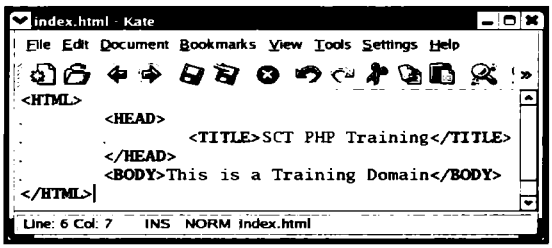


Diagram 3.10.4: index.html file under **phptraining**

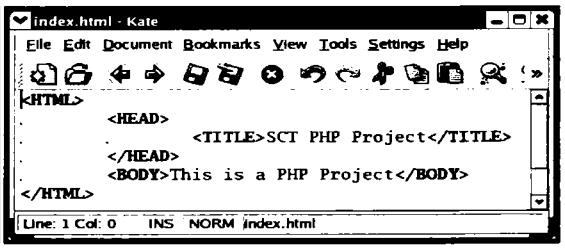


Diagram 3.10.5: index.html file under **phpproj**

7. Point the browser to both the URLs, i.e. one at a time as seen in diagram 3.11.1 and diagram 3.11.2

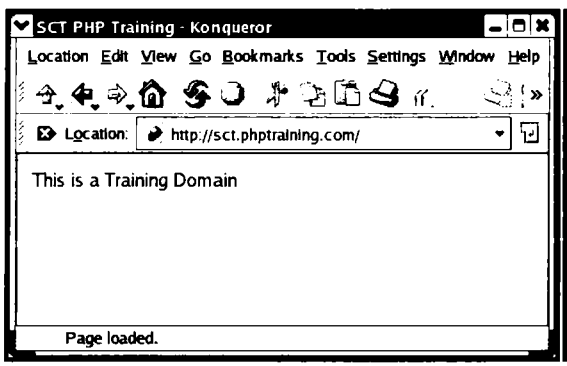


Diagram 3.11.1: Testing **sct.phptraining.com**

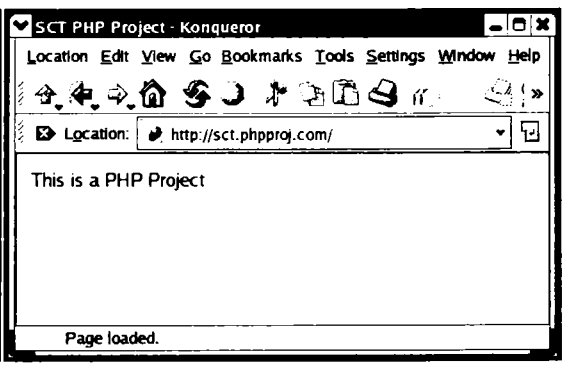


Diagram 3.11.2: Testing **sct.phpproj.com**

httpd.conf File For The Framework

The following entries should exist in the **httpd.conf** file, found in, **/usr/local/apache2/conf** directory:

```
ServerName 192.168.0.3
(Here enter the actual IP of the host computer this IP is the one
we used)

NameVirtualHost 192.168.0.3

NameVirtualHost 192.168.0.3
<VirtualHost 192.168.0.3>
  DocumentRoot /var/www/sct/phptraining
  ServerName sct.phptraining.com
</VirtualHost>

<VirtualHost 192.168.0.3>
  DocumentRoot /var/www/sct/phpproj
  ServerName sct.phpproj.com
</VirtualHost>
```

The hosts File For The Framework

The Following entry should be appended to the **hosts** file, found in, /etc directory:

```
192.168.0.3   sct.phptraining.com
192.168.0.3   sct.phpproj.com
```

Registering The Changes Made In The httpd.conf With Apache2

After making any changes to the **httpd.conf** ensure to restart the apache server to apply the new changes. This can be done by:

- Stop the Apache2 HTTPD service:

```
<System Prompt> /usr/local/apache2/bin/apachectl stop ↵
```

- Start the Apache2 HTTPD service:

```
<System Prompt> /usr/local/apache2/bin/apachectl start ↵
```

Now since the server name has been set in the **httpd.conf** file apache will restart smoothly without any warning. (Refer diagram 3.6 and diagram 3.12)

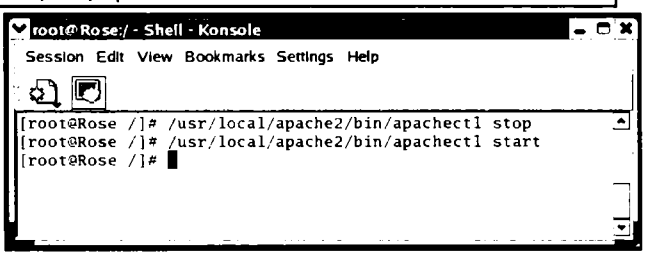


Diagram 3.12: Apache Restart

Self Review Questions

FILL IN THE BLANKS

1. Configuration files can be checked for syntax errors without starting the server by using _____ or the _____ command line option.
2. The _____ and _____ directives map a URL path to a directory on the server.
3. <Directory> is a _____ directive.
4. Apache modules are code segments are written using _____ specifications.
5. The _____ and _____ directives indicate the Server-pool size regulation.
6. The _____ indicates the absolute path of the document tree.
7. The _____ directive indicates the number of seconds to wait for the next request from the same client on the same connection.
8. The _____ tool allows the building and installation of an Apache extension module during source compile time.

TRUE OR FALSE

9. Apache and either MySQL or Postgre SQL are from the open source domain.
10. The latest version of Apache is 1.3.X.XX-X.
11. Apache 2.0 add-in modules are compatible with Apache 1.3 modules.
12. The various directives that end in Log control indicate whether log files exist at all.
13. Apache disallows decentralized management of configuration via special files placed inside the web tree.
14. Multiple websites can be served from one computer, even if they have different hostnames.
15. A unique IP address is required for each host.
16. The ServerAdmin directive sets the hostname the server will return.
17. The ServerName accepts the Email address, where problems with the server should be e-mailed.
18. A Web server (*Apache2*) cannot communicate directly with a database management system (*MySQL*).

SECTION I: SETTING UP THE FRAMEWORK

Setting Up PHP

This material is designed to guide through the initial steps of setting up PHP on Linux. The Linux distribution being used for this material is Red Hat Linux Advanced Server 3, however the steps should be very similar across most distributions. This material makes the assumption that the required development tools are already loaded for compiling programs from source. Also, it assumes that the Oracle Database and the Apache Web Server are installed and ready to use.

Apache, Oracle and PHP have become one of the most utilized combinations for developing content driven websites. They are robust, flexible, provide a decent level of security, and they are available for many different platforms.

Here is a quick set of steps that will build PHP as a dynamic Apache module (DSO) **with Oracle support**.

Quick Install Of PHP As A Dynamic Shared Object (DSO)

For this to work the Apache httpd (**i.e.** apachectl) must have mod_so enabled.

Check using:

```
<System Prompt> /usr/local/apache2/bin/apachectl -l ↵
(i.e. hyphen lower case L)
```

The output visible on the VDU should be something like:

Compiled-in modules:

```
http_core.c
mod_so.c
```

The chances are there will be a lot more modules than these two. That's OK, as long as **mod_so.c** is present one can proceed with the following steps:

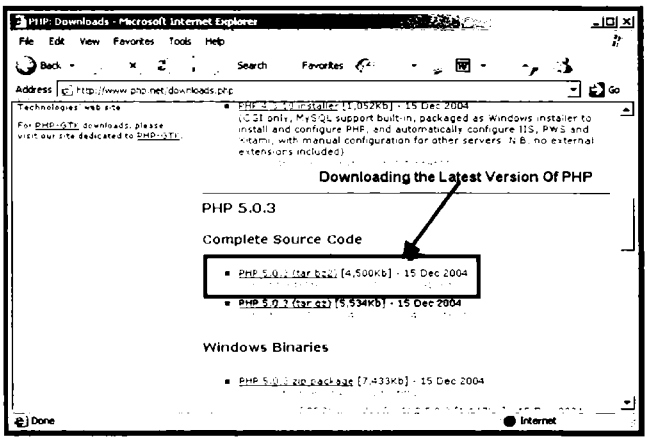


Diagram 4.1: Downloading PHP

Download PHP

Download the **latest stable version** of PHP from http://www.php.net/downloads.php.

Latest Stable Version Of PHP Complete Source Code are: (Refer diagram 4.1)

- PHP 5.0.3 (tar.bz2) [4,500Kb] - 15 Dec 2004
  md5: fd26455febdddee0977ce226b9108d9c
- PHP 5.0.3 (tar.gz) [5,534Kb] - 15 Dec 2004
  md5: bf89557056ce34d502e20e24071616c7

The file chosen for download was **php-5.0.3.tar.bz2**.

Starting The Install Process

Login as root

Erasing An Older Version

Check if any older version of PHP exists. This can be done as follows: (Refer diagram 4.2.1)

```
<System Prompt> rpm -q php ↵
```

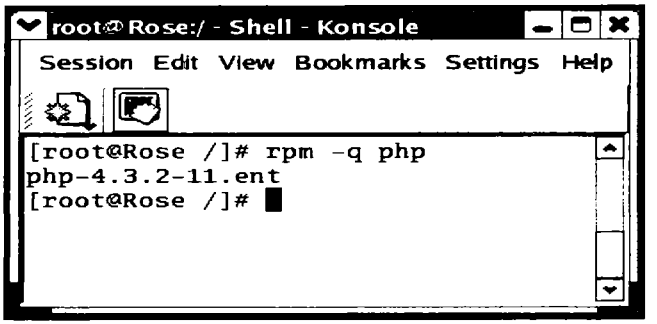


Diagram 4.2.1: Verifying the existence of older PHP version.

If an older version exists, erase it as follows: (Refer diagram 4.2.2)

```
<System Prompt> rpm -e php ↵
```

If any dependencies are reported then erase PHP as follows: (Refer diagram 4.2.2)

```
<System Prompt> rpm -e php
                    --nodeps ↵
i.e. two hyphens followed
by nodeps)
```

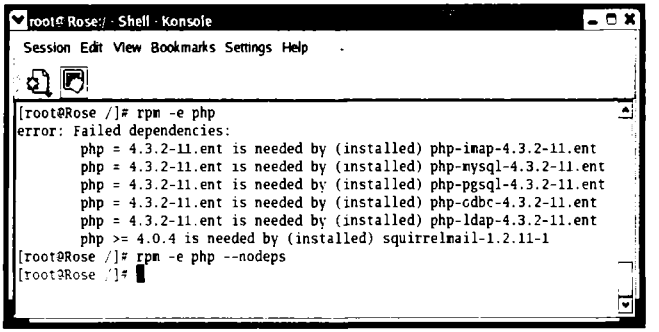


Diagram 4.2.2: Erasing the older PHP version.

Install The New Version

Make a directory **/php**. (Refer diagram 4.3)

Copy the downloaded file **php-5.0.3.tar.bz2** here. (Refer diagram 4.3)

Unzip the file as: (Refer diagram 4.3)

```
<System Prompt> bunzip2
                php-5.0.3.tar.bz2 ↵
```

This will extract **php-5.0.3.tar** file from the **php-5.0.3.tar.bz2** file.

The **php-5.0.3.tar.bz2** will be replaced by the **php-5.0.3.tar** in the same sub directory.

The contents of php-5.0.3.tar must be extracted after which the actual install process can begin. To extract the content of php-5.0.3.tar: (Refer diagram 4.4)

```
<System Prompt> tar -xvf
                php-5.0.3.tar ↵
```

The extraction process creates a directory called **php-5.0.3** into which the contents of the tar file are extracted. This is just one level below the **/php** subdirectory. Change to this sub-directory: (Refer diagram 4.5)

```
<System Prompt> cd php-5.0.3
                ↵
```

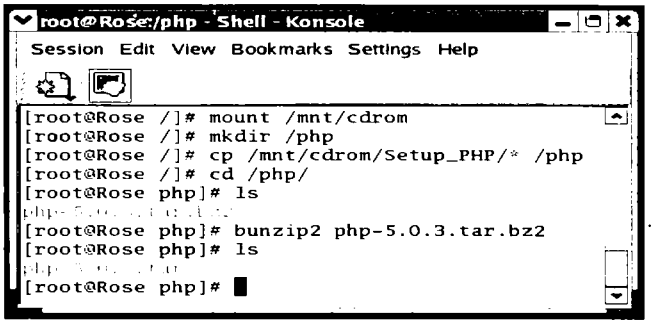


Diagram 4.3: Staring the Install process.

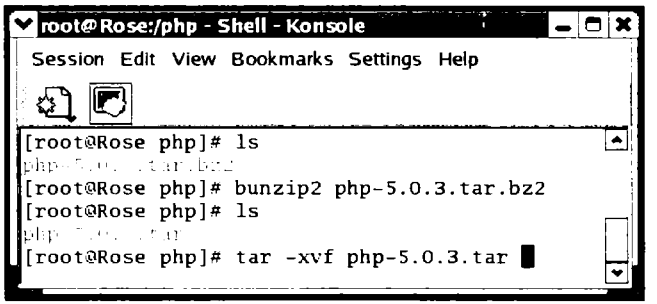


Diagram 4.4: Staring the Install process

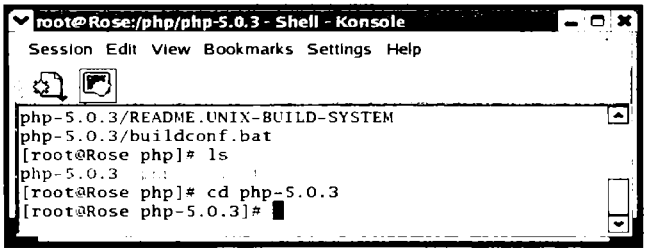


Diagram 4.5: The tar file extracted.

To Begin PHP Configuration

Make sure that apache is stopped before proceeding with PHP Installation.

Stop the Apache2 HTTPD service as: (Refer diagram 4.6)

```
<System Prompt> /usr/local/apache2/bin/apachectl stop ↵
<System Prompt> ./configure --with-oci8=/u01/app/oracle/product/10.1.0/Db_1
                --with-apxs2=/usr/local/apache2/bin/apxs --enable-sigchild ↵
```

In the above command:

1. **--with-oci8=/u01/app/oracle/product/10.1.0/Db_1** line should point to the location of the database client software i.e. **$ORACLE_HOME** directory, in Oracle terms. This option actually adds Oracle support to PHP. In short PHP will be built using **Oracle** support
2. **--with-apxs2=/usr/local/apache2/bin/apxs** line **must** point to apache web server's **apxs** script
3. **--enable-sigchild** line obviates a problem which is characterized by a large number of defunct processes existing when using the Apache/PHP/Oracle combination under Linux

Diagram 4.6: The **./configure** command.

REMINDER

If an error is generated which indicates that the apxs script cannot be found, look for it on the system (i.e. use Find Files) and if found, note down the path to the file. Then provide the full path such as: **--with-apxs2=/path-to-apxs**

Make sure to specify the **version of apxs** that is actually installed on the system and **NOT** the one that is in the apache source tarball.

If an error appears about **apxs** and the help screen from apxs is displayed, then **recompile** Apache and **ensure** that **--enable-module=so** is specified to the **configure** command.

The configuration routine commences. The time taken depends upon the amount of memory available and the processor speed. After the configuration runs successfully: (Refer diagram 4.7)

<System Prompt> make ↵

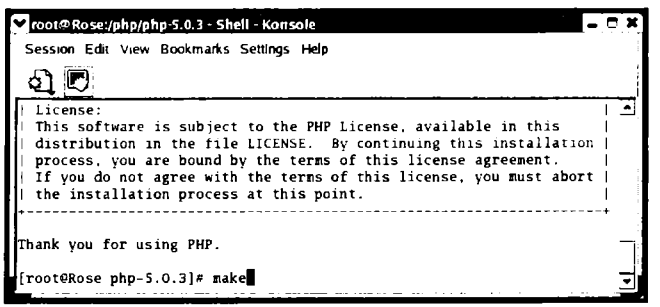


Diagram 4.7: The **make** command.

Next run the command:
(Refer diagram 4.8)

<System Prompt> make install ↵

Finally execute the command: (Refer diagram 4.9)

```
<System Prompt> cp php.ini-dist /usr/local/lib/php.ini ↵
```

This will create a local copy of the PHP configuration file.

This completes the installation of PHP.

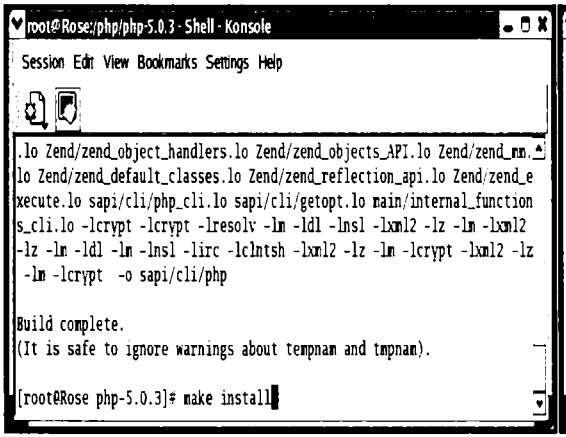


Diagram 4.8: The **make install** command.

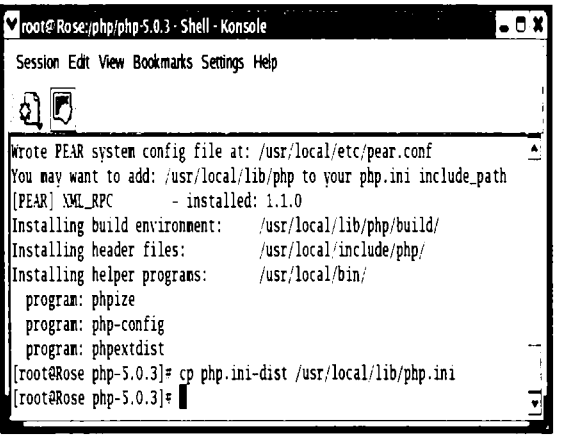


Diagram 4.9: The installation completes.

Binding The PHP Installation With Apache2

The only thing left to do is to edit Apache's **httpd.conf** file and add a handler for .php files. This handler can be appended towards the end of the file: (Refer diagram 4.10.1)

```
<IfModule mod_php5.c>
  AddType application/x-httpd-php .php
  AddType application/x-httpd-php .phtml
  AddType application/x-httpd-php-source .phps
</IfModule>
```

Ensure that the PHP module is loaded by noticing the following line in the **httpd.conf** file (Refer diagram 4.10.2)

```
LoadModule php5_module    modules/libphp5.so
```

This line loads the PHP module dynamically into Apache. The PHP installer automatically inserts this line when it was installed. If this line does not exist then insert the same in the httpd.conf file. (Refer diagram 4.10.2)

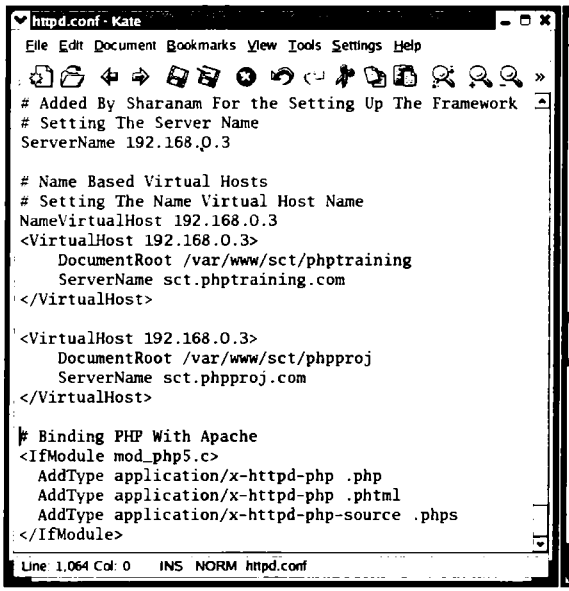


Diagram 4.10.1: Binding PHP with Apache2

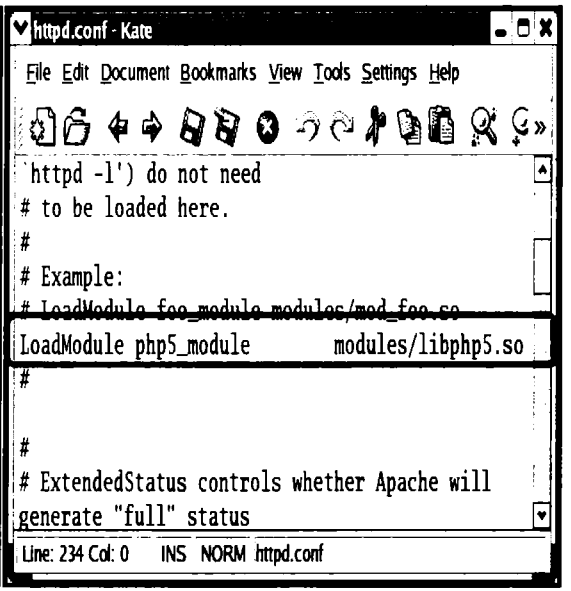


Diagram 4.10.2: Verifying the Load Module

Start Apache Server (**i.e.** apachectl start), PHP files should be able to be served up now.

Registering The Changes Made In The httpd.conf With Apache2

Restart the Apache2 HTTPD service **or** simply start it using the parameter **start:**

```
<System Prompt> /usr/local/apache2/bin/apachectl restart ↵
```

Testing the PHP Setup

To test whether PHP has been successfully setup and intergrated with Apache2 create a simple script named phpinfo.php that contains the following code: (Refer diagram 4.11)

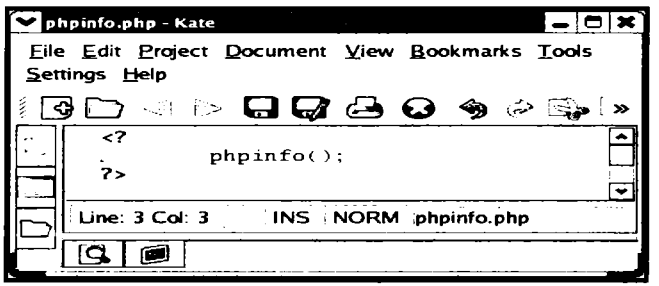


Diagram 4.11: The phpinfo.php file in kate.

```
<?
  phpinfo();
?>
```

Place this file in the Web server's *virtual domain* created earlier under **/var/www/sct/phptraining** directory. Examine the output of this script in a Web browser by pointing to **http://*sct.phptraining.com/phpinfo.php***. If PHP setup was successful then a screen similar to that shown in diagram 4.12 appears.

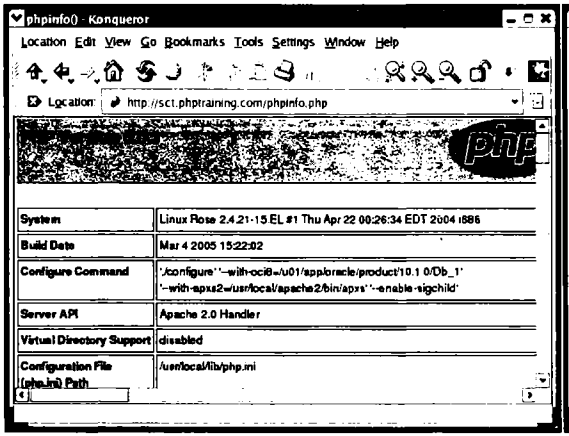


Diagram 4.12: The PHP info output

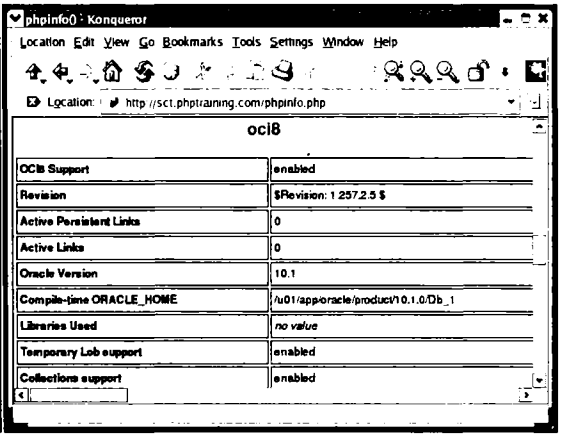


Diagram 4.13: The oracle support verification

The **oracle** support was enabled while installing php5. This can be verified by scrolling through the **phpinfo** output and noticing the oracle section as seen in diagram 4.13.

The httpd.conf File For The Framework

The following entries should exist in the **httpd.conf** file available in the **/usr/local/apache2/conf** directory:

```
ServerName 192.168.0.3
(Here specify the actual IP of the host computer. The IP address
mentioned above have been used while generating this material.)

NameVirtualHost 192.168.0.3
<VirtualHost 192.168.0.3>
   DocumentRoot /var/www/sct/phptraining
   ServerName sct.phptraining.com
</VirtualHost>

<VirtualHost 192.168.0.3>
   DocumentRoot /var/www/sct/phpproj
   ServerName sct.phpproj.com
</VirtualHost>

LoadModule php5_module        modules/libphp5.so

<IfModule mod_php5.c>
   AddType application/x-httpd-php .php
   AddType application/x-httpd-php .phtml
   AddType application/x-httpd-php-source .phps
</IfModule>
```

SECTION: SETTING UP THE FRAMEWORK

Setting Up PHP-Oracle Communication Via Apache

PHP is a server side, cross platform, scripting language for writing Internet / Intranet based commercial applications. It allows the embedding of program logic **within** HTML pages. This enables dynamic, database driven, web pages to be served from a Web server from anywhere. PHP allows the creation of Internet/Intranet based, user interfaces that interact in real time with Oracle database tables.

The Architecture On Which PHP Works

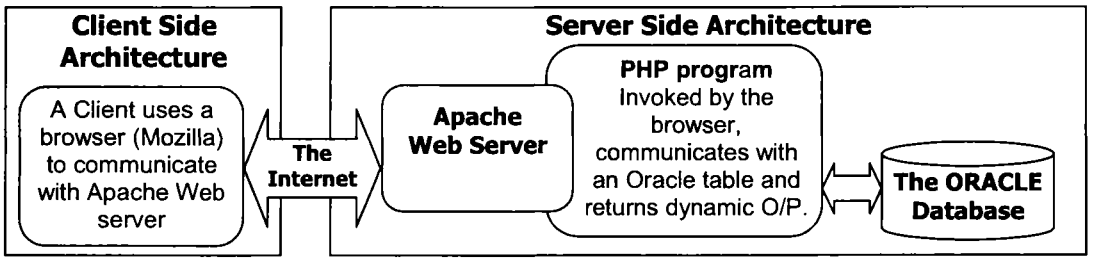


Diagram 5.1

The operational framework therefore includes:

1. **Red Hat Linux Advance Server** as the **Operating System** of choice
2. **Oracle 10g Database** as the **RDBMS** engine of choice
3. **Apache Web Server** as a **Web Server**
4. **PHP** as the scripting language (Loaded dynamically as an Apache module)

To begin working with this operational framework follow these steps:

1. **Start ORACLE** (Refer diagram 5.2)

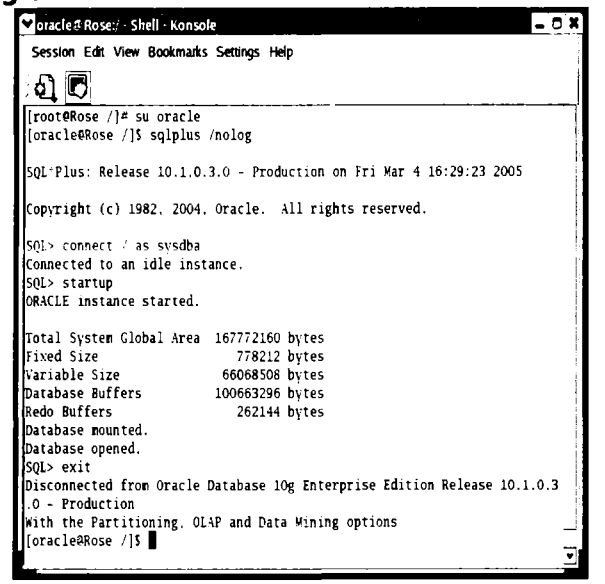


Diagram 5.2: The Oracle database starting up

The steps for starting the Oracle database engine are:

- Invoke the Linux Console window by clicking on the **Konsole** menu option in the taskbar.
- In the Console window, using the **su** command change the login to **oracle:**

```
[System Prompt]$ su oracle
```

- Invoke a session using SQL *Plus with the nolog switch:

```
[System Prompt]$ sqlplus /nolog
```

- On getting the SQL prompt, connect to the Oracle instance as **sysdba**.

```
SQL> connect / as sysdba
```

- When connected, execute the command to start the database.

```
SQL> startup
```

The startup process will take some time. The Linux console will echo each step of Oracle database engine startup progress as shown in diagram 5.2. Once the database is up, exit the SQL prompt.

2. Start the default ORACLE Listener

The steps for starting the oracle listener are as follows:

- Invoke a Linux Console by clicking on **Konsole** menu option in the taskbar.
- In the Console window, using the **su** command change the login to **oracle.**

```
[System Prompt]$ su oracle
```

- Execute the command to start the Listener service.

```
[System Prompt]$ lsnrctl start
```

The Linux console will echo each step of Oracle Listener startup progress as shown in diagrams 5.3.1 and 5.3.2.

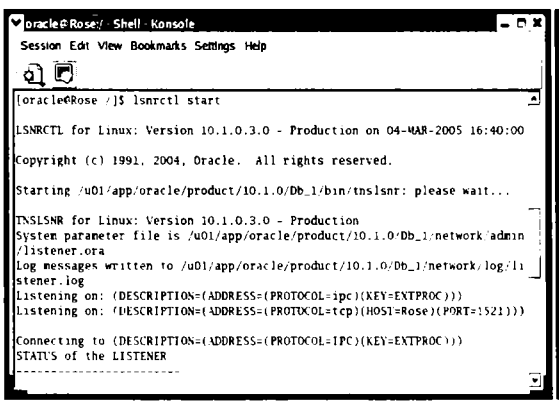


Diagram 5.3.1: Output of the **lsnrctl** command.

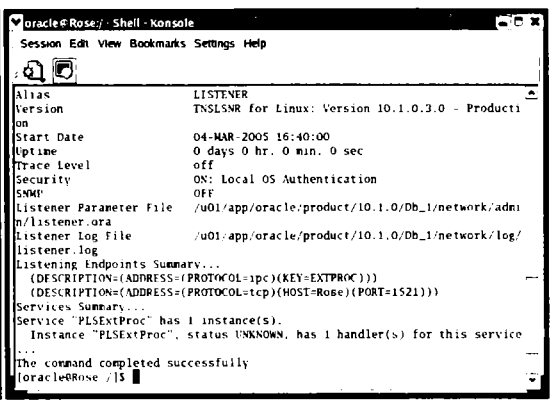


Diagram 5.3.2: The Listener started successfully.

3. **Switch back to the previous user (i.e. root) by using the exit command.** (Refer diagram 5.4)

4. **Start Apache**

When the Apache HTTP Server is started, **ensure** that the required Oracle environment variables such as ORACLE_HOME and ORACLE_SID are set properly for Oracle.

These variables were set previously for the oracle user. But apache can be started only by root and hence the same variables need to be set for the root user as well.

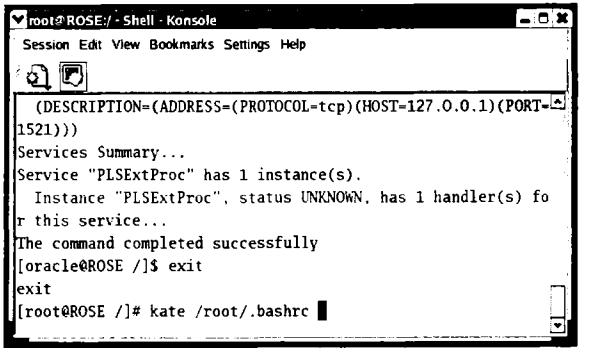


Diagram 5.4: Switching to root and editing **.bashrc** file

To simplify things, edit the file **/root/.bashrc** and **append the following lines which will set variables.** (Refer diagram 5.5)

```
ORACLE_HOME=/u01/app/oracle/product/10.1.0/Db_1
ORACLE_SID=SCT
export ORACLE_HOME
export ORACLE_SID
```

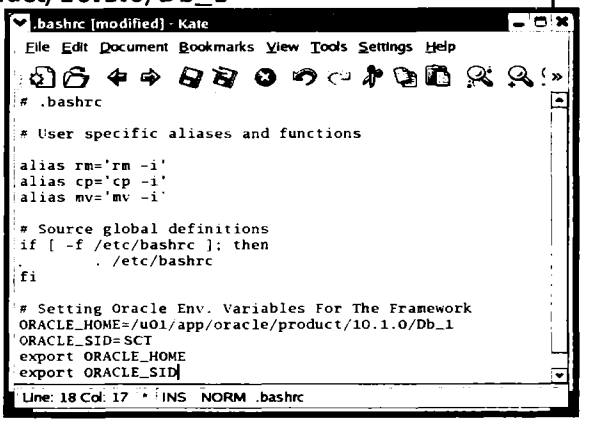


Diagram 5.5: Editing the **/root/.bashrc** file.

REMINDER

Apply these variables by exiting the current console window and then starting Apache in a different console. (*Assuming the Apache service was stopped. If it was running then restart it.*) (Refer diagram 5.5, diagram 5.6.1 and diagram 5.6.2.)

HINT

The reason to use a different console is to apply the changes made in the **.bashrc** file, which will only be applied on exiting the current console and starting a new console.

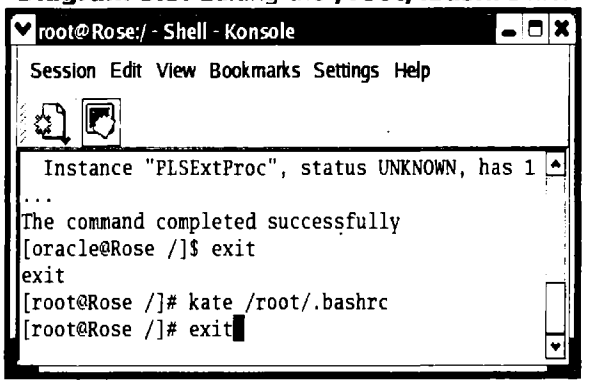


Diagram 5. 6.1: Exiting current console

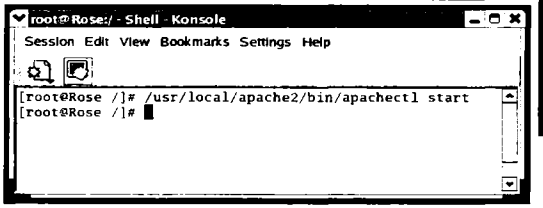


Diagram 5.6.2: Starting Apache

Testing the connectivity

Create a file named **oratest.php** under the host **sct.phptraining.com** created earlier at **/var/www/sct/phptraining** (Refer diagram 5.7 and diagram 5.8).

Contents Of oratest.php:

```
<?php
  $db_conn = ocilogon("scott", "tiger");

  $cmdstr = "SELECT ENAME, SAL FROM EMP";

  $parsed = ociparse($db_conn, $cmdstr);

  ociexecute($parsed);

  $nrows = ocifetchstatement($parsed, $results);

  echo "<HTML><HEAD>";
  echo "<TITLE>ORACLE & PHP Via APACHE</TITLE>";
  echo "</HEAD><BODY>";
  echo "<CENTER><H2>Testing Oracle PHP And Apache </H2><BR>";
  echo "<TABLE BORDER=1 CELLSPACING='0' WIDTH='50%'>\n<TR>\n";
  echo "<TD><B>Employee Name</B></TD>\n";
  echo "<TD><B>Salary</B></TD>\n</TR>\n";

  for ($i = 0; $i < $nrows; $i++ ) {
      echo "<TR>\n";
      echo "<TD>" . $results["ENAME"][$i] . "</TD>";
      echo "<TD>$ " . number_format($results["SAL"][$i], 2) . "</TD>";
  echo "</TR>\n";
  }

  echo "<TR><TD COLSPAN='2'>Number of Rows: $nrows</TD></TR></Table>";
  echo "</CENTER></BODY></HTML>\n";
?>
```

Point the browser to **http://sct.phptraining.com/oratest.php**. (Refer diagram 5.8)

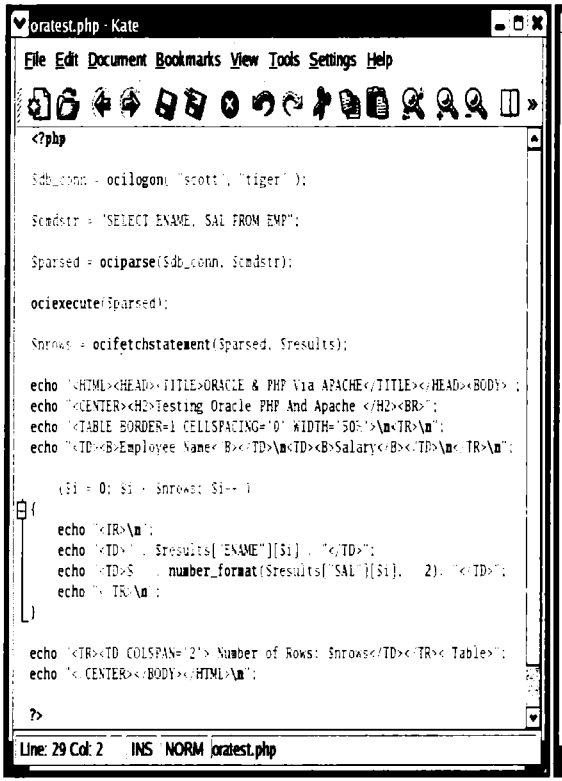


Diagram 5.7: Creating the file oratest.php in kate.

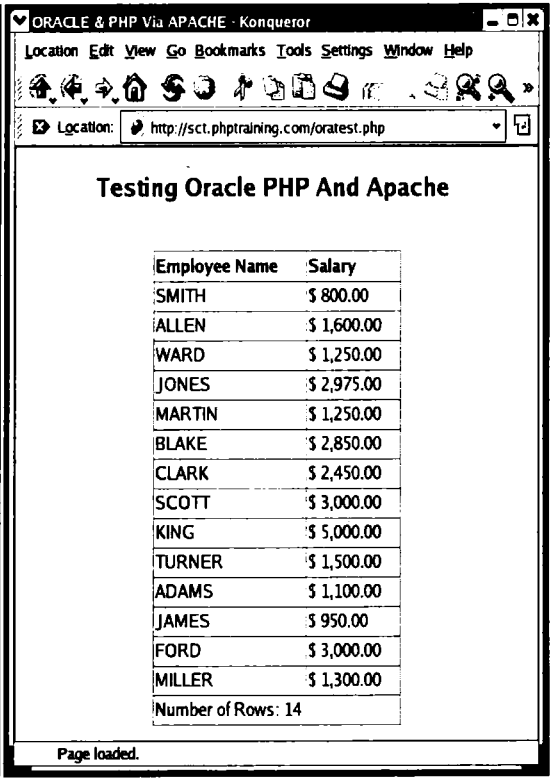


Diagram 5.8: Testing oratest.php.

This completes the testing of Oracle and PHP communication via Apache. This means the Oracle Database, Apache Web Server and PHP are installed and configured to talk with each other. At this point the framework is ready and will allow PHP applications to run and communicate with Oracle data tables in real time.

SECTION I: SETTING UP THE FRAMEWORK

Answers To Self Review Questions

3. SETTING UP APACHE

FILL IN THE BLANKS

1. apachect1 configtest, -t
2. Alias, ScriptAlias
3. container
4. Apache's API
5. MinSpareServers, MaxSpareServers
6. DocumentRoot
7. KeepAliveTimeout
8. apxs (APache eXtension)

TRUE OR FALSE

9. True
10. False
11. False
12. True
13. False
14. True
15. True
16. False
17. False
18. True

SECTION II: WORKING WITH ORACLE 10g

Basic Interaction With SQL

Basic Oracle SQL Elements

The Oracle database uses a **table** to hold user data. A table can be visualized as a two dimensional matrix consisting of rows and columns. Oracle allows each column in the table to be bound to a specific type of user data. For example a column of the table can be bound to numeric type of data. This column then will refuse to store character data within it. The Oracle database engine makes sure that user data, identical to the type of data to which the column is bound is stored in the column.

Oracle's Segregation Of User Data Types

Oracle specifically recognizes several standard user Data types. This allows a programmer to create a table with column data types carefully bound to any kind of user data type within the scope of most commercial applications. Using the correct column data type to hold user data helps in improving the performance of the Oracle database engine. It's a good idea to understand the Oracle data types in detail.

Table 6.1 has details of most of Oracle's data types:

Data Type	Description
CHAR(size [BYTE | CHAR])	Used to store a string value of fixed length. The number mentioned in the brackets specifies the number of characters the column can hold. The maximum number of characters this data type can hold is 255 characters. The data, placed in the column right-padded with spaces to the length specified is at column construction time.
VARCHAR2(size [BYTE | CHAR])	Used to store a Variable-length string value. The number mentioned in the brackets specifies the number of characters the column can hold. The maximum that this column can hold is 4000 bytes or characters. The minimum is 1 byte or 1 character. A size must be specified for VARCHAR2. BYTE indicates that the column will have byte length semantics; CHAR indicates that the column will have character semantics.

Table 6.1

Data Type	Description
NVARCHAR2 (size)	Used to store a Variable-length character string. The number mentioned in the brackets specifies the number of characters the column can hold. Maximum size is determined by the **national character** set definition, with an upper limit of 4000 bytes. A size must be specified with NVARCHAR2.
NUMBER(p, s)	Used to hold Number values having precision p and scale s. The precision p can range from 1 to 38. The scale s can range from -84 to 127. Valid values are 0 and positive or negative numbers with magnitude 1.0E-130 to 9.9 ... E125.
LONG	Used to hold Character data of variable length up to 2 gigabytes or 2^{31} -1 bytes.
DATE	Used to hold Valid date range from January 1, 4712 BC to December 31, 9999 AD. The standard format is DD-MON-YY as in 01-JAN-00.
BINARY_ FLOAT	Used to hold a 32-bit floating point number. This datatype requires 5 bytes, including the length byte.
BINARY_ DOUBLE	Used to hold 64-bit floating point number. This datatype requires 9 bytes, including the length byte.
TIMESTAMP (fractional_ seconds_precision)	Used to hold year, month and day values of date. As well as hour, minute and second values of time, where fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values of fractional_seconds_precision are 0 to 9. The default is 6.
TIMESTAMP (fractional_ seconds_precision) WITH TIME ZONE	Used to hold all values of TIMESTAMP as well as time zone displacement values where fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values are 0 to 9. The default is 6.
TIMESTAMP (fractional_ seconds_precision) WITH LOCAL TIME ZONE	Used to hold all values of TIMESTAMP WITH TIME ZONE - Except the following: ❑ Data is normalized to the database time zone when it is stored in the database. ❑ When the data is retrieved, users see the data in the session time zone.
INTERVAL YEAR (year_ precision) TO MONTH	Used to hold a period of time in years and months, where the year_precision is the number of digits in the YEAR datetime field. Accepted values are 0 to 9. The default is 2.
INTERVAL DAY (day_precision) TO SECOND (fractional_seconds _precision)	Used to hold a period of time in days, hours, minutes and seconds, where ❑ day_precision is the maximum number of digits in the DAY datetime field. Accepted values are 0 to 9. The default is 2. ❑ fractional_seconds_precision is the number of digits in the fractional part of the SECOND datetime field. Accepted values are 0 to 9. The default is 6.

Table 6.1 (Continued)

Data Type	Description
RAW(size)	Used to hold Raw binary data of the length specified in size. Maximum size is 2000 bytes. It is mandatory to specify the size for RAW.
LONG RAW	Used to hold Raw binary data of variable length up to 2 gigabytes.
ROWID	Base 64 string representing the unique address of a row in its table. This datatype is primarily for values returned by the ROWID pseudo-column.
UROWID [(size)]	Used to hold Base 64 string values that represent the logical address of a row in an index-organized table. The optional size is the size of a column of type UROWID. The maximum size and default is 4000 bytes.
NCHAR(size)	Used to hold Fixed-length character data of length specified in size. Maximum size is determined by the national character set definition, with an **upper limit of 2000 bytes**. Default and minimum size is 1 character.
CLOB	Used to hold a character type, large object, containing single-byte or multi-byte characters. Both fixed-width and variable-width character sets are supported, using Oracle's character set. Maximum size is (4 gigabytes - 1) * (database block size).
NCLOB	Used to hold a character type large object containing Unicode characters. Both fixed-width and variable-width character sets are supported, using Oracle's national character set. Maximum size is (4 gigabytes - 1) * (database block size). Stores national character set data.
BLOB	Used to hold a binary large object. Maximum size is (4 gigabytes - 1) * (database block size).
BFILE	Used to hold the path to a large binary file stored outside the database. Enables byte stream I/O access to external LOBs residing on the database server. Maximum size is 4 gigabytes.

Table 6.1 (Continued)

A table must be created before it can be used to store user data. This is done using the ANSI SQL CREATE TABLE syntax. On creation of a table to hold user data Oracle stores the table structure information within its Data Dictionary. This is often referred to as the table's META data.

Choosing A Business Model to follow

Learning ANSI SQL commands and techniques is easier if the examples used are mapped to a real-life scenario. To apply this concept this material uses examples based on a simple Personnel Management System. Its entity relationship model is as shown in diagram 6.1.

Diagram 6.1 provides a visual representation of **E**ntity **R**elationships **i.e.** the relationships between the tables used to store business data of the Personnel Management System.

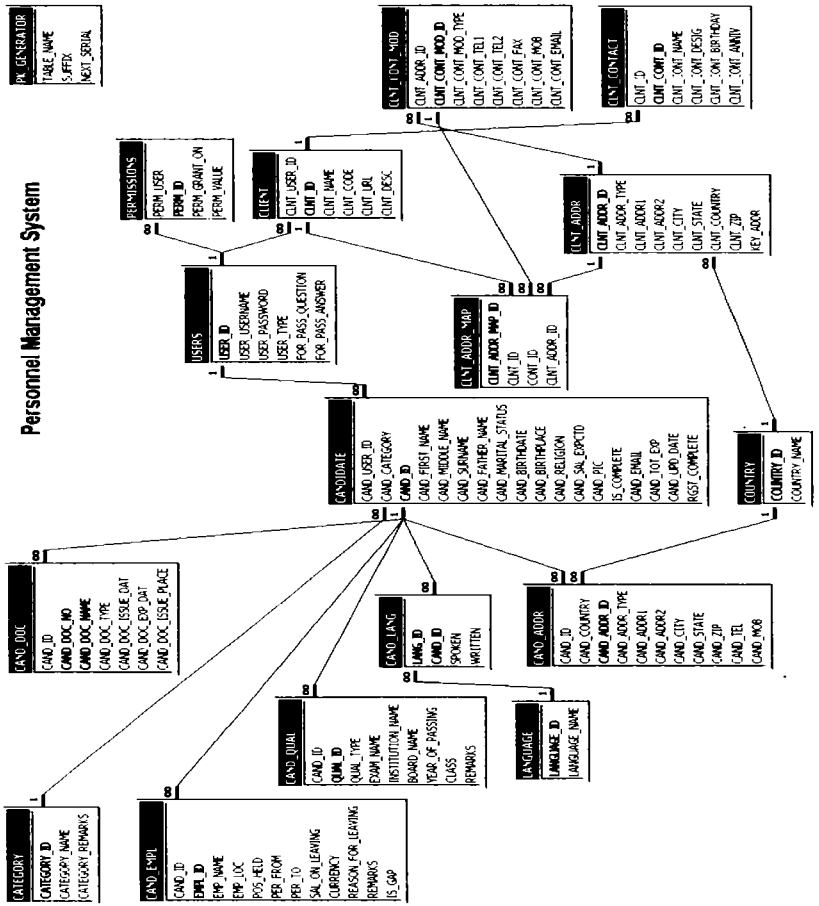


Diagram 6.1

REMINDER

The symbols **1** and **1** indicate a one-to-one relationship, while the symbols **1** and ∞ indicate a one-to-many relationship.

Fully normalized table structures together with a detailed description of each table column used in the personnel management system follow:

System And Referential Information

Table Definition:

Table Name : **PK_GENERATOR**
Primary Key : TABLE_NAME
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
TABLE_NAME	VARCHAR2	35	NOT NULL	
SUFFIX	VARCHAR2	5	NOT NULL	
NEXT_SERIAL	NUMBER	4	NOT NULL	

Table Description:

Column Name	Description
TABLE_NAME	Table name for generating ID
SUFFIX	Suffix for the table
NEXT_SERIAL	The unique number for the respective table

Explanation:
The PK_GENERATOR table stores the unique number for forming primary key for each of the tables.

Table Definition:
Table Name : **CATEGORY**
Primary Key : CATEGORY_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CATEGORY_ID	NUMBER	2	NOT NULL	
CATEGORY_NAME	VARCHAR2	35	NOT NULL	
CATEGORY_REMARKS	VARCHAR2	255		NULL

Table Description:

Column Name	Description
CATEGORY_ID	Internal ID. Auto Generated
CATEGORY_NAME	Name of the category
CATEGORY_REMARKS	Remarks for the category if any

Explanation:
The CATEGORY table is a lookup table, which records the categories available.

Table Definition:
Table Name : **COUNTRY**
Primary Key : COUNTRY_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
COUNTRY_ID	NUMBER	2	NOT NULL	
COUNTRY_NAME	VARCHAR2	35	NOT NULL	

Table Description:

Column Name	Description
COUNTRY_ID	Internal ID. Auto Generated
COUNTRY_NAME	Name of the country

Explanation:
The COUNTRY table is a lookup table, which records the names of countries.

Table Definition:
Table Name : **LANGUAGE**
Primary Key : LANGUAGE_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
LANGUAGE_ID	NUMBER	2	NOT NULL	
LANGUAGE_NAME	VARCHAR2	35	NOT NULL	

Table Description:

Column Name	Description
LANGUAGE_ID	Internal ID. Auto Generated
LANGUAGE_NAME	Name of the language

Explanation:
The LANGUAGE table is a lookup table, which records the names of languages (known to a candidate).

Authentication Information

Table Definition:
Table Name : **USERS**
Primary Key : USER_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
USER_ID	VARCHAR2	7	NOT NULL	
USER_USERNAME	VARCHAR2	15	NOT NULL	
USER_PASSWORD	VARCHAR2	20	NOT NULL	
USER_TYPE	VARCHAR2	5	NOT NULL	admn / cand / clnt
FOR_PASS_QUESTION	VARCHAR2	100		NULL
FOR_PASS_ANSWER	VARCHAR2	50		NULL

Table Description:

Column Name	Description
USER_ID	Auto Generated unique user number beginning with **US**, followed by 5 digits
USER_USERNAME	Username for the user to login
USER_PASSWORD	Password for the user
USER_TYPE	User types: candidate (cand), client (clnt), administrator (admn)
FOR_PASS_QUESTION	The question for Forgot Password Option
FOR_PASS_ANSWER	The answer for Forgot Password Option

Explanation:
The USERS table stores authentication information of system users (clients and candidates).

Table Definition:
Table Name : **PERMISSIONS**
Primary Key : PERM_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
PERM_USER	VARCHAR2	7	NOT NULL	
PERM_ID	VARCHAR2	7	NOT NULL	
PERM_GRANT_ON	VARCHAR2	35	NOT NULL	
PERM_VALUE	NUMBER	2	NOT NULL	

Table Description:

Column Name	Description
PERM_USER	User's login name. Reference existing in USERS.USER_ID
PERM_ID	Internal ID. Auto generated
PERM_GRANT_ON	Module name
PERM_VALUE	Permission value assigned

Explanation:
The PERMISSIONS table stores the permissions assigned to users of the system (clients and candidates).

6. BASIC INTERACTION WITH SQL

Client Information

Table Definition:

Table Name : **CLIENT**
Primary Key : CLNT_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_USER_ID	VARCHAR2	7		NULL
CLNT_ID	NUMBER	4	NOT NULL	
CLNT_NAME	VARCHAR2	35	NOT NULL	
CLNT_CODE	VARCHAR2	5		NULL
CLNT_URL	VARCHAR2	50		NULL
CLNT_DESC	VARCHAR2	255		NULL

Table Description:

Column Name	Description
CLNT_USER_ID	Client's login name. Reference existing in USERS.USER_ID
CLNT_ID	Internal ID number. Auto generated
CLNT_NAME	Name of the client
CLNT_CODE	Code Used by the company
CLNT_URL	URL of company website
CLNT_DESC	Any information the client wishes to furnish

Explanation:
The CLIENT table stores information of companies/clients.

Table Definition:

Table Name : **CLNT_CONTACT**
Primary Key : CLNT_CONT_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ID	NUMBER	4	NOT NULL	
CLNT_CONT_ID	NUMBER	4	NOT NULL	
CLNT_CONT_NAME	VARCHAR2	35	NOT NULL	
CLNT_CONT_DESIG	VARCHAR2	35	NOT NULL	
CLNT_CONT_BIRTHDAY	VARCHAR2	12		NULL
CLNT_CONT_ANNIV	VARCHAR2	12		NULL

Table Description:

Column Name	Description
CLNT_ID	Client's number. Reference existing in CLIENT.CLNT_ID
CLNT_CONT_ID	Internal ID number for the contact person. Auto generated
CLNT_CONT_NAME	Name of the contact person associated with the client
CLNT_CONT_DESIG	Designation
CLNT_CONT_BIRTHDAY	Birthday of the contact person
CLNT_CONT_ANNIV	Anniversary of the contact person

Explanation:
The CLNT_CONTACT table stores information of the key contacts in a particular company.

Table Definition:
Table Name : **CLNT_ADDR**
Primary Key : CLNT_ADDR_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ADDR_ID	NUMBER	4	NOT NULL	
CLNT_ADDR_TYPE	VARCHAR2	20	NOT NULL	
CLNT_ADDR1	VARCHAR2	50	NOT NULL	
CLNT_ADDR2	VARCHAR2	50		NULL
CLNT_CITY	VARCHAR2	35	NOT NULL	
CLNT_STATE	VARCHAR2	35	NOT NULL	
CLNT_COUNTRY	NUMBER	2	NOT NULL	
CLNT_ZIP	VARCHAR2	10		NULL
KEY_ADDR	VARCHAR2	3		NULL

Table Description:

Column Name	Description
CLNT_ADDR_ID	Internal ID number for the client address information. Auto generated
CLNT_ADDR_TYPE	Type of address. Whether official/personal
CLNT_ADDR1	Address Line 1
CLNT_ADDR2	Address Line 2
CLNT_CITY	City of the contact
CLNT_STATE	State
CLNT_COUNTRY	Country reference
CLNT_ZIP	Pincode of the client address
KEY_ADDR	Identifier for the key address of the client

Explanation:
The CLNT_ADDR table stores multiple addresses of the clients.

6. BASIC INTERACTION WITH SQL

Table Definition:

Table Name : **CLNT_CONT_MOD**
Primary Key : CLNT_CONT_MOD_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ADDR_ID	NUMBER	4		NULL
CLNT_CONT_MOD_ID	NUMBER	4	NOT NULL	
CLNT_CONT_MOD_TYPE	VARCHAR2	20		NULL
CLNT_CONT_TEL1	VARCHAR2	35	NOT NULL	
CLNT_CONT_TEL2	VARCHAR2	35		NULL
CLNT_CONT_FAX	VARCHAR2	35		NULL
CLNT_CONT_MOB	VARCHAR2	35		NULL
CLNT_CONT_EMAIL	VARCHAR2	50		NULL

Table Description:

Column Name	Description
CLNT_ADDR_ID	Reference for Address information for a client existing in CLNT_ADDR.CLNT_ADDR_ID
CLNT_CONT_MOD_ID	Internal ID number for the client contact information. Auto generated
CLNT_CONT_MOD_TYPE	Contact Mode Type. Whether official/personal
CLNT_CONT_TEL1	Telephone Number 1
CLNT_CONT_TEL2	Telephone Number 2
CLNT_CONT_FAX	Fax number of the contact
CLNT_CONT_MOB	Mobile number for contact
CLNT_CONT_EMAIL	Email address for contact

Explanation:

The CLNT_CONT_MOD table stores the different contact numbers/modes for a client.

Table Definition:

Table Name : **CLNT_ADDR_MAP**
Primary Key : CLNT_ADDR_MAP_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ADDR_MAP_ID	NUMBER	4	NOT NULL	
CLNT_ID	NUMBER	4	NOT NULL	
CONT_ID	NUMBER	4		NULL
CLNT_ADDR_ID	NUMBER	4		NULL

Table Description:

Column Name	Description
CLNT_ADDR_MAP_ID	Internal ID number for the link between clients and addresses. Auto generated
CLNT_ID	Client number. Reference existing in CLIENT.CLNT_ID
CONT_ID	Reference for Contact information for a client existing in CLNT_CONT_MOD.CLNT_CONT_MOD_ID
CLNT_ADDR_ID	Reference for Address information for a client existing in CLNT_ADDR.CLNT_ADDR_ID

Explanation:
The CLNT_ADDR_MAP table stores the mapping of addresses and contact information to clients.

Candidate Information

Table Definition:
Table Name : **CANDIDATE**
Primary Key : CAND_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_USER_ID	VARCHAR2	7	NOT NULL	
CAND_CATEGORY	NUMBER	2		
CAND_ID	VARCHAR2	7	NOT NULL	
CAND_FIRST_NAME	VARCHAR2	35		
CAND_MIDDLE_NAME	VARCHAR2	35		NULL
CAND_SURNAME	VARCHAR2	35		NULL
CAND_FATHER_NAME	VARCHAR2	35		NULL
CAND_MARITAL_STATUS	VARCHAR2	4	NOT NULL	
CAND_BIRTHDATE	VARCHAR2	12	NOT NULL	
CAND_BIRTHPLACE	VARCHAR2	35		NULL
CAND_RELIGION	VARCHAR2	35		NULL
CAND_SAL_EXPCTD	VARCHAR2	20		NULL
CAND_PIC	VARCHAR2	50		NULL
IS_COMPLETE	VARCHAR2	2		NULL
CAND_EMAIL	VARCHAR2	50		NULL
CAND_TOT_EXP	VARCHAR2	8		NULL
CAND_UPD_DATE	VARCHAR2	12		NULL
RGST_COMPLETE	VARCHAR2	3		NULL

Table Description:

Column Name	Description
CAND_USER_ID	Candidate's login name. Reference existing in USERS.USER_ID
CAND_CATEGORY	Category number. Reference existing in CATEGORY.CATEGORY_ID
CAND_ID	A unique address number beginning with CD, followed by 5 digits
CAND_FIRST_NAME	First Name of candidate
CAND_MIDDLE_NAME	Middle Name of candidate
CAND_SURNAME	Surname of candidate
CAND_FATHER_NAME	Father's Name of Candidate
CAND_MARITAL_STATUS	Marital Status of candidate: Yes (Yes) / No (No)
CAND_BIRTHDATE	Date of Birth
CAND_BIRTHPLACE	Place of Birth
CAND_RELIGION	Religion
CAND_SAL_EXPCTD	Salary Expected
CAND_PIC	Image Path of the Candidate's photograph
IS_COMPLETE	Status of the Record. Whether complete
CAND_EMAIL	E-mail
CAND_TOT_EXP	Total Job Experience
CAND_UPD_DATE	Last Updated Date For Experience
RGST_COMPLETE	Registration Status

Explanation:
The CANDIDATE table records personal details of the candidates.

Table Definition:

Table Name : **CAND_ADDR**
Primary Key : CAND_ADDR_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7	NOT NULL	
CAND_COUNTRY	NUMBER	2	NOT NULL	
CAND_ADDR_ID	VARCHAR2	7	NOT NULL	
CAND_ADDR_TYPE	VARCHAR2	20	NOT NULL	
CAND_ADDR1	VARCHAR2	50	NOT NULL	
CAND_ADDR2	VARCHAR2	50		NULL
CAND_CITY	VARCHAR2	35	NOT NULL	
CAND_STATE	VARCHAR2	35	NOT NULL	
CAND_ZIP	VARCHAR2	20		NULL
CAND_TEL	VARCHAR2	35		NULL
CAND_MOB	VARCHAR2	35		NULL

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
CAND_COUNTRY	Country number. Reference existing in COUNTRY.COUNTRY_ID
CAND_ADDR_ID	A unique address number beginning with **CA**, followed by 5 digits
CAND_ADDR_TYPE	Address Type: Official / Permanent / Present / Native
CAND_ADDR1	Address Line1
CAND_ADDR2	Address Line2
CAND_CITY	City of the candidate
CAND_STATE	State
CAND_ZIP	Zip code of the candidate
CAND_TEL	Telephone Number of the candidate
CAND_MOB	Mobile Number of the candidate

Explanation:
The CAND_ADDR table records multiple contact information of the candidates.

Table Definition:

Table Name : **CAND_DOC**
Primary Key : CAND_DOC_NO, CAND_DOC_NAME
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7	NOT NULL	
CAND_DOC_NO	VARCHAR2	20	NOT NULL	
CAND_DOC_NAME	VARCHAR2	16	NOT NULL	
CAND_DOC_TYPE	VARCHAR2	20		NULL
CAND_DOC_ISSUE_DAT	VARCHAR2	12		NULL
CAND_DOC_EXP_DAT	VARCHAR2	12		NULL
CAND_DOC_ISSUE_PLACE	VARCHAR2	35		NULL

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
CAND_DOC_NO	A unique passport / drivers license number
CAND_DOC_NAME	Document type: Passport (Passport) / Drivers License (Driving License)
CAND_DOC_TYPE	The type of the document, if any
CAND_DOC_ISSUE_DAT	Issue date for the document
CAND_DOC_EXP_DAT	Expiry date for the document
CAND_DOC_ISSUE_PLACE	Place of Issue for the document

Explanation:
The CAND_DOC table records the details of the documents possessed by the candidate.

Table Definition:
Table Name : **CAND_QUAL**
Primary Key : QUAL_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7	NOT NULL	
QUAL_ID	VARCHAR2	7	NOT NULL	
QUAL_TYPE	VARCHAR2	15	NOT NULL	
EXAM_NAME	VARCHAR2	50	NOT NULL	
INSTITUTION_NAME	VARCHAR2	50		NULL
BOARD_NAME	VARCHAR2	50		NULL
YEAR_OF_PASSING	VARCHAR2	10		NULL
CLASS	VARCHAR2	20		NULL
REMARKS	VARCHAR2	255		NULL

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
QUAL_ID	A number for the qualification beginning with **CQ**, followed by 5 digits
QUAL_TYPE	Records the type of qualification (Academic / Professional / Technical)
EXAM_NAME	Name of the examination
INSTITUTION_NAME	Name of the institution
BOARD_NAME	Board/University attached to
YEAR_OF_PASSING	Year of Passing
CLASS	Class obtained in the examination
REMARKS	Any details user may wish to furnish

Explanation:
The CAND_QUAL table records the qualification details of the candidate.

Table Definition:
Table Name : **CAND_LANG**
Primary Key : LANG_ID, CAND_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
LANG_ID	NUMBER	2	NOT NULL	
CAND_ID	VARCHAR2	7	NOT NULL	
SPOKEN	VARCHAR2	5	NOT NULL	
WRITTEN	VARCHAR2	5	NOT NULL	

Table Description:

Column Name	Description
LANG_ID	Language number. Reference existing in LANGUAGE.LANGUAGE_ID
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
SPOKEN	Spoken ability of the candidate for a language (Good / Fair / Poor)
WRITTEN	Written ability of the candidate for a language (Good / Fair / Poor)

Explanation:

The CAND_LANG table records the languages know to a candidate.

Table Definition:

Table Name : **CAND_EMPL**
Primary Key : EMPL_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7	NOT NULL	
EMPL_ID	VARCHAR2	7	NOT NULL	
EMP_NAME	VARCHAR2	35		
EMP_LOC	VARCHAR2	35		
POS_HELD	VARCHAR2	35		
PER_FROM	VARCHAR2	12		
PER_TO	VARCHAR2	12		
SAL_ON_LEAVING	VARCHAR2	20		
CURRENCY	VARCHAR2	25		
REASON_FOR_LEAVING	VARCHAR2	255		
REMARKS	VARCHAR2	255		
IS_GAP	VARCHAR2	3		

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
EMPL_ID	A unique employment number beginning with CE, followed by 5 digits
EMP_NAME	Name of the employer
EMP_LOC	Employer Location

Table Description: (Continued)

Column Name	Description
POS_HELD	Position Held
PER_FROM	Period From
PER_TO	Period To
SAL_ON_LEAVING	Salary on Leaving
CURRENCY	Currency in which salary is received
REASON_FOR_LEAVING	Reason for Leaving
REMARKS	Any details user wishes to furnish
IS_GAP	Indicator to identify candidates with incomplete employment information

Explanation:
The CAND_EMPL table records the employment details of the candidates.

Exploring The Oracle Database

Creating Tablespaces

The Oracle database engine stores data tables in **tablespaces**. A tablespace is a logical entity. Within the tablespace user data tables are **physically** stored in **data files**. Data files must be bound to a specific tablespace. Oracle uses a tablespace as a mechanism by which its logical objects such as tables, indexes and views are bound to a specific data file. Tablespaces can be used to group different types of (logical) database objects together.

An Oracle DBA can use tablespaces to:

- Control hard disk space allocation for database data
- Assign specific hard disk space quotas for database users
- Control the availability of data by taking individual tablespaces online or offline
- Perform partial database backup or recovery operations
- Allocate data storage across multiple data storage devices to improve database engine performance
- A database is made up of one or more tablespaces. Each tablespace is made up of one or more data files. The size of the tablespace can be determined as the sum of the sizes of all its data files. The sum of the size of all tablespaces represents the total storage capacity of the database.

REMINDER

When any database is created, the Oracle engine creates a default tablespace named System within the database.

Oracle's SYSTEM Tablespace

On the installation of Oracle for the very first time **or** when creating a new database in an existing Oracle installation using the **CREATE DATABASE** statement a **System** tablespace is automatically created with the database schema. The System tablespace holds the Oracle data dictionary, which holds the definitions of all Oracle objects within the database schema.

No other tablespace is created automatically, any additional tablespaces have to be manually created and then used. If users begin to use this database, then all user objects such as Tables, Views, Indexes and so on, will be stored in the System tablespace. This is because whenever a user is being added to the Oracle database they have to be specifically bound to a tablespace. This is so that each user will have a specific place within which their user objects can be placed.

WARNING

When a new user is bound to a database, in the absence of any other tablespace in the database, the user will be automatically bound to the **System** tablespace.

If user objects are stored in the **System** tablespace the likelihood of **space management issues** increases, which could lead to the destruction of the System tablespace and require it to be rebuilt. This is a really serious system failure if it occurs.

Oracle **strongly recommends** that a separate tablespace be created within the database and users bound to this tablespace. This ensures that Oracle objects are kept completely isolated from user objects. A **separate tablespace** can be created (within the database) either using Oracle's GUI tools or ANSI SQL syntax.

Using Tablespaces

A commercial application being developed could require a large number of tables to capture and hold user data. If application table are placed within the System tablespace, **free space** can be consumed very rapidly by the rapid growth of user tables.

If the user table consumes the free space within the System tablespace then the Oracle engine will not have space to save any of its vital database information to the System tablespace. This will cause the System tablespace to get corrupted. Once the System tablespace corrupts, the Oracle database cannot be mounted when invoked and is thus damaged.

To protect the Oracle database and prevent this happening Oracle Corp. strongly recommends that User tables and all other User objects are kept in another Tablespace and not within the System tablespace.

An **D**ata**B**ase **A**dministrator (DBA) is often part of every software development house. This individual is responsible for the day to day smooth running of the Oracle database. It is generally the responsibility of the Oracle DBA to create a separate tablespace within a database to hold user objects. It is very rare (definitely not recommended) that users are given permission to create their own tablespaces.

REMINDER

In the absence of a DBA login it may be necessary to manually create one. It is imperative that such login id's have a password to protect them and that the password is carefully preserved and changed regularly.

In the absence of a DBA login, the Sys or System logins can be used to create tablespaces. These logins are created by default when Oracle is being installed. Read the installation documentation for the default passwords assigned when the Login ID's were created.

The ANSI SQL syntax for creating tablespaces along with examples follows.

Creating A Tablespace Using SQL *Plus

Start a session of **SQL *Plus** by invoking a **Console Window** for Linux. When the Console window opens, log in as **oracle** using the command:

```
<System Prompt> su oracle ↵
```

If already logged in as an Oracle user type in the following command to start an interactive session with the oracle engine as the **sys user.** (Refer diagram 6.2)

```
<System Prompt> sqlplus sys/<password> AS sysdba ↵
```

REMINDER

It is generally the Oracle DBA who has the privileges necessary to create tablespaces in an Oracle database. Hence, the parameter **sysdba** is used.

After a successful login the **Oracle SQL * Plus** window will be visible as shown in diagram 6.2.

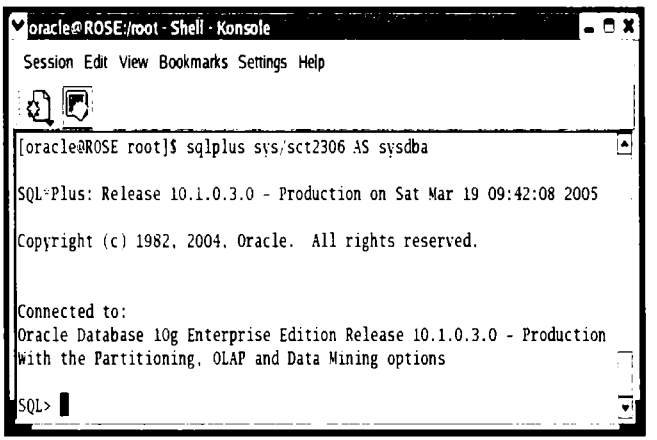


Diagram 6.2: Logging In To Oracle

The CREATE TABLESPACE Command

All user objects in an Oracle database are stored in **tablespaces**. The **CREATE TABLESPACE** command allows creating a tablespace and one or more initial data files. It also allows specifying default storage parameters.

The syntax for CREATE TABLESPACE is as follows:

```
CREATE [TEMPORARY] TABLESPACE <TablespaceName>
  DATAFILE/TEMPFILE '<Path and Filename>'
     [AUTOEXTEND [OFF/ON NEXT <integer>[K/M]
        MAXSIZE [<integer>[K/M]/UNLIMITED] ] ]
  [LOGGING/NOLOGGING] [TEMPORARY/PERMANENT] [OFFLINE/ONLINE]
  DEFAULT STORAGE <storage_clause>
     [MINIMUM EXTENT <integer>[K/M] ]
  [EXTENT MANAGEMENT
     [DIRECTORY/LOCAL
        [AUTOALLOCATE/UNIFORM [SIZE <integer>[K/M] ] ] ] ]
  [CHUNK <integer>]
  [NOCACHE]
```

The CREATE TABLESPACE command has the following keywords and parameters:

Attribute Name	Description	
TABLESPACENAME	Will accept the Name of the tablespace to be created.	
DATAFILE	Will accept the data file or data files names used within the tablespace.	
MINIMUM EXTENT	Accepts an Integer. That determines the free-space fragmentation in the tablespace by ensuring that every used and/or free extent size in a tablespace is at least as large as the integer or is a multiple of the integer.	
AUTOEXTEND	Enables or disables the automatic extension of the data files when they fill up:	
	Options	**Description**
	OFF	Disables **AUTOEXTEND** if set to **OFF**, **NEXT** and **MAXSIZE** are set to **zero**. To re-enable the feature after **AUTOEXTEND** is disabled, specify the values again for **NEXT** and **MAXSIZE** in the **ALTER TABLESPACE AUTOEXTEND** commands.
	ON	Enables **AUTOEXTEND**.
	NEXT	Accepts how much disk space to allocate to the data file when more extents are required.
	MAXSIZE	Accepts the maximum disk space allowed for allocation to the data file.
	UNLIMITED	Enables a data file to have unlimited allocation of disk space

Attribute Name	Description
LOGGING/ NOLOGGING	Accepts the default logging attributes of all tables, indexes and partitions within a tablespace. **LOGGING** is the default. If **NOLOGGING** is specified, no undo and redo logs are generated for operations that are bound to **LOGGING**. The tablespace-level, logging attribute can be overridden by logging specifications at the table, index and partition level.
DEFAULT STORAGE	Accepts the default storage parameters for all objects created in the tablespace.
PERMANENT	Accepts that the tablespace is used to hold permanent objects. This is default.
TEMPORARY	Accepts that the tablespace is used to hold only temporary objects. For example, Segments used by implicit sorts to handle ORDER BY clause.
ONLINE	Accepts that a tablespace must be available immediately after creation to users who have been granted access to the tablespace.
OFFLINE	Accepts that a tablespace is unavailable immediately after creation.

REMINDER

By default, the tablespace is set to **ONLINE** and has **LOGGING** enabled.

Attribute Name	Description
EXTENT MANAGEMENT	Can be either **DICTIONARY** (the default) or **LOCAL**. If **LOCAL** management is specified, a bitmap located in the tablespace itself is used to manage extends reducing the load on the **FET$** and **UET$** data dictionary extent management tables and recursive SQL. **LOCAL** managed extents can either be **AUTOALLOCATED** or **UNIFORM**. If **UNIFORM**, the **SIZE** for each extent in **K** or **M** can be specified.
CHUNK	Accepts a multiple of the database block size up to **32K**. Used for specifying **LOB** storage area.
NOCACHE	Accepts that the objects within the tablespace should not be cached.

The STORAGE clause and its options are as follows:

```
... STORAGE(
   [INITIAL <integer>[K/M] ] [NEXT <integer>[K/M] ]
   [MINEXTENTS <integer>] [MAXEXTENTS <integer>/UNLIMITED]
   [PCTINCREASE <integer>] [FREELISTS <integer>]
   [FREELIST GROUPS <integer>] [OPTIMAL <integer>[K/M]/NULL ]
   [BUFFER_POOL '<pool_name>' [DEFAULT] / [KEEP/RECYCLE] ]
) ...
```

The STORAGE clause has the following parameters:

Attribute Name	Description
INITIAL	Accepts the Size in bytes of the initial extent of the object segment. The default value is the size of **5 data blocks**. The minimum value is the size of **2 data blocks** for **non-bitmapped segments** or **3 data blocks** for **bitmapped segments**, plus **one data block** for each **free list group** specified. The maximum value depends on the operating system. Oracle rounds values up to the next multiple of the data block size for values less than 5 data blocks and it rounds up to the next multiple of 5 data blocks for values greater than 5 data blocks.
NEXT	Accepts the Size for the next extent after the **INITIAL** extent is used. The default is **5 blocks**, the minimum is **1 block** and the maximum is **4,095 MB**. This is the value that will be used for each new extent, if **PCTINCREASE** is set to **0**.
MINEXTENTS	Accepts the number of initial extents for an object. Generally, (except for rollback segments), it is set to **1**. If a large amount of space is required and if there's not enough contiguous space for the table, using a smaller extent size and specifying several extents may solve the problem.
MAXEXTENTS	Accepts the largest number of extents allowed for object. This defaults to the maximum allowed for the block size, as of Oracle **version 7.3**. However, it is possible to set **MAXEXTENTS** to unlimited after **version 8**, allowing over **2 billion extents**. However, Oracle suggests **not** going over **4,000 extents** for a single object.
PCTINCREASE	Accepts how much to grow each extent after the **INITIAL** and **NEXT** extents are used. A specification of **50** will grow each extent after **NEXT** by **50%** for each subsequent extent. This means that for a table created with one **INITIAL** and a **NEXT** extent, any further extents will increase in size by **50%** over its predecessors. The value of **PCTINCREASE** indicates a growth rate for subsequent extents. A tablespace with a default storage setting for **PCTINCREASE** of **0** will not be automatically coalesced by the SMON process.
OPTIMAL	Used only for rollback segments. Specifies the value to which a rollback segment will **shrink** after extending.
FREELISTS	Used for objects other than tablespaces, specifies the number of freelists for each of the freelist groups for the table, index or cluster. The minimum value is **1** and the maximum is block-size dependent.

Attribute Name	Description	
BUFFER_POOL <pool_name>	Accepts the area of the buffer pool where the object will be cached. The **<Pool_Name>** parameter corresponds to:	
	DEFAULT	The value assigned if no **BUFFER_POOL** parameter is specified
	KEEP	For objects that should not be rapidly aged out of the buffer pool.
	RECYCLE	For objects that should be rapidly aged out of the buffer pool.
	The **KEEP** and **RECYCLE** pools are sub-sections of the **DEFAULT** pools and must be configured in the initialization parameters **BUFFER_POOL_KEEP** and **BUFFER_POOL_RECYCLE** before being used.	
FREELISTS GROUPS	This parameter accepts the number of freelist groups to maintain for a table or index. This parameter is generally meaningful for only parallel server database and cannot be specified unless a database is altered into parallel or share mode.	

REMINDER

Once a tablespace is created, it must be Online to be used. The create tablespace syntax, by **default**, ensures that the Tablespace is placed online. Only when the tablespace is online can users bound to the tablespace use it for storing their objects.

Example 1.1:
Create a tablespace named **SCT_ADMIN** connected to data file **sct_admin.dat** with a size of **10 MB**.

```
CREATE TABLESPACE SCT_ADMIN DATAFILE 'sct_admin.dat' SIZE 10M ONLINE;
```

Example 1.2:
Create a tablespace named **PM_SYS**. The name of the data file is **PM_SYS.dat** and the size is **20 MB**. The specification for the tablespace storage parameters is as follows:

INITIAL extent size 10k
NEXT EXTENT size 50k
MINEXTENTS 1
MAXEXTENTS 999
PCTINCREASE 10

```
CREATE TABLESPACE PM_SYS
  DATAFILE 'PM_SYS.dat' SIZE 20M
  DEFAULT STORAGE(INITIAL 10K NEXT 50K MINEXTENTS 1
     MAXEXTENTS 999 PCTINCREASE 10) ONLINE;
```

When the above SQL commands are typed into Oracle **SQL * Plus** window the results are as shown in diagram 6.3.1 and diagram 6.3.2.

```
SQL> CREATE TABLESPACE SCT_ADMIN DATAFILE 'sct_admin.dat' SIZE 10M ONLINE;

Tablespace created.

SQL>
```

Diagram 6.3.1

```
SQL> CREATE TABLESPACE PM_SYS
  2   DATAFILE 'PM_SYS.dat' SIZE 20M
  3   DEFAULT STORAGE(
  4    INITIAL 10K NEXT 50K MINEXTENTS 1 MAXEXTENTS 999 PCTINCREASE 10
  5   ) ONLINE;

Tablespace created.

SQL>
```

Diagram 6.3.2

Creating And Binding A User To The Tablespace

To begin exercising examples a user called DBA_PM needs to be created using the **sys** login. (Refer Diagram 6.4.1)

```
CREATE USER "DBA_PM"
  PROFILE "DEFAULT"
  IDENTIFIED BY "sct2306"
  DEFAULT TABLESPACE "PM_SYS"
  TEMPORARY TABLESPACE "TEMP"
  ACCOUNT UNLOCK;
```

Make the user DBA by issuing the following command: (Refer Diagram 6.4.2)

```
GRANT "DBA" TO "DBA_PM" WITH ADMIN OPTION;
```

```
SQL> CREATE USER "DBA_PM"
  2   PROFILE "DEFAULT"
  3   IDENTIFIED BY "sct2306"
  4   DEFAULT TABLESPACE "PM_SYS"
  5   TEMPORARY TABLESPACE "TEMP"
  6   ACCOUNT UNLOCK;

User created.

SQL>
```

Diagram 6.4.1

```
SQL> GRANT "DBA" TO "DBA_PM" WITH ADMIN OPTION;

Grant succeeded.

SQL>
```

Diagram 6.4.2

Now onwards use the **DBA_PM** user to log in to oracle and exercise the following examples.

Table Creation Rules And Checklist

Rules For Creating Tables

1. A table name can have maximum of 30 characters
2. Alphabets from A-Z, a-z and numbers from 0-9 are allowed
3. A table name should begin with an alphabet
4. The use of the special character like _ is allowed and also recommended. (Special characters like $, # are allowed **only in Oracle**).
5. SQL reserved words **not** allowed. For Example: create, select and so on.

Syntax:

```
CREATE TABLE [Schema.]<TableName> (
  <ColumnName1> <DataType>(<Size>)[, . . . ] );
```

REMINDER

Each column must have a datatype. The column should either be defined as null or not null and if this value is left blank, the database assumes "null" as the default.

A Brief Checklist When Creating Tables

The following provides a small checklist for the issues that need to be considered before creating a table:

- What are the attributes of the rows to be stored?
- What are the data types of the attributes?
- Can varchar2 be used instead of char?
- Which columns should be used to build the primary key?
- Which columns should not allow null values? Which columns do / do not allow duplicates?
- Are there default values for certain columns that **also** allow null values?

Creating Tables

In the **CREATE TABLE** command each column of the table is defined uniquely. Each column has a minimum of three attributes, a name, datatype and size (**i.e.** column width). Each table column definition is a **single** clause in the create table syntax. Each table column definition is separated from the other by a comma. Finally, the SQL statement is terminated with a semi colon.

Example 2:
Create the **COUNTRY** table belonging to the **Personnel Management System**.

```
CREATE TABLE COUNTRY(
  COUNTRY_ID Number(2), COUNTRY_NAME VarChar2(35));
```

Output:

```
Table created.
```

REMINDER

All table columns belong to a **single record**. Therefore all the table column definitions are enclosed within parenthesis.

Displaying A Table's Structure

To display information about the columns defined in a table use the following syntax

Syntax:

```
DESCRIBE <TableName>;
```

This command displays the column names, the data types and the special attributes connected to the table.

Example 3:
Show the structure of table **COUNTRY**

```
DESCRIBE COUNTRY;
```

Output:

```
Name                           Null?    Type
------------------------------ -------- --------------------
COUNTRY_ID                              NUMBER(2)
COUNTRY_NAME                            VARCHAR2(35)
```

The Oracle Table - DUAL

The Oracle user **SYS** owns the table **DUAL**. The Oracle database, data dictionary is owned by SYS and the table DUAL is part of the data dictionary. DUAL is a small Oracle worktable, which consists of only one row and one column and contains the value **x** in that column. Besides arithmetic calculations, it also supports **date** retrieval and formatting.

If simple arithmetic needs to be done, for example: 2*2, the only ANSI SQL verb produce an output to the VDU is **SELECT**. However, a SELECT **must have** a table name in it's FROM clause, otherwise the SELECT will **fail**.

When an arithmetic exercise is to be performed such as 2*2 or 4/2 and so on (only **numeric literals** being used) and there is no table being referenced hence the SELECT must fail.

To enable such calculations via the use of a SELECT, Oracle provides the **worktable** called **DUAL**, against which SELECT statements used to manipulate numeric literals can be fired and appropriate output obtained.

The structure of DUAL is as follows:

```
DESC DUAL;
```

Output:

```
Name                 Null?    Type
-------------------- -------- ---------------
DUMMY                         VARCHAR2(1)
```

If the dual table is queried for records the output is as follows:

```
SELECT * FROM DUAL;
```

Output:

```
D
-
X
```

Example 4:

```
SELECT 2*2 FROM DUAL;
```

Output:

```
       2*2
---------
         4
```

Performing Operations On Table Data

Populating Tables With Data

Once a table is created, the most natural thing to do is load this table with data to be manipulated later.

When inserting a single row of data into the table, the insert operation:

- Creates a new empty row in the database table
- Loads the values passed (by the SQL insert) into the empty columns specified

Syntax:

```
INSERT INTO <TableName> (<ColumnName1>[, ...])
  VALUES (<Expression1>[, <Expression2>]);
```

Example 5:
Insert the following values into the COUNTRY table:

Table Name: COUNTRY

COUNTRY_ID	COUNTRY_NAME
1	India
3	United Kingdoms
5	Pakistan

COUNTRY_ID	COUNTRY_NAME
2	United States Of America
4	United Arab Emirates
6	Sri Lanka

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES (1, 'India');
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME)
  VALUES (2, 'United States Of America');
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME)
  VALUES (3, 'United Kingdoms');
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME)
  VALUES (4, 'United Arab Emirates');
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES (5, 'Pakistan');
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES (6, 'Sri Lanka');
```

Output for each of the above INSERT INTO statements:

```
1 row created.
```

REMINDER

The SQL statements for creating and inserting records in the Personnel Management System are available under *Book_CDROM:/Chap06_Codes/* on the accompanying Book CD-ROM. *To continue with the following examples these records have to be fired in SQL *Plus using a copy paste technique.*

HINT

Character expressions used within **INSERT INTO** must be enclosed in **single quotes** (').

In **INSERT INTO**, table columns and their values have a **one to one relationship**, (**i.e.** the first value described is inserted into the first column, while the second value described is inserted into the second column and so on).

Hence, in an **INSERT INTO** SQL sentence if there are exactly the same numbers of values as there are columns and the values are sequenced exactly in accordance with the data type of the table columns, there is **no need** to indicate the column names.

However, if there are less values being described than there are columns in the table, then it is **mandatory** to indicate **both** the table **column name** and its corresponding **value** in the **INSERT INTO** SQL sentence.

In the absence of mapping a table column name to a value in the **INSERT INTO** SQL sentence; the Oracle engine will not know which columns to insert data into. This will generally cause a loss of data integrity. Then the data held within the table will be largely useless.

Globally Retrieving Data From Tables

Once data has been inserted into a table, the next most logical operation would be to view what has been inserted. The **SELECT** SQL verb is used to achieve this. The SELECT command is used to retrieve rows selected from one or more tables.

All Rows And All Columns

In order to view global table data the **syntax** is:

```
SELECT <ColumnName1> TO <ColumnNameN> FROM <TableName>;
```

REMINDER

Here, **ColumnName1 to ColumnNameN** represents table column names.

Syntax:

```
SELECT * FROM <TableName>;
```

Example 6:
Display the details held in the **CAND_LANG** table.

An extract of the **CAND_LANG** table is as shown below:

Table Name: CAND_LANG

LANG_ID	CAND_ID	SPOKEN	WRITTEN
1	CD00001	Good	Fair
2	CD00003	Poor	Poor
5	CD00004	Good	Fair
3	CD00005	Good	Good

LANG_ID	CAND_ID	SPOKEN	WRITTEN
1	CD00002	Good	Good
1	CD00004	Good	Good
3	CD00004	Good	Good
6	CD00005	Fair	Poor

```
SELECT * FROM CAND_LANG;
```

Output:

```
   LANG_ID CAND_ID SPOKE WRITT
---------- ------- ----- -----
         1 CD00001 Good  Fair
         1 CD00002 Good  Good
         2 CD00003 Poor  Poor
         1 CD00004 Good  Good
         5 CD00004 Good  Fair
         3 CD00004 Good  Good
         3 CD00005 Good  Good
         6 CD00005 Fair  Poor
8 rows selected.
```

Example 7:
Show all details held in the **COUNTRY** table.

A partial extract of the **COUNTRY** table is as shown below:

Table Name: COUNTRY

COUNTRY_ID	COUNTRY_NAME
1	India
3	United Kingdoms
5	Pakistan

COUNTRY_ID	COUNTRY_NAME
2	United States Of America
4	United Arab Emirates
6	Sri Lanka

```
SELECT * FROM COUNTRY;
```

Output:

```
COUNTRY_ID COUNTRY_NAME
---------- ------------------------------
         1 India
         2 United States Of America
         3 United Kingdoms
         4 United Arab Emirates
         5 Pakistan
         6 Sri Lanka
8 rows selected.
```

HINT

When data from all rows and columns in the table must be viewed the syntax of the SELECT statement will be: **SELECT * FROM** <TableName>;

Oracle allows the use of the Meta character asterisk (*), this is expanded by Oracle to mean all rows and all columns in the table.

The Oracle database engine parses and compiles the SQL query, executes it and retrieves data from all rows/columns from the table.

Retrieving Filtered Table Data

While viewing data from a table it is rare that **all the data** from the table will be required **each time**. Hence, SQL provides a method of filtering table data.

The ways of filtering table data are:

- Selected columns and all rows
- Selected rows and all columns
- Selected columns and selected rows

Selected Columns And All Rows

The retrieval of specific columns from a table can be done as shown below:

Syntax:

```
SELECT <ColumnName1>[, <ColumnName2>, . . . ] FROM <TableName>;
```

Example 8:
Show the user name and type of the user's of the personnel management system. A partial extract of the **USERS** table is as shown below:

Table Name: USERS

USER_ID	USER_USERNAME	USER_TYPE
US00000	admin	admn
US00002	hansel	cand
US00004	chhaya	cand
US00006	vaishali	clnt
US00008	Ajay	clnt

USER_ID	USER_USERNAME	USER_TYPE
US00001	sharanam	cand
US00003	ivan	cand
US00005	shravan	cand
US00007	Vijay	clnt
US00009	Joshi	clnt

```
SELECT USER_USERNAME, USER_TYPE FROM USERS;
```

Output:

```
USER_USERNAME   USER TYPE
--------------- ---------
admin           admn
sharanam        cand
hansel          cand
ivan            cand
chhaya          cand
shravan         cand
vaishali        clnt
Vijay           clnt
Ajay            clnt
Joshi           clnt
10 rows selected.
```

Selected Rows And All Columns

If a particular client's information is to be retrieved from a table, this information must be retrieved based on a **specific condition**.

Till now the SELECT statement displayed all rows in a table. This is because there was no condition set that informed Oracle about how to choose a specific row (**or** a set of rows) from a table. To select a specific row (or a set of rows) Oracle permits the use of a **WHERE Clause** in the SQL query.

On declaring a where clause in the SQL query, the Oracle engine compares each record in the table with the condition specified in the where clause and displays only those records that satisfy the condition specified.

Syntax:

```
SELECT * FROM <TableName> WHERE <Condition>;
```

here, **<Condition>** is always quantified as **<ColumnName = Value>**

Example 9:
Display details held in the **CAND_LANG** table, where in the candidates have GOOD speaking skills. An extract of the **CAND_LANG** table is as shown below:

Table Name: CAND_LANG

LANG_ID	CAND_ID	SPOKEN	WRITTEN
1	CD00001	Good	Fair
2	CD00003	Poor	Poor
5	CD00004	Good	Fair
3	CD00005	Good	Good

LANG_ID	CAND_ID	SPOKEN	WRITTEN
1	CD00002	Good	Good
1	CD00004	Good	Good
3	CD00004	Good	Good
6	CD00005	Fair	Poor

```
SELECT * FROM CAND_LANG WHERE SPOKEN = 'Good';
```

Output:

```
   LANG_ID CAND_ID SPOKE WRITT
---------- ------- ----- -----
         1 CD00001 Good  Fair
         1 CD00002 Good  Good
         1 CD00004 Good  Good
         5 CD00004 Good  Fair
         3 CD00004 Good  Good
         3 CD00005 Good  Good
6 rows selected.
```

REMINDER

When specifying a condition in the **WHERE clause** all standard operators such as logical, arithmetic, predicates and so on, can be used.

Selected Columns And Selected Rows

To view a specific set of rows and columns from a table the syntax will be as described below.

Syntax:

```
SELECT <ColumnName1>[, <ColumnName2>, . . . ] FROM <TableName>
  WHERE <Condition>;
```

Example 10:
Display the candidate's ID and the address ID of candidates who have submitted a permanent resident address. A partial extract of the **CAND_ADDR** table is as shown below:
Table Name: CAND_ADDR

CAND_ID	CAND_ADDR_ID	CAND_ADDR_TYPE
CD00001	CA00001	Present
CD00002	CA00003	Present
CD00003	CA00005	Present
CD00004	CA00007	Present

CAND_ID	CAND_ADDR_ID	CAND_ADDR_TYPE
CD00001	CA00002	Present
CD00003	CA00004	Permanent
CD00004	CA00006	Permanent
CD00005	CA00008	Permanent

```
SELECT CAND_ID, CAND_ADDR_ID FROM CAND_ADDR
  WHERE CAND_ADDR_TYPE = 'Permanent';
```

Output:

```
CAND_ID CAND_AD
------- -------
CD00003 CA00004
CD00004 CA00006
CD00005 CA00008
```

Non-Duplicate Rows

A table could hold duplicate rows. In such a case, to view only unique rows the **DISTINCT** clause can be used. The DISTINCT clause allows removing duplicates from the result set. The DISTINCT clause can only be used with select statements.

The **DISTINCT** clause scans through the values of the column/s specified and displays only unique values from amongst them.

Syntax:

```
SELECT DISTINCT <ColumnName1>[, <ColumnName2>, . . . ] FROM <TableName>;
```

The **SELECT DISTINCT** * SQL syntax scans through **entire rows** and eliminates rows that have exactly the same contents in each column.

Syntax:

```
SELECT DISTINCT * FROM <TableName>;
```

Example 11:
Show different cities of candidate's residence by eliminating the repeated city names

```
SELECT DISTINCT CAND_CITY FROM CAND_ADDR;
```

Output:

```
CAND_CITY
--------------------
Mira Road, Dist.Thane
Mumbai
NAVI MUMBAI
Palakkad
Sydney
THIRUVANANTHAPURAM
6 rows selected.
```

Example 12:
Show the language proficiency of candidates without any repetitions. An extract of the **CAND_LANG** table is as shown below:

Table Name: CAND_LANG

LANG_ID	CAND_ID	SPOKEN	WRITTEN
1	CD00001	Good	Fair
2	CD00003	Poor	Poor
5	CD00004	Good	Fair
3	CD00005	Good	Good

LANG_ID	CAND_ID	SPOKEN	WRITTEN
1	CD00002	Good	Good
1	CD00004	Good	Good
3	CD00004	Good	Good
6	CD00005	Fair	Poor

First insert one more record (To create duplicates rows) in the table **CAND_LANG** so as to see the output for this example.

```
INSERT INTO CAND_LANG (LANG_ID, CAND_ID, SPOKEN, WRITTEN)
  VALUES (6, 'CD00005', 'Fair', 'Poor');
SELECT DISTINCT * FROM CAND_LANG;
```

The following output shows the entry for LANG_ID 6 only once even though entered twice in the table.

Output:

```
   LANG_ID CAND_ID SPOKE WRITT
---------- ------- ----- -----
         1 CD00001 Good  Fair
         1 CD00002 Good  Good
         2 CD00003 Poor  Poor
         1 CD00004 Good  Good
         5 CD00004 Good  Fair
         3 CD00004 Good  Good
         3 CD00005 Good  Good
         6 CD00005 Fair  Poor
8 rows selected.
```

Adding Line Feeds To Select Statement Output

Example 13:
Display the Candidate Details as:
Candidate Name: Sharanam Shah ↵
Birthdate: 03-Jan-1981 ↵
Birthplace: Mumbai ↵

REMINDER

The return carriage symbol (↵) shown above is used to indicate a newline character and are not printed in the output.

Solution:

```
SELECT 'Candidate Name: ' || CAND_FIRST_NAME || ' ' || CAND_SURNAME ||
  CHR(10) || 'Birthdate: ' || CAND_BIRTHDATE || CHR(10) || 'Birthplace: ' ||
  CAND_BIRTHPLACE "Candidate Details" FROM CANDIDATE;
```

Output:

```
Candidate Details
-----------------------------
Candidate Name: Sharanam Shah
Birthdate: 03/01/1981
Birthplace:  Mumbai

Candidate Name: Hansel Colaco
Birthdate: 01/01/1982
Birthplace:  Mumbai

Candidate Name: Ivan Bayross
Birthdate: 23/06/1952
Birthplace:  Mumbai

Candidate Name: Chhaya Bankar
Birthdate: 22/06/1976
Birthplace:  Palakkad

Candidate Name: Shravan Dhone
Birthdate: 21/02/1982
Birthplace:  Thane
```

Sorting Data In A Table

Oracle allows data from a table to be viewed in a sorted order. The rows retrieved from the table will be sorted in either **ascending** or **descending** order depending on the condition specified in the **SELECT** sentence. The syntax for viewing data in a sorted order is as follows:

Syntax:

```
SELECT * FROM <TableName>
  ORDER BY <ColumnName1>[, <ColumnName2>, . . .] [<Sort Order>];
```

The **ORDER BY** clause sorts the result set based on the columns specified. The **ORDER BY** clause can only be used in **SELECT** statements.

Example 14:
Show the candidate ID and name details after ascending data in accordance to the first name of the candidate. A partial extract of the **CANDIDATE** table is as shown below:

Table Name: CANDIDATE

CAND_ID	CAND_FIRST_NAME	CAND_SURNAME
CD00001	Sharanam	Shah
CD00002	Hansel	Colaco
CD00003	Ivan	Bayross
CD00004	Chhaya	Bankar
CD00005	Shravan	Dhone

```
SELECT CAND_ID, CAND_FIRST_NAME, CAND_SURNAME FROM CANDIDATE
  ORDER BY CAND_FIRST_NAME;
```

Output:

```
CAND_ID CAND_FIRST_NAME CAND_SURNAME
------- --------------- --------------
CD00004 Chhaya          Bankar
CD00002 Hansel          Colaco
CD00003 Ivan            Bayross
CD00001 Sharanam        Shah
CD00005 Shravan         Dhone
```

HINT

For viewing data in descending sorted order the word **DESC** must be mentioned **after** the column name and before the semi colon in the **order by** clause. In case there is no mention of the sort order, sorting is in **ascending order**.

Example 15:
Show the candidate ID and name in accordance to the first name of the candidate in descending order.

```
SELECT CAND_ID, CAND_FIRST_NAME, CAND_SURNAME FROM CANDIDATE
  ORDER BY CAND_FIRST_NAME DESC;
```

Output:

```
CAND_ID CAND_FIRST_NAME CAND_SURNAME
------- --------------- --------------
CD00005 Shravan         Dhone
CD00001 Sharanam        Shah
CD00003 Ivan            Bayross
CD00002 Hansel          Colaco
CD00004 Chhaya          Bankar
```

View Every Nth Row From A Table

In Oracle, to select all even, odd or Nth rows from a table use SQL queries like:

Example 16:

Solution 1: Using Sub Queries

```
SELECT ROWNUM RN, CAND_ID, CAND_FIRST_NAME FROM CANDIDATE
  WHERE (ROWID, 0) IN (SELECT ROWID, MOD(ROWNUM, 2)
    FROM CANDIDATE);
```

Output:

```
        RN CAND_ID CAND_FIRST_NAME
---------- ------- ----------------
         1 CD00002 Hansel
         2 CD00004 Chhaya
```

Solution 2: Using dynamic views

```
SELECT * FROM (SELECT ROWNUM RN, CAND_ID, CAND_FIRST_NAME
  FROM CANDIDATE) C WHERE MOD(C.RN, 2) = 0;
```

Output:

```
        RN CAND_ID CAND_FIRST_NAME
---------- ------- ----------------
         2 CD00002 Hansel
         4 CD00004 Chhaya
```

Solution 3: Using GROUP BY and HAVING

```
SELECT ROWNUM, CAND_ID, CAND_FIRST_NAME FROM CANDIDATE
  GROUP BY ROWNUM, CAND_ID, CAND_FIRST_NAME
    HAVING MOD(ROWNUM, 2) = 0 OR ROWNUM = 2-0;
```

Output:

```
        RN CAND_ID CAND_FIRST_NAME
---------- ------- ----------------
         2 CD00002 Hansel
         4 CD00004 Chhaya
```

Retrieve Only Rows X To Y From A Table

Example 17:

Retrieve records (The Candidate Name) ranging between 4 and 7 from the **CANDIDATE** table

Solution 1:

```
SELECT * FROM (SELECT ROWNUM RN, CAND_FIRST_NAME FROM CANDIDATE
  WHERE ROWNUM < 8) WHERE RN BETWEEN 4 AND 7;
```

Note that **8** is just one greater than the maximum row of the required rows. This means x=4, y=7, so the inner value is y+1 i.e. 8).

Output:

```
        RN CAND_FIRST_NAME
---------- ---------------
         4 Chhaya
         5 Shravan
```

Solution 2:

```
SELECT ROWNUM RN, CAND_FIRST_NAME FROM CANDIDATE
  GROUP BY ROWNUM, CAND_FIRST_NAME
     HAVING ROWNUM BETWEEN 4 AND 7;
```

Output:

```
        RN CAND_FIRST_NAME
---------- ---------------
         4 Chhaya
         5 Shravan
```

Solution 3:

```
SELECT ROWNUM RN, CAND_FIRST_NAME FROM CANDIDATE WHERE ROWID IN(
  SELECT ROWID FROM CANDIDATE WHERE ROWNUM <= 7
  MINUS
  SELECT ROWID FROM CANDIDATE WHERE ROWNUM < 4);
```

Output:

```
        RN CAND_FIRST_NAME
---------- ---------------
         1 Chhaya
         2 Shravan
```

Automatically Generating SQL Statements

The technique of writing SQL statements that produce other SQL statements is very useful. It reduces effort put into writing SQL statements that are similar. The automatically generated SQL statements can be saved to a file and used on demand.

Example 18:
The following query produces a series of DROP TABLE statements that drop the tables in the **DBA_PM** user schema:

```
SELECT 'DROP TABLE ' || TABLE_NAME || ';' FROM USER_TABLES;
```

REMINDER

The table **USER_TABLE** contains the details of the tables in the user's schema. The **Table_Name** column contains names of the tables.

Output:

```
'DROPTABLE'||TABLE_NAME||';'
----------------------------------------
DROP TABLE CATEGORY;
DROP TABLE PK_GENERATOR;
DROP TABLE COUNTRY;
DROP TABLE LANGUAGE;
DROP TABLE USERS;
DROP TABLE PERMISSIONS;
DROP TABLE CLIENT;
DROP TABLE CLNT_CONTACT;
DROP TABLE CLNT_ADDR;
DROP TABLE CLNT_CONT_MOD;
DROP TABLE CLNT_ADDR_MAP;
DROP TABLE CANDIDATE;
DROP TABLE CAND_ADDR;
DROP TABLE CAND_DOC;
DROP TABLE CAND_QUAL;
DROP TABLE CAND_LANG;
DROP TABLE CAND_EMPL;

17 rows selected.
```

Passing A Value To A Variable In A Script

Oracle 10g facilitate run-time passing of value(s) to variable(s) contained in SQL scripts. The variable(s) in the script are referred using a number. A script may contain multiple variables and are denoted by preceding a number with an ampersand (&). While executing the script, the values are passed immediately after naming the script file.

Example 19:
The following script (stored as **CandRprt.sql**) identifies the variable in the script file by using the **&1:**

```
SET ECHO OFF
SET VERIFY OFF

SELECT CAND_ID, CAND_FIRST_NAME, CAND_SURNAME FROM CANDIDATE
  WHERE Cand_Category = (SELECT CATEGORY_ID FROM CATEGORY
      WHERE CATEGORY_NAME = '&1');
```

REMINDER

The file CandRprt.sql can be created using the any Linux based text editor (i.e. KATE).

To execute the script **CandRprt.sql** enter the following command at the SQL prompt:

```
@/oraclefiles/CandRprt.sql 'Arabic Cook'
```

Assuming the files was created and saved under **/oraclefiles**.

This command will list all the Arabic Cooks.

Output:

```
CAND_ID CAND_FIRST_NAME                    CAND_SURNAME
------- ---------------------------------- ------------
CD00005 Shravan                            Dhone
```

```
@/oraclefiles/CandRprt.sql 'Analyst Programmer'
```

This command will list all the Analyst Programmers.

Output:

```
CAND_ID CAND_FIRST_NAME                    CAND_SURNAME
------- ---------------------------------- ------------
CD00001 Sharanam                           Shah
CD00003 Ivan                               Bayross
CD00004 Chhaya                             Bankar
```

Deleting Data From Tables

The DELETE command deletes rows from the table that satisfies the condition provided by its where clause **and** returns the number of records deleted.

WARNING

If a **DELETE** statement without a **WHERE** clause is issued then all rows are deleted.

The verb **DELETE** in SQL is used to remove either:

- All the rows from a table

OR

- A set of rows from a table

Delete All Rows

Syntax:

```
DELETE FROM <TableName>;
```

Example 20:
Empty the **CANDIDATE** table

```
DELETE FROM CANDIDATE;
```

Output:

```
5 rows deleted.
```

Delete Specific Row(s)

Syntax:

```
DELETE FROM <TableName> WHERE <Condition>;
```

Example 21:
Remove only those languages information belonging to candidate CD00005 from the **CAND_LANG** table.

```
DELETE FROM CAND_LANG WHERE CAND_ID = 'CD00005';
```

Output:

```
2 rows deleted.
```

Updating Data In Tables

The **UPDATE** command is used to change or modify data values in a table.

The verb update in SQL is used to either update:

- All the rows from a table

OR

- A set of rows from a table

Updating All Rows

The **UPDATE** statement updates the columns specified in all the existing table's rows with new values. The **SET** clause indicates which column data should be modified and the new values that they should hold. The **WHERE** clause, if given, defines particular rows that should be updated. Otherwise, **all** table rows are updated.

Syntax:

```
UPDATE <TableName>
  SET <ColumnName1>=<Expression1>[, <ColumnName2>=<Expression2>, . . .];
```

Example 22:
Update the client's address details by changing its city name to Bombay

```
UPDATE CLNT_ADDR SET CLNT_CITY = 'Bombay';
```

Output:

```
8 rows updated.
```

Update Specific Row(s)

Syntax:

```
UPDATE <TableName>
  SET <ColumnName1>=<Expression1>[, <ColumnName2>=<Expression2>, . . .]
     WHERE <Condition>;
```

Example 23:
Update the candidate's address details by changing the state named 'Mumbai' to 'Bombay'.

```
UPDATE CAND_ADDR SET CAND_CITY='Bombay' WHERE CAND_CITY='Mumbai';
```

Output:

```
3 rows updated.
```

New Datatypes In Oracle 10G

BINARY_FLOAT And BINARY_DOUBLE Data Types

One of the evolutionary changes that Oracle 10g has brought, in terms of front-end support for new language features include the BINARY_FLOAT and BINARY_DOUBLE datatypes. Both these features have an efficient and robust implementation in machine arithmetic for mathematical real numbers.

A positive outcome of this implementation is the fact that algorithmic tasks that are expressed in PL/SQL using these new features can now run **much faster** than an implementation that does not have these features.

- BINARY_DOUBLE is a 64-bit, double-precision floating-point number datatype. Each BINARY_DOUBLE value requires 9 bytes, including a length byte.
- BINARY_FLOAT is a 32-bit, single-precision floating-point number datatype. Each BINARY_FLOAT value requires 5 bytes, including a length byte.

In a NUMBER column, floating point numbers have decimal precision. In a BINARY_FLOAT or BINARY_DOUBLE column, floating-point numbers have binary precision. The binary floating-point numbers support the special values **infinity and NaN (not a number)**.

Example 1:
The following statement creates a table named **EMPEXP** that contains BINARY_FLOAT and BINARY_DOUBLE columns:

```
CREATE TABLE EMPEXP (EMPNO NUMBER(2) PRIMARY KEY,
  LASTSAL BINARY_FLOAT, EXPSAL BINARY_DOUBLE);
```

Example 2:
The following statements populate the table named SCT:

```
INSERT INTO EMPEXP (EMPNO, LASTSAL, EXPSAL) VALUES('1', 232.7f, 242.7d);
INSERT INTO EMPEXP (EMPNO, LASTSAL, EXPSAL) VALUES('2', 452.2f, 470.9d);
```

Example 3:
The following statement describes the table named **EMPEXP:**

```
DESC EMPEXP;
```

Output:

```
Name        Null?    Type
--------- -------- -------------
EMPNO       NOT NULL NUMBER(2)
LASTSAL              BINARY_FLOAT
EXPSAL               BINARY_DOUBLE
```

Example 4:
The following statement displays the contents of the table named EMPEXP:

```
SELECT * FROM EMPEXP;
```

Output:

```
     EMPNO     LASTSAL     EXPSAL
---------- ---------- ----------
         1 2.327E+002 2.427E+002
         2 4.522E+002 4.709E+002
```

In addition to literal values special values can be used with the BINARY_FLOAT and BINARY_DOUBLE types. These values include:

- BINARY_FLOAT_NAN: Not a number
- BINARY_FLOAT_ INFINITY: Infinity
- BINARY_DOUBLE_NAN: Not a number
- BINARY_DOUBLE_INFINITY: Infinity

Example 5:
The following statements populate the table named **EMPEXP** using special values:

```
INSERT INTO EMPEXP (EMPNO, LASTSAL, EXPSAL)
  VALUES('3', BINARY_FLOAT_NAN, BINARY_DOUBLE_NAN);
INSERT INTO EMPEXP (EMPNO, LASTSAL, EXPSAL)
  VALUES('4', BINARY_FLOAT_INFINITY, BINARY_DOUBLE_INFINITY);
```

Example 6:
The following statement displays the contents of the table named **EMPEXP** (with Special values):

```
SELECT * FROM EMPEXP;
```

Output:

```
     EMPNO     LASTSAL     EXPSAL
---------- ---------- ----------
         1 2.327E+002 2.427E+002
         2 4.522E+002 4.709E+002
         3         Nan         Nan
         4         Inf         Inf
```

TIMESTAMP

The Oracle database has recently introduced the ability to store Timestamps. The advantages of a timestamp over a DATE are:

- It can store a fractional second
- It can store zero time

The structure of information stored in the TIMESTAMP is of the format:

```
YYYY-MM-DD HH24:MI:SS.SSSSSSSSS
```

where, **YYYY** is year, **MM** is the month, **DD** is the day, **HH24** is the hour, **MI** is the minutes, **SS.SSSSSSSSS** is the seconds and its fraction

Example 1:
The following statement creates a table named **EMPREGISTER** that contains TIMESTAMP columns:

```
CREATE TABLE EMPREGISTER (EMPNO Number(2), INTIME TIMESTAMP,
  OUTTIME TIMESTAMP);
```

Example 2:
The following statements populate the table named **EMPREGISTER:**

```
INSERT INTO EMPREGISTER (EMPNO, INTIME, OUTTIME)
  VALUES('1', TIMESTAMP '2005-01-23 08:35:55.1245',
     TIMESTAMP '2005-01-23 05:05:15.0');
INSERT INTO EMPREGISTER (EMPNO, INTIME, OUTTIME)
  VALUES('2', TIMESTAMP '2005-01-23 09:25:15.548',
     TIMESTAMP '2005-01-23 05:45:25.45');
```

Example 3:
The following statement describes the table named **EMPREGISTER:**

```
DESC EMPREGISTER;
```

Output:

```
Name       Null?    Type
---------- -------- ------------
EMPNO               NUMBER(2)
INTIME              TIMESTAMP(6)
OUTTIME             TIMESTAMP(6)
```

Example 4:
The following statement displays the contents of the table named **EMPREGISTER:**

```
SELECT * FROM EMPREGISTER;
```

Output:

```
EMPNO INTIME                          OUTTIME
----- ------------------------------ ------------------------------
    1 23-JAN-05 08.35.55.124500 AM 23-JAN-05 05.05.15.000000 AM
    2 23-JAN-05 09.25.15.548000 AM 23-JAN-05 05.45.25.450000 AM
```

Dump / Examine The Exact Content Of A Database Column

Example 1:
Extract the column details of the column **CAND_FIRST_NAME** from the **CANDIDATE** table.

Solution:

```
SELECT DUMP(CAND_FIRST_NAME) FROM CANDIDATE;
```

Output:

```
DUMP(CAND_FIRST_NAME)
--------------------------------------
Typ=1 Len=8: 83,104,97,114,97,110,97,109
Typ=1 Len=6: 72,97,110,115,101,108
Typ=1 Len=4: 73,118,97,110
Typ=1 Len=6: 67,104,97,97,121,97
Typ=1 Len=7: 83,104,114,97,118,97,110
```

In the above example, the type is **1**, which indicates the column is VARCHAR2, the **Len** indicates the length of the value held in the column for a particular record and values such as `83,104,97,114,97,110,97,109` indicate the ASCII code for the value held.

Restructuring Tables

A table's structure can be modified by using the **ALTER TABLE** command which can only be run on **an existing table**. With **ALTER TABLE** it is possible to **add** or **delete** columns, **create** or **destroy** indexes, **change the data type** of existing columns, **rename columns** or the **table** itself.

ALTER TABLE works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted and the new one is renamed. While **ALTER TABLE** is executing, the original table is still readable by users.

Updates and **writes** to the table are **stalled** until the new table is ready and then are automatically redirected to the new table **without any failed updates**.

REMINDER

To use **ALTER TABLE**, the **ALTER**, **INSERT** and **CREATE** privileges for the table are required.

Adding New Columns

Syntax:

```
ALTER TABLE <TableName>
  ADD(<NewColumnName> <Datatype> (<Size>)[,
      <NewColumnName> <Datatype> (<Size>), . . .]);
```

Example 24:
Enter a new field called **READING** in the table **CAND_LANG**.

```
ALTER TABLE CAND_LANG ADD (READING VarChar2(5));
```

Output:

```
Table altered.
```

Dropping A Column

Syntax:

```
ALTER TABLE <TableName> DROP COLUMN <ColumnName>;
```

Example 25:
Drop the column **READING** from the **CAND_LANG** table.

```
ALTER TABLE CAND_LANG DROP COLUMN READING;
```

Output:

```
Table altered.
```

Ways To Drop A Column From A Table

Prior to Oracle 8i dropping a column was **not possible** but there were **workarounds** to do this.

Example 26:
Drop the column **CAND_SURNAME** from the **CANDIDATE** table.

Solution 1:

```
UPDATE CANDIDATE SET CAND_SURNAME = NULL;
```

Output:

```
5 rows updated.
```

```
RENAME CANDIDATE TO CANDIDATEBASE;
```

Output:

```
Table renamed.
```

```
CREATE VIEW CANDIDATE AS
  SELECT CAND_USER_ID, CAND_CATEGORY, CAND_ID, CAND_FIRST_NAME,
      CAND_MIDDLE_NAME, CAND_FATHER_NAME, CAND_MARITAL_STATUS,
      CAND_BIRTHDATE, CAND_BIRTHPLACE, CAND_RELIGION, CAND_SAL_EXPCTD,
      CAND_PIC, IS_COMPLETE, CAND_EMAIL, CAND_TOT_EXP, CAND_UPD_DATE,
      RGST_COMPLETE FROM CANDIDATEBASE;
```

Output:

```
View created.
```

In the above example, to drop the column named **CAND_SURNAME** the following steps are carried out:

1. The value held in the **CAND_SURNAME** column is set to NULL for all the records of the **CANDIDATE** table
2. The table named **CANDIDATE** is then **renamed** to **CANDIDATEBASE**
3. Finally a view named **CANDIDATE** is created which comprises of all the columns except the column **CAND_SURNAME**

The users of the table **CANDIDATE**, while retrieving the data, will still use **CANDIDATE** as the name of the table and the data will be retrieved the same way, as it was, via a table even though a view is used.

Solution 2:

```
CREATE TABLE CANDIDATENEW AS
  SELECT CAND_USER_ID, CAND_CATEGORY, CAND_ID, CAND_FIRST_NAME,
      CAND_MIDDLE_NAME, CAND_FATHER_NAME, CAND_MARITAL_STATUS,
      CAND_BIRTHDATE, CAND_BIRTHPLACE, CAND_RELIGION, CAND_SAL_EXPCTD,
      CAND_PIC, IS_COMPLETE, CAND_EMAIL, CAND_TOT_EXP, CAND_UPD_DATE,
      RGST_COMPLETE FROM CANDIDATE;
```

Output:

```
Table created.
```

```
DROP TABLE CANDIDATE CASCADE CONSTRAINTS;
```

Output:

```
Table dropped.
```

```
RENAME CANDIDATENEW TO CANDIDATE;
```

Output:

```
Table renamed.
```

In the above example, to drop the column named **CAND_SURNAME** the following steps are carried out:

1. A table named **CANDIDATENEW is created** comprising of all the columns except the column **CAND_SURNAME**
2. The table named **CANDIDATE** is now **dropped**
3. Finally the table just created i.e. **CANDIDATENEW is renamed** to **CANDIDATE**

The table **CANDIDATE** is referenced by a FOREIGN KEY constraint (i.e. via its primary key **CAND_ID**, it is referenced by other tables). The Oracle engine will not allow dropping the table. Using **CASCADE CONSTRAINTS** solves this problem. The **CASCADE CONSTRAINTS** option drops the FOREIGN KEY constraints of the child tables.

From Oracle8 onwards, dropping of columns can be done, by using **ALTER TABLE** command.

```
ALTER TABLE CANDIDATE DROP COLUMN CAND_SURNAME;
```

Output:

```
Table altered.
```

Here, the column is dropped directly using the **ALTER TABLE** command.

```
ALTER TABLE CANDIDATE SET UNUSED COLUMN CAND_SURNAME;
```

Output:

```
Table altered.
```

The above command can be verified as:
Log in as SYS user:

```
CONNECT sys/<password>@<service_name> AS SYSDBA;
```

Retrieve the unused columns:

```
SELECT * FROM SYS.DBA_UNUSED_COL_TABS;
```

Output:

```
OWNER                TABLE_NAME                COUNT
-------------------- ------------------------- ----------
DBA_PM               CANDIDATE                          1
```

Log in as DBA_PM user:

```
CONNECT DBA_PM/<password>@<service_name>;
```

```
ALTER TABLE Candidate DROP UNUSED COLUMNS;
```

Output:

```
Table altered.
```

Log in as SYS user:

```
CONNECT sys/<password>@<service_name> AS SYSDBA;
```

Retrieve the unused columns:

```
SELECT * FROM SYS.DBA_UNUSED_COL_TABS;
```

Output:

```
no rows selected
```

In this example, the column **CAND_SURNAME** is set to represent itself as an unused column. To verify the same a SELECT is fired which displays the output as shown above.

Finally, using the **ALTER TABLE command,** all those columns marked as unused are dropped and this is further verified by issuing the select command on the **SYS.DBA_UNUSED_COL_TABS table.**

Modifying Existing Columns

Syntax:

```
ALTER TABLE <TableName>
  MODIFY (<ColumnName> <NewDatatype>(<NewSize>));
```

Example 27:
Alter the **CANDIDATE** table to allow the **CAND_FIRST_NAME** field to hold maximum of 50 characters.

```
ALTER TABLE CANDIDATE MODIFY (CAND_FIRST_NAME VarChar2(50));
```

Output:

```
Table altered.
```

Restrictions On Restructuring Tables

The following tasks **cannot** be performed when using the **ALTER TABLE** clause:

- Change the name of the table
- Change the name of the column
- Decrease the size of a column if table data exists

Ways To Rename A Column In A Table

Example 28:
Rename the column named **CAND_FIRST_NAME** to **CANDIDATENAME** from the table **CANDIDATE**.

Solution 1:

```
RENAME CANDIDATE TO CANDIDATEBASE;
```

Output:

```
Table renamed.
```

```
CREATE VIEW CANDIDATE AS
  SELECT CAND_USER_ID, CAND_CATEGORY, CAND_ID,
  CAND_FIRST_NAME "CandidateName", CAND_MIDDLE_NAME, CAND_SURNAME,
  CAND_FATHER_NAME, CAND_MARITAL_STATUS, CAND_BIRTHDATE,
  CAND_BIRTHPLACE, CAND_RELIGION, CAND_SAL_EXPCTD, CAND_PIC,
  IS_COMPLETE, CAND_EMAIL, CAND_TOT_EXP, CAND_UPD_DATE, RGST_COMPLETE
      FROM CANDIDATEBASE;
```

Output:

```
View created.
```

In the above example,

1. The table **CANDIDATE** is first renamed to **CANDIDATEBASE**
2. A view named **CANDIDATE** is created, by specifying the new column names from the table **CANDIDATEBASE**

Solution 2:

```
CREATE TABLE CANDIDATENEW AS
  SELECT CAND_USER_ID, CAND_CATEGORY, CAND_ID,
  CAND_FIRST_NAME "CandidateName", CAND_MIDDLE_NAME, CAND_SURNAME,
  CAND_FATHER_NAME, CAND_MARITAL_STATUS, CAND_BIRTHDATE,
  CAND_BIRTHPLACE, CAND_RELIGION, CAND_SAL_EXPCTD, CAND_PIC,
  IS_COMPLETE, CAND_EMAIL, CAND_TOT_EXP, CAND_UPD_DATE, RGST_COMPLETE
      FROM CANDIDATE;
```

Output:

```
Table created.
```

```
DROP TABLE CANDIDATE CASCADE CONSTRAINTS;
```

Output:

```
Table dropped.
```

```
RENAME CANDIDATENEW TO CANDIDATE;
```

Output:

```
Table renamed.
```

In the above example,

1. A table named **CANDIDATENEW** is created comprising of columns with new names from the old table **CANDIDATE**
2. The old table is then dropped
3. Finally, the newly created table is renamed to the old table name i.e. **CANDIDATE**

Solution 3:

```
ALTER TABLE CANDIDATE ADD (CANDIDATENAME VARCHAR2(25));
```

Output:

```
Table altered.
```

```
UPDATE CANDIDATE SET CANDIDATENAME = CAND_FIRST_NAME;
```

Output:

```
5 rows updated.
```

```
ALTER TABLE CANDIDATE DROP COLUMN CAND_FIRST_NAME;
```

Output:

```
Table altered.
```

In the above example,

1. The table **CANDIDATE** is altered and a new column is added to represent the new name of the column to be renamed
2. The table **CANDIDATE** is updated by copying the data from the old column to the new column
3. Finally, the old column is dropped

Renaming Tables

Oracle allows renaming of tables. The rename operation is done **atomically**, which means that **no other thread** can access the table /s while a rename process is running.

REMINDER

To rename a table the ALTER and DROP privileges on the original table and the CREATE and INSERT privileges on the new table are required .

The syntax to rename a table is described as:

Syntax:

```
RENAME <TableName> TO <NewTableName>;
```

Example 29:
Change the name of **USERS** table to **LOGIN_INFO** table

```
RENAME USERS TO LOGIN_INFO;
```

Output:

```
Table renamed.
```

Truncating Tables

TRUNCATE TABLE empties a table completely. Logically, this is equivalent to a **DELETE** statement that deletes all rows, but there are practical differences under some circumstances.

TRUNCATE TABLE differs from **DELETE** in the following ways:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one
- Truncate operations are not transaction-safe (i.e. an error will occur if an active transaction or an active table lock exists)
- The number of deleted rows are not returned

Syntax:

```
TRUNCATE TABLE <TableName>;
```

Example 30:
Truncate the table **LANGUAGE**

```
TRUNCATE TABLE LANGUAGE;
```

Output:

```
Table truncated.
```

Destroying Tables

Sometimes tables within a particular database become obsolete and need to be discarded. In such situation using the DROP TABLE statement with the table name can destroy a specific table.

Syntax:

```
DROP TABLE <TableName>;
```

WARNING

If a table is dropped all records held within it are lost and cannot be recovered.

Example 31:
Remove the table **COUNTRY** along with the data held.

```
DROP TABLE COUNTRY;
```

Output:

```
Table dropped.
```

Constraining Table Data

The business world runs on its data being gathered, validated, stored and finally analyzed. Business managers determine a set of business rules that must be applied to the data being gathered **prior** it being stored in the database/table to guarantee its integrity.

An example of a business rule would be, no employee in the sales department can have a salary of less than Rs.1000/-.

Such rules have to be enforced on data being gathered for storage. Only data, which satisfies these rules, should be stored for future business analysis. If the data gathered fails to satisfy the business rule, it must be rejected and an error displayed. This ensures that the data stored in the table is valid, has integrity and business decisions can be made with confidence based on the data.

Business rules applied to data are completely **Commercial Application dependent**. The rules applied to data gathered and processed by a **Savings Bank application** will be very different to the business rules applied to data gathered and processed by an **Inventory application**, which in turn will be very different to the business rules applied to data gathered and processed by the **Personnel Management application**.

Business rules enforced on data being stored in a table column are called **Constraints**. Constraints **super control** the data being entered into a table prior its permanent storage.

To help develop and clarify the concept of data constraints, several tables will be created and different types of constraints will be applied to the table columns **or** the table itself. The set of tables are described below. Appropriate examples of data constraints are bound to these tables.

Applying Data Constraints

Oracle permits data constraint definitions to be attached to table columns via SQL syntax. Hence, just prior the db engine inserting data into the table column, it recognizes the existence of a data constraint. The db engine then applies the data constraint definition to the data just about to be inserted in the column. This checks the data for integrity prior storage. If the data passes this check, it is stored in the table column, else the data is rejected. Even if a single column of the record being entered into the table fails a constraint, the **entire record is rejected and not stored in the table**. Both the **Create Table** and **Alter Table** SQL verbs can be used to write SQL sentences that attach constraints (i.e. Business / System rules) to a table column.

WARNING

Until now tables created in this material have **not** had any data constraints attached to their table columns. Hence the tables have **not** been given any instructions to filter what is being stored in the table. This situation **can** and **does** result in erroneous data being stored in the table.

Once a constraint is attached to a table column, any SQL **INSERT** or **UPDATE** statement automatically causes these constraints to be applied to data prior it being inserted into the table column for storage.

REMINDER

Oracle also permits applying data constraints at **Table level**. More on table level constraints later in this material.

Type Of Data Constraints

There are two types of data constraints that can be applied to data being inserted into an Oracle table. One type of constraint is called an **I/O** constraint (**I**nput / **O**utput). This data constraint determines the speed at which data can be inserted or extracted from an Oracle table. The other type of constraint is called a **Business Rule** constraint.

I/O Constraints

The input/output data constraints are further divided into **two** distinctly different constraints (**i.e.** Primary Key and Foreign Key constraints)

What Is A Primary Key

A primary key is the value held in one or more columns in a table, which is used to uniquely identify **a single row** in the table. NULL cannot be used as such a value. A table can have only one primary key. A **primary key column** in a table has the following special attributes:

- It defines the column, as a **mandatory** column (**i.e.** the column cannot be left blank). Its NOT NULL attribute is set to active. This prevents a NULL value from being stored in the column.
- The data held in the column, right across the entire column, MUST be UNIQUE. (**i.e.** Duplicate values are not allowed right across the column)

A single column primary key is called a **Simple** key. A multicolumn primary key is called a **Composite** primary key. The only function of a primary key in a table is to **uniquely identify each individual row** in the table. When a row (**i.e.** a data record) cannot be uniquely identified using a single value held in a simple key, a composite key must be defined. A primary key can be defined in either the **CREATE TABLE** statement or **ALTER TABLE** statement.

For example, **CAND_LANG** table will hold multiple records in form of languages spoken/written by a candidate. Each candidate will know multiple languages. Standard business rules do not allow multiple entries for the language. However, multiple languages will definitely have multiple entries for the same candidate.

Under these circumstances, the only way to uniquely identify a row in the **CAND_LANG** table is via a composite primary key, consisting of **CAND_ID** and **LANG_ID**. Thus the combination of candidate identity and language identity will uniquely identify each row.

Features of Primary key

1. The Primary key is a column **or** a set of columns that uniquely identifies each row in the table
2. The Primary key will not allow duplicate or NULL values
3. A Primary key is not compulsory but it is recommended
4. A Primary key helps in identifying one record from another record. It is also used in relating tables with one another in conjunction with a Foreign Key
5. The Primary key cannot be LONG or LONG RAW data type
6. Only one Primary key is allowed per table
7. A Unique Index is created automatically if there is a Primary key
8. A table can have upto 16 columns in a Composite Primary key

PRIMARY KEY Constraint Defined At Column Level

Syntax:

```
<ColumnName> <Datatype>(<Size>) PRIMARY KEY
```

Example 1:
Drop the **COUNTRY** table, if it already exists. Create a table **COUNTRY** such that the contents of the column **COUNTRY_ID** is unique and not null.

```
DROP TABLE COUNTRY;
CREATE TABLE COUNTRY(COUNTRY_ID Number(2) PRIMARY KEY,
  COUNTRY_NAME VarChar2(35));
```

Output:

```
Table created.
```

For testing purpose, execute the following **INSERT INTO** statement:

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES (1, 'India');
```

Output:

```
1 row created.
```

To verify whether the Primary Key Constraint is functional, reissue the same **INSERT INTO** statement. The result is the following error:

Output:

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES (1,
'India')
*
ERROR at line 1:
ORA-00001: unique constraint (DBA_PM.SYS_C005427) violated
```

PRIMARY KEY Constraint Defined At Table Level

Syntax:

```
PRIMARY KEY (<ColumnName>[, <ColumnName>])
```

Example 2:
Drop the **CAND_LANG** table, if it already exists. Create a table **CAND_LANG** where there is a composite primary key mapped to the columns **LANG_ID** and **CAND_ID**.
Since this constraint spans across columns, **it must be described at table level.**

```
DROP TABLE CAND_LANG;
CREATE TABLE CAND_LANG (LANG_ID Number(2), CAND_ID VarChar2(7),
  SPOKEN VarChar2(5), WRITTEN VarChar2(5),
  PRIMARY KEY(LANG_ID, CAND_ID));
```

Output:

```
Table created.
```

For testing purpose, execute the following **INSERT INTO** statement:

```
INSERT INTO CAND_LANG (LANG_ID, CAND_ID, SPOKEN, WRITTEN)
  VALUES(1, 'CD00001', 'Good', 'Fair');
```

Output:

```
1 row created.
```

To verify whether the Composite Primary Key Constraint is functional, reissue the same **INSERT INTO** statement.

The result is the following error:

Output:

```
INSERT INTO CAND_LNG (LANG_ID, CAND_ID, SPOKEN, WRITTEN)
*
ERROR at line 1:
ORA-00001: unique constraint (DBA_PM.SYS_C005428) violated
```

Now, simply modify the INSERT INTO statement as shown below, to allow the record to pass the composite primary key constraint:

```
INSERT INTO CAND_LANG (LANG_ID, CAND_ID, SPOKEN, WRITTEN)
  VALUES(1, 'CD00002', 'Good', 'Fair');
```

Output:

```
1 row created.
```

What is a Foreign Key

A Foreign key is a column whose values exclusively represent a relationship between tables. A foreign key is a column (or a group of columns) whose values are derived from the **primary key** or **unique key** of some other table, which binds the table in which it is defined to the table from which the F_Key column value is derived.

The table in which the foreign key column is defined is called a **Foreign table** or **Detail table**. The table that defines the **primary** or **unique** key and is referenced by the **foreign key** is called the **Primary table** or **Master table**. A Foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement

The Master table's Primary Key column can be referenced in the Foreign key definition by using the clause **REFERENCES TableName.ColumnName** when defining the foreign key column and its attributes in the detail table.

Features of Foreign Keys

1. Foreign key is a column that references a column of a table. The referenced column can be in a different table or in the same table
2. The Master table being referenced has to have a Primary key or Unique index.
3. The Foreign Key in the Detail table can hold multiple duplicate values.
4. The Foreign key constraint must be specified at the Detail table not at the Master table

This constraint establishes a relationship between records (i.e. column data) across a Master and a Detail table. This relationship ensures:

- Records cannot be **inserted** into a **detail table** if corresponding records in the master table do not exist
- Records of the **master table** cannot be **deleted** if corresponding records in the detail table actually exist

FOREIGN KEY Constraint Defined At The Column Level

Syntax:

```
<ColumnName> <DataType>(<Size>)
    REFERENCES <TableName> [(<ColumnName>)] [ON DELETE CASCADE]
```

REMINDER

Example 3 and Example 4 use **PERM_USER** as the Foreign Key which references **USER_ID** in the **PERMISSIONS** table. For these examples to go through the **USERS** table needs to be recreated with **USER_ID** as a Primary Key. This can be done as follows:

```
DROP TABLE USERS;
CREATE TABLE USERS (USER_ID VarChar2(7),
    USER_USERNAME VarChar2(15), USER_PASSWORD VarChar2(20),
    USER_TYPE VarChar2(5), FOR_PASS_QUESTION VarChar2(100),
    FOR_PASS_ANSWER VarChar2(50), PRIMARY KEY (USER_ID));
```

Example 3:
Drop the table **PERMISSIONS**, if it already exists. Create a table **PERMISSIONS**, with its primary as **PERM_ID** and **PERM_USER** referencing the foreign key **USER_ID** in the **USERS** table.

```
DROP TABLE PERMISSIONS;
CREATE TABLE PERMISSIONS (PERM_USER VarChar2(7) REFERENCES USERS,
  PERM_ID VarChar2(7), PERM_GRANT_ON VarChar2(35), PERM_VALUE Number(2),
  PRIMARY KEY (PERM_ID));
```

Output:

```
Table created.
```

The **REFERENCES** key word points to the table **USERS**. *The table **USERS** must have the column **USER_ID** as its primary key column*. Since no column is specified in the foreign key definition, Oracle applies an automatic (default) link to the primary key column i.e. **USER_ID** of the table **USERS**.

The foreign key definition is specified as:

```
USER_ID VarChar2(7) REFERENCES USERS
```

FOREIGN KEY Constraint Defined At The Table Level

Syntax:

```
FOREIGN KEY (<ColumnName> [,<ColumnName>] )
    REFERENCES <TableName> [(<ColumnName>, <ColumnName>)]
```

Example 4:
Drop the table **PERMISSIONS**, if it already exists. Create a table **PERMISSIONS** with **PERM_USER** as foreign key referencing column **USER_ID** in the **USERS** table

```
DROP TABLE PERMISSIONS;
CREATE TABLE PERMISSIONS (PERM_USER VarChar2(7),   PERM_ID VarChar2(7),
  PERM_GRANT_ON VarChar2(35), PERM_VALUE Number(2),
  PRIMARY KEY (PERM_ID),
  FOREIGN KEY(PERM_USER) REFERENCES USERS(USER_ID));
```

Output:

```
Table created.
```

FOREIGN KEY Constraint Defined With ON DELETE CASCADE

Example 5:
Drop the tables **PERMISSIONS** and **USERS**, if they already exist. Create a table **USERS** with the field **USER_ID** as its primary key.

Create a table **PERMISSIONS** with its foreign key as **PERM_USER** with the **ON DELETE CASCADE** option. The foreign key references **USER_ID**, which is available as a primary key column in the **USERS** table.

Insert some records into both the tables.

```
DROP TABLE PERMISSIONS;
DROP TABLE USERS;
CREATE TABLE USERS (USER_ID VarChar2(7), USER_USERNAME VarChar2(15),
  USER_PASSWORD VarChar2(20), USER_TYPE VarChar2(5),
  FOR_PASS_QUESTION VarChar2(100), FOR_PASS_ANSWER VarChar2(50),
  PRIMARY KEY (USER_ID));
```

Output:

```
Table created.
```

```
CREATE TABLE PERMISSIONS (PERM_USER VarChar2(7), PERM_ID VarChar2(7),
  PERM_GRANT_ON VarChar2(35), PERM_VALUE Number(2),
  PRIMARY KEY (PERM_ID),
  FOREIGN KEY(PERM_USER) REFERENCES USERS(USER_ID)
    ON DELETE CASCADE);
```

Output:

```
Table created.
```

Insert data into the USERS and PERMISSIONS tables. Now delete a record from the USERS table as:

```
DELETE FROM USERS WHERE USER_ID='US00001';
```

Output:

```
1 row deleted.
```

Query the table PERMISSIONS for records:

```
SELECT * FROM PERMISSIONS;
```

Notice the deletion of the records in the **PERMISSIONS** table belonging to user with **PERM_USER** US00001.

Explanation:
In this example, a primary key is created in the **USERS** table. i.e. **USRS_ID** field. Then, a foreign key (**PERM_USER**) is created in the **PERMISSIONS** table that references the **USERS** table based on the contents of the **USER_ID** field.

Because of the **CASCADE DELETE** option, when a record in the **USERS** table is deleted, all records in the **PERMISSIONS** table, having matching values in **USER_ID** and **PERM_USER** columns, will also be deleted.

FOREIGN KEY Constraint Defined With ON DELETE SET NULL:

A FOREIGN key with a **SET NULL ON DELETE** means that if a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to **NULL**. The records in the child table **will not be deleted**.

A FOREIGN key with a **SET NULL ON DELETE** can be defined in either a **CREATE TABLE** statement or an **ALTER TABLE** statement.

Example 6:
Drop the tables **PERMISSIONS** and **USERS**, if they already exist. Create a table **USERS** with the field **USER_ID** as its primary key.

Create a table **PERMISSIONS** with its foreign key as **PERM_USER** with the **ON DELETE SET NULL** option. The foreign key references **USER_ID**, which is available as a primary key column in the **USERS** table.

Insert some records into both the tables.

```
DROP TABLE PERMISSIONS;
DROP TABLE USERS;
CREATE TABLE USERS (USER_ID VarChar2(7), USER_USERNAME VarChar2(15),
  USER_PASSWORD VarChar2(20), USER_TYPE VarChar2(5),
  FOR_PASS_QUESTION VarChar2(100), FOR_PASS_ANSWER VarChar2(50),
  PRIMARY KEY (USER_ID));
```

Output:

```
Table created.
```

```
CREATE TABLE PERMISSIONS (PERM_USER VarChar2(7), PERM_ID VarChar2(7),
  PERM_GRANT_ON VarChar2(35), PERM_VALUE Number(2),
  PRIMARY KEY (PERM_ID),
  FOREIGN KEY(PERM_USER) REFERENCES USERS(USER_ID)
     ON DELETE SET NULL);
```

Output:

```
Table created.
```

Insert data into the USERS and PERMISSIONS tables. Now delete a record from the **Customers** table as:

```
DELETE FROM USERS WHERE USER_ID='US00001';
```

Output:

```
1 row deleted.
```

Query the table PERMISSIONS for records:

```
SELECT * FROM PERMISSIONS;
```

Notice that value held in the **PERM_USER** field of the **PERMISSIONS** table is set to null, for those records belonging to user with **USER_ID** US00001.

Explanation:
In this example, a primary key is created in the **USERS** table. i.e. **USER_ID** field. Then, a foreign key (**PERM_USER**) is created in the **PERMISSIONS** table that references the **USERS** table based on the **USER_ID** field.

Because of the **CASCADE SET NULL**, when a record in the **USERS** table is deleted, all corresponding records in the **PERMISSIONS** table, having matching values in **USER_ID** and **PERM_USER** columns, will have the **PERM_USER** values set to **null**.

Rejects a **DELETE** from the Master table if **corresponding records** in the DETAIL table **exist**

- Must reference a **PRIMARY KEY** or **UNIQUE** column(s) in primary table
- Requires that the **FOREIGN KEY** column(s) and the **CONSTRAINT** column(s) have **matching** data types
- Can reference the same table named in the **CREATE TABLE** statement

How Insert And Update Operations Work On A Table With A Foreign Key

The existence of a foreign key implies that the table with the foreign key is **related** to the master table from which the foreign key is derived. A foreign key value must have a **corresponding** primary key or unique key value in the master table.

For example a personnel information system includes two tables (i.e. department and employee). An employee cannot belong to a department that does not exist. Thus the department number specified in the employee table must be present in the department table.

How A Delete Operation Works On A Table Bound Another By A Foreign Key

Oracle **displays an error message** when a record in the master table is deleted and corresponding records exists in a detail table and **prevents the delete operation from going through**.

REMINDER

This **default** behavior of the foreign key can be changed, by using the **ON DELETE CASCADE** option. When the **ON DELETE CASCADE** option is specified in the foreign key definition, if a record is deleted in the master table, all corresponding records in the detail table will be deleted when a record in the master table is deleted.

Principles of **Foreign Key/References** constraint:

- Rejects an **INSERT** or **UPDATE** of a value in the foreign key table, if a corresponding value does not currently exist in the master key table
- If the **ON DELETE CASCADE** option is set, a **DELETE** operation in the master table will trigger a **DELETE** operation for corresponding records **in all** detail tables
- If the **ON DELETE SET NULL** option is set, a **DELETE** operation in the master table will set the value held by the foreign key of the detail tables to null

The Unique Key Constraint

The **Unique** column constraint **permits multiple entries** of NULL into the column. These NULL values are clubbed at the top of the column in the order in which they were entered into the table. This is **the essential difference** between the Primary Key and the Unique constraints when applied to table column(s).

Key points about the Unique Constraint:

1. A Unique key will not allow duplicate values
2. A Unique index is created automatically on any table column which has a Unique constraint attached
3. A table can have more than one Unique key
4. A Unique key can combine upto 16 columns in a Composite Unique key
5. A Unique key cannot be the LONG or LONG RAW data type

UNIQUE Constraint Defined At The Column Level

Syntax:

```
<ColumnName> <Datatype>(<Size>) UNIQUE
```

Example 7:
Drop the **CAND_DOC** table, if it already exists. Create a table **CAND_DOC** such that the contents of the column **CAND_DOC_NO** are unique across the entire column.

```
DROP TABLE CAND_DOC;
CREATE TABLE CAND_DOC (CAND_ID VarChar2(7),
  CAND_DOC_NO VarChar2(20) UNIQUE,
  CAND_DOC_NAME VarChar2(16), CAND_DOC_TYPE VarChar2(20),
  CAND_DOC_ISSUE_DAT VarChar2(12), CAND_DOC_EXP_DAT VarChar2(12),
  CAND_DOC_ISSUE_PLACE VarChar2(35));
```

Output:

```
Table created.
```

For testing the Unique constraint execute the following INSERT INTO statements:

```
INSERT INTO CAND_DOC VALUES ('CD00001', 'E3097703', 'Passport', NULL,
  '01/11/2002', '30/10/2012', 'Trivandrum');
INSERT INTO CAND_DOC VALUES ('CD00001', 'E3097703', 'Driving License',
  'Light Vehicles', '07/09/1999', '06/08/2009', 'Trivandrum');
INSERT INTO CAND_DOC VALUES ('CD00001', 'T2110/99', 'Driving License',
  'Light Vehicles', '07/09/1999', '06/08/2009', 'Trivandrum');
```

Output:
The first **INSERT INTO** statement will execute without any errors as show below:

```
1 row created.
```

When the second **INSERT INTO** statement is executed an errors occurs as show below:

```
INSERT INTO CAND_DOC VALUES ('CD00001', 'E3097703', 'Driving
License', 'Light Vehicles', '07/09/1999', '06/08/2009',
'Trivandrum')
*
ERROR at line 1:
ORA-00001: unique constraint (DBA_PM.SYS_C005446) violated
```

The third **INSERT INTO** statement rectifies this and the result is as show below:

```
1 row created.
```

When a **SELECT** statement is executed on the Cars table the records retrieved are:

```
SELECT CAND_ID, CAND_DOC_NO, CAND_DOC_NAME FROM CAND_DOC;
```

Output:

```
CAND_ID CAND_DOC_NO          CAND_DOC_NAME
------- -------------------- ----------------
CD00001 E3097703             Passport
CD00001 T2110/99             Driving License
```

UNIQUE Constraint Defined At The Table Level

Syntax:

```
CREATE TABLE <TableName> (<ColumnName1> <Datatype>(<Size>),
  <ColumnName2> <Datatype>(<Size>)[, . . .],
  UNIQUE (<ColumnName1>, <ColumnName2>));
```

Example 8:
Drop the **CAND_DOC** table, if it already exists. Create a table **CAND_DOC** such that the contents of the column **CAND_DOC_NO** are unique across the entire column.

```
DROP TABLE CAND_DOC;
CREATE TABLE CAND_DOC (CAND_ID VarChar2(7), CAND_DOC_NO VarChar2(20),
  CAND_DOC_NAME VarChar2(16), CAND_DOC_TYPE VarChar2(20),
  CAND_DOC_ISSUE_DAT VarChar2(12), CAND_DOC_EXP_DAT VarChar2(12),
  CAND_DOC_ISSUE_PLACE VarChar2(35), UNIQUE (CAND_DOC_NO));
```

Output:

```
Table created.
```

In the case of the table level unique constraints, the result for the following INSERT INTO statement will remain the same as explained earlier.

```
INSERT INTO CAND_DOC VALUES ('CD00001', 'E3097703', 'Passport', NULL,
  '01/11/2002', '30/10/2012', 'Trivandrum');
INSERT INTO CAND_DOC VALUES ('CD00001', 'E3097703', 'Driving License',
  'Light Vehicles', '07/09/1999', '06/08/2009', 'Trivandrum');
INSERT INTO CAND_DOC VALUES ('CD00001', 'T2110/99', 'Driving License',
  'Light Vehicles', '07/09/1999', '06/08/2009', 'Trivandrum');
```

NULL Value Concepts

Often there may be records in a table that do not have values for every field. This could be because the information is not available at the time of data entry or because the field is not applicable in every case. If the column was created as **NULLABLE**, Oracle will place a NULL value in the column in the **absence** of a user-defined value.

The difference between a NULL value from a blank or a zero is that a **NULL value** can be inserted into **columns** of **any data type**.

Principles Of NULL Values

- Setting a NULL value is appropriate when the actual value is unknown or when a value would not be meaningful
- A NULL value is **not equivalent** to a value of **zero** if the data type is **number** and is not equivalent to **spaces** if the data type is **character**
- A NULL value will evaluate to NULL in any expression (**e.g.** NULL multiplied by 10 is NULL)
- NULL value can be inserted into columns of **any data type**
- If the column has a NULL value, Oracle **ignores** any UNIQUE, FOREIGN KEY, CHECK constraints that may be attached to the column

Difference Between An Empty String And A NULL Value

Oracle has changed its rules about empty strings and null values in newer versions of Oracle. Now, an empty string is treated as a null value in Oracle.

To understand this, go through the following example:

Example 9:
Drop the table **COUNTRY**, if it already exists. Create a table **COUNTRY** with its primary as **COUNTRY_ID**.

```
DROP TABLE COUNTRY;
CREATE TABLE COUNTRY (COUNTRY_ID Number(2) PRIMARY KEY,
  COUNTRY_NAME VarChar2(35));
```

Insert two records into the COUNTRY table.

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES(1, NULL);
```

Output:

```
1 row created.
```

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES(2, '');
```

Output:

```
1 row created.
```

The first statement inserts a record with a COUNTRY_NAME that is NULL, while the second statement inserts a record with an empty string as a country name.

Now, retrieve all rows with a COUNTRY_NAME that is an empty string value as follows:

```
SELECT * FROM COUNTRY WHERE COUNTRY_NAME = '';
```

When this statement is executed, it is expected to retrieve the row that was inserted above. But instead, this statement will not retrieve any records at all.

Now, try retrieving all rows where the country name contains a NULL value:

```
SELECT * FROM COUNTRY WHERE COUNTRY_NAME IS NULL;
```

When this statement is executed, both rows are retrieved. This is because Oracle has now changed its rules so that empty strings behave as null values.

It is also important to note that the null value is unique. Usual operands such as =, <, > and so on cannot be used on a null value. Instead, the IS NULL and IS NOT NULL conditions have to be used.

NOT NULL Constraint Defined At The Column Level

In addition to Primary key and Foreign Key, Oracle has **NOT NULL** as column constraint. The **NOT NULL** column constraint ensures that a table column cannot be left empty.

When a column is defined as **NOT NULL**, it becomes a **mandatory** column. Hence, a value must be entered into the column if the record is to be stored in the table else the record is rejected and not stored.

Syntax:

```
<ColumnName> <Datatype>(<Size>) NOT NULL
```

Example 10:
Drop the table **COUNTRY**, if it already exists and then create it again making the country name field **NOT NULL**.

```
DROP TABLE COUNTRY;
CREATE TABLE COUNTRY (COUNTRY_ID Number(2) PRIMARY KEY,
  COUNTRY_NAME VarChar2(35) NOT NULL);
```

Output:

```
Table created.
```

REMINDER

The **NOT NULL** constraint can only be applied at column level.

Execute the following INSERT INTO statements to verify whether mandatory field constraints are applied**:**

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES(1, NULL);
```

Output:

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES(1, NULL)
                                                      *
ERROR at line 1:
ORA-01400: cannot insert NULL into
("DBA_PM"."COUNTRY"."COUNTRY_NAME")
```

The above **error** message confirms that the mandatory field constraints are applied successfully.

WARNING

The **NOT NULL** constraint can only be applied at column level. Although **NOT NULL** can be applied as a **CHECK** constraint, Oracle Corp recommends that this should **not be done**.

Assigning User Defined Names To Constraints

At the time of constraint definition, Oracle assigns a **unique name** to each constraint. The convention used by Oracle is**:**

```
SYS_Cn
```

where, **n** is a **numeric value** that makes the constraint name **unique.**

Constraints can be given a unique user-defined name along with the constraint definition. A constraint can then be dropped by referring to the constraint by its name. When dropping a user-defined constraint its name is used to reference it.

User named constraints simplifies the task of dropping constraints. A constraint can be given a user-defined name by preceding the constraint definition with the reserved word **CONSTRAINT** and a **user-defined name**.

Syntax:

```
CONSTRAINT <Constraint Name> <Constraint Definition>
```

Example 11:
Drop the **USERS** table, if it already exists. Create a table **USERS** with a primary key constraint on the column **USER_ID** and also define its constraint name.

```
DROP TABLE USERS;
CREATE TABLE USERS (
  USER_ID VarChar2(7) CONSTRAINT PK_USERS_USERID PRIMARY KEY,
  USER_USERNAME VarChar2(15), USER_PASSWORD VarChar2(20),
  USER_TYPE VarChar2(5), FOR_PASS_QUESTION VarChar2(100),
  FOR_PASS_ANSWER VarChar2(50));
```

Output:

```
Table created.
```

Example 12:
Drop the table **PERMISSIONS**, if it already exists. Create a table **PERMISSIONS** with its foreign key as **PERM_USER**. *The foreign key is **PERM_USER** which references **USER_ID** and **should be** available as a primary key in the **USERS** table.* Also define the name of the foreign key.

```
DROP TABLE PERMISSIONS;
CREATE TABLE PERMISSIONS (PERM_USER VarChar2(7),   PERM_ID VarChar2(7),
  PERM_GRANT_ON VarChar2(35), PERM_VALUE Number(2),
  CONSTRAINT FK_PERM_USERID FOREIGN KEY(PERM_USER)
     REFERENCES USERS(USER_ID));
```

Output:

```
Table created.
```

The CHECK Constraint

Business Rule validations can be applied to a table column by using the **CHECK** constraint. **CHECK** constraints must be specified as a logical expression that evaluates either to **TRUE** or **FALSE**.

REMINDER

A **CHECK** constraint takes substantially longer to execute as compared to NOT NULL, PRIMARY KEY, FOREIGN KEY or UNIQUE. Thus CHECK constraints must be avoided, if the constraint can be defined using the Not Null, Primary key or Foreign key constraint.

CHECK Constraint Defined At The Column Level

Syntax:

```
<ColumnName> <Datatype>(<Size>) CHECK (<Logical Expression>)
```

Example 13:
Drop the table **COUNTRY**, if already exists. Create a table **COUNTRY** with the check constraints such that, the data values being inserted into the column **COUNTRY_NAME** should be in **upper case** only.

```
DROP TABLE COUNTRY;
CREATE TABLE COUNTRY (COUNTRY_ID Number(2), COUNTRY_NAME VarChar2(35)
  CHECK (COUNTRY_NAME=UPPER(COUNTRY_NAME)));
```

Output:

```
Table created.
```

CHECK Constraint Defined At The Table Level

Syntax:

```
CHECK (<Logical Expression>)
```

Example 14:
Drop the table **COUNTRY**, if already exists. Create a table **COUNTRY** with the check constraints such that, the data values being inserted into the column **COUNTRY_NAME** should be in **upper case** only.

```
DROP TABLE COUNTRY;
CREATE TABLE COUNTRY (COUNTRY_ID Number(2), COUNTRY_NAME VarChar2(35),
  CHECK (COUNTRY_NAME=UPPER(COUNTRY_NAME)));
```

Output:

```
Table created.
```

Execute the following INSERT INTO statements to verify whether check constraints are applied:

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES (1, 'India');
```

Output:

```
INSERT INTO COUNTRY (COUNTRY_ID, COUNTRY_NAME) VALUES (1,
'India')
*
ERROR at line 1:
ORA-02290: check constraint (DBA_PM.SYS_C005454) violated
```

The above **error** messages confirm that the check constraints defined are applied successfully.

When using **CHECK** constraints, consider the ANSI / ISO standard, which states that a CHECK constraint is violated only If the condition evaluates to **False**. A check constraint is not violated if the condition evaluates to **True**.

REMINDER

If the expression in a check constraint does not return a **true / false**, the value is **Indeterminate** or **Unknown**. Unknown values do not violate a check constraint condition. For example, consider the following CHECK constraint for Total column in the Bills table:

CHECK (CAND_SAL_EXPCTD > 0)

At first glance, this rule may be interpreted as "do not allow a row in the **Candidate** table unless the **CAND_SAL_EXPCTD** is **greater than** 0". However, note that if a row is inserted with a **NULL CAND_SAL_EXPCTD**, the row **does not violate** the CHECK constraint because the entire check condition is evaluated as **unknown**.

In this particular case, prevent such violations by placing the **NOT NULL** integrity constraint along with the check constraint on **CAND_SAL_EXPCTD** column of the table **Candidate**.

Restrictions On CHECK Constraints

A **CHECK** integrity constraint requires that its condition resolve to **true** or **unknown** for the row to be processed. If an SQL statement causes the condition to evaluate to **false**, an appropriate error message is displayed and processing stops.

A **CHECK** constraint has the following limitations:

- The condition must be a **Boolean** expression that can be evaluated using the values in the row being inserted or updated.
- The condition cannot contain **sub queries** or **sequences**.
- The condition cannot include the SYSDATE, UID, USER or USERENV SQL functions.

Defining Different Constraints On A Table

Example 15:
Drop the table **PERMISSIONS**, if already exists. Create **PERMISSIONS** table where:

- The **PERM_ID** is a primary key to this table
- The **PERM_USER** is the foreign key referencing the **USER_ID** column in the table **USERS**
- The fields **PERM_GRANT_ON** and **PERM_VALUE** cannot have a NULL value

```
DROP TABLE PERMISSIONS;
CREATE TABLE PERMISSIONS (PERM_USER VarChar2(7), PERM_ID VarChar2(7),
   PERM_GRANT_ON VarChar2(35) NOT NULL, PERM_VALUE Number(2) NOT NULL,
   CONSTRAINT PK_PERM_PERMID PRIMARY KEY(PERM_ID),
   CONSTRAINT FK_PERM_USERID FOREIGN KEY(PERM_USER)
      REFERENCES USERS(USER_ID));
```

Output:

```
Table created.
```

Generate Primary Key Values For A Table

Example 16:
Create a table Employers with a **NOT NULL** column such as **EMP_ID** and a column named **NAME**. Populate these columns with some values.

Solution:

```
CREATE TABLE Employers (EMP_ID NUMBER, NAME VARCHAR2(25));
```

Output:

```
Table created.
```

```
INSERT INTO EMPLOYERS VALUES(0, 'Silicon Chip Technologies');
INSERT INTO EMPLOYERS VALUES(0, 'Perfect Systemz');
INSERT INTO EMPLOYERS VALUES(0, 'Sigma Inc');
INSERT INTO EMPLOYERS VALUES(0, 'Essenpros');
INSERT INTO EMPLOYERS VALUES(0, 'Jasper International');
```

Output: (For each of the above INSERT INTO statement)

```
1 row created.
```

The data held in the table **EMPLOYERS:**

```
SELECT * FROM EMPLOYERS;
```

Output:

```
    EMP_ID NAME
---------- -------------------------
         0 Silicon Chip Technologies
         0 Perfect Systemz
         0 Sigma Inc
         0 Essenpros
         0 Jasper International
```

Now issue the following command to generate primary key for the column **EMP_ID:**

```
UPDATE EMPLOYERS SET EMP_ID = ROWNUM;
```

Output:

```
5 rows updated.
```

The data held in the table **EMPLOYERS** now:

```
SELECT * FROM EMPLOYERS;
```

Output:

```
    EMP_ID NAME
---------- -------------------------
         1 Silicon Chip Technologies
         2 Perfect Systemz
         3 Sigma Inc
         4 Essenpros
         5 Jasper International
```

OR

Use a sequences generator:

```
CREATE SEQUENCE SEQ_Emp_ID START WITH 1 INCREMENT BY 1;
```

Output:

```
Sequence created.
```

```
UPDATE EMPLOYERS SET EMP_ID=SEQ_Emp_ID.NEXTVAL;
```

Output:

```
5 rows updated.
```

The data held in the table **EMPLOYERS** now:

```
SELECT * FROM EMPLOYERS;
```

Output:

```
    EMP_ID NAME
---------- -------------------------
         1 Silicon Chip Technologies
         2 Perfect Systemz
         3 Sigma Inc
         4 Essenpros
         5 Jasper International
```

Finally, create a unique index on the column **EMP_ID** as:

```
CREATE UNIQUE INDEX idxEmp_ID ON EMPLOYERS(EMP_ID);
```

Output:

```
Index created.
```

This will now restrict the insertion of duplicate values:

```
INSERT INTO EMPLOYERS VALUES(1, 'Silicon Chip Technologies');
```

Output:

```
INSERT INTO EMPLOYERS VALUES(1, 'Silicon Chip Technologies')
*
ERROR at line 1:
ORA-00001: unique constraint (DBA_PM.IDXEMP_ID) violated
```

The USER_CONSTRAINTS Table

A table can be created with multiple constraints attached to its columns. If a user wishes to see the table structure along with its constraints, Oracle provides the **DESCRIBE <TableName>** command.

This command displays only the column names, data type, size and the NOT NULL constraint. The information about the other constraints that may be attached to the table columns such as the PRIMARY KEY, FOREIGN KEY and so on is not available using the DESCRIBE verb.

Oracle stores such information in a table called **USER_CONSTRAINTS**. Querying **USER_CONSTRAINTS** provides information bound to the names of ail the constraints on the table. **USER_CONSTRAINTS** comprises of multiple columns, some of which are described below:

USER_CONSTRAINTS Table:

Column Name	Description
OWNER	The owner of the constraint.
CONSTRAINT_NAME	The name of the constraint
TABLE_NAME	The name of the table associated with the constraint
CONSTRAINT_TYPE	The type of constraint: **P**: Primary Key Constraint **R**: Foreign Key Constraint **U**: Unique Constraint **C**: Check Constraint
SEARCH_CONDITION	The search condition used (for CHECK Constraints)
R_OWNER	The owner of the table referenced by the FOREIGN KEY constraints
R_CONSTRAINT_NAME	The name of the constraint referenced by a FOREIGN KEY constraint.

Example 17:
View the constraints of the table **PERMISSIONS**

```
SELECT OWNER, CONSTRAINT_NAME, CONSTRAINT_TYPE
  FROM USER_CONSTRAINTS WHERE TABLE_NAME='PERMISSIONS';
```

Output:

```
OWNER        CONSTRAINT_NAME                C
------------ ------------------------------ -
DBA_PM       SYS_C005708                    C
DBA_PM       SYS_C005709                    C
DBA_PM       PK_PERM_PERMID                 P
DBA_PM       FK_PERM_USERID                 R
```

Defining Integrity Constraints Via The Alter Table Command

Integrity constraints can be defined using the **constraint** clause, in the **ALTER TABLE** command.

REMINDER

Oracle **will not allow** constraints defined using the **ALTER TABLE**, to be applied to the table if data previously placed in the table **violates such constraints**.

If a Primary key constraint was being applied to a table in retrospect and the column has duplicate values in it, the Primary key constraint **will not** be set to that column.

The following examples show the definitions of several integrity constraints:

Example 18:
Alter the table **COUNTRY** by adding a primary key on the column **COUNTRY_ID**.

```
ALTER TABLE COUNTRY ADD PRIMARY KEY (COUNTRY_ID);
```

Output:

```
Table altered.
```

Example 19:
Add FOREIGN KEY constraint on the column **CAND_ID** belonging to the table **CAND_ADDR**, which references the table **CANDIDATE**. Reduce the size of the column **CAND_ZIP** to 10.

```
ALTER TABLE CAND_ADDR ADD CONSTRAINT FK_CANDADDR_CANDID
  FOREIGN KEY(CAND_ID) REFERENCES CANDIDATE(CAND_ID)
  MODIFY(CAND_ZIP VARCHAR2(10));
```

Output:

```
Table altered.
```

Dropping Integrity Constraints Via The ALTER TABLE Command

Integrity constraint can be dropped if the rule that it enforces is no longer **true** or if the constraint is no longer **needed**. Drop the constraint using the **ALTER TABLE** command with the **DROP** clause. The following examples illustrate the dropping of integrity constraints.

Example 20:
Drop the PRIMARY KEY constraint from **COUNTRY**.

```
ALTER TABLE COUNTRY DROP PRIMARY KEY;
```

Output:

```
Table altered.
```

Example 21:
Drop FOREIGN KEY constraint on column CAND_ID from the table CAND_ADDR

```
ALTER TABLE CAND_ADDR DROP CONSTRAINT FK_CANDADDR_CANDID;
```

Output:

```
Table altered.
```

REMINDER

Dropping UNIQUE and PRIMARY KEY constraints **also drops all** associated indexes.

Default Value Concepts

At the time of table creation a **default value** can be assigned to a column. When a record is loaded into the table and the column is left empty, the Oracle engine will automatically load this column with the default value specified. The data type of the default value should match the data type of the column. The **DEFAULT** clause can be used to specify a default value for a column.

Syntax:

```
<ColumnName> <Datatype>(<Size>) DEFAULT <Value>;
```

Example 22:
Create CAND_LANG table where the column SPOKEN and WRITTEN are a VarChar2 fields and by default holds the value **Good**.

```
DROP TABLE CAND_LANG;
CREATE TABLE CAND_LANG (LANG_ID Number(2), CAND_ID VarChar2(7),
  SPOKEN VarChar2(5) DEFAULT 'Good', WRITTEN varchar2(5) DEFAULT
'Good');
```

Output:

```
Table created.
```

REMINDER

- The data type of the default value should match the data type of the column
- Character and date values will be specified in single quotes
- If a column level constraint is defined on the column with a default value, the default value clause must precede the constraint definition

Thus the syntax will be:

```
<ColumnName> <Datatype>(<Size>)
    DEFAULT <Value> <Constraint Definition>
```

The Business Model With Business Rule Constraints

Business Rule Constraints

Oracle allows the application of **business rules** to table columns. Business managers determine business rules which may vary from system to system as mentioned earlier. These rules are applied to data, **prior** the data is being inserted into table columns. This ensures that the data (**records**) in the table have integrity.

For example, the rule that no employee in the company shall get a salary less than Rs.1000/- is a business rule. This means that no cell in the **salary** column of the employee table should hold a **value** less than 1000. If an attempt is made, to insert a value less than 1000 into the salary column, the database engine rejects the entire record automatically and displays some sort of error message.

Business rules can be implemented in Oracle by using **CHECK** constraints. Check Constraints can be bound to a **column** or a **table** using the **CREATE TABLE** or **ALTER TABLE** command.

Business rule validation checks are performed when any table **write** operation is carried out. Any **insert** or **update** statement causes the relevant Check constraint to be evaluated. The Check constraint must be satisfied for the write operation to succeed. Thus **Check constraints** ensure the integrity of data in tables.

Conceptually, data constraints are connected to a column, by the Oracle engine, as **flags**. Whenever, an attempt is made to load the column with data, the Oracle engine observes the flag and recognizes the presence of a constraint. The Oracle engine then retrieves the Check constraint definition and then applies the Check constraint definition, to the data being loaded into the table column. If the data being entered into a column fails any of the data constraint checks, the **entire** record is rejected. The Oracle engine will then flash an appropriate **error message** to the console where the insert statement originated.

Oracle allows programmers to define constraints at:

- Column Level
- Table Level

Column Level Constraints

If data constraints are defined as an attribute of a column definition when creating or altering a table structure, they are **column level constraints**.

WARNING

Column level constraints are applied to the **current column**. The current column is the column that immediately **precedes** the constraint (**i.e.** they are local to a specific column). A column level constraint **cannot** be applied if the data constraint spans **across multiple columns** in a table.

Table Level Constraints

If data constraints are defined **after defining all table column attributes** when creating or altering a table structure, it is a **table level constraint.**

REMINDER

A table level constraint **must** be applied if the data constraint **spans across multiple columns** in a table.

Constraints are stored as a part of the global table definition by the Oracle engine in its **system tables**. The SQL syntax used to attach the constraint will change depending upon whether it is a column level or table level constraint.

Performing Operations On A Table Based On Other Tables

Creating A Table From A Table

Syntax:

```
CREATE TABLE <NewTableName> (<ColumnName1>[, < ColumnName2>, . . .]) AS
  SELECT <ColumnName1>[, <ColumnName2>, . . . ] FROM <ExistingTableName>;
```

Example 23:
Create a table named **JOB_SPEC** having three fields i.e. CATEGORY_ID, CATEGORY_NAME and CATEGORY_REMARKS from the source table named CATEGORY and rename the field CATEGORY_REMARKS to CATEGORY_INFO.

```
CREATE TABLE JOB_SPEC (CATEGORY_ID, CATEGORY_NAME, CATEGORY_INFO)
  AS SELECT CATEGORY_ID, CATEGORY_NAME, CATEGORY_REMARKS
     FROM CATEGORY;
```

Output:

```
Table created.
```

REMINDER

If the Source Table **CATEGORY** was populated with records then the target table **JOB_SPEC** will also be populated with the same.

The **Source** table is the table identified in the **SELECT** section of this SQL sentence. The **Target** table is one identified in the **CREATE** section of this SQL sentence. This SQL sentence populates the Target table with data from the Source table.

To create a Target table without the records from the source table (i.e. create the structure only), the select statement must have a **WHERE clause**. The **WHERE clause** must specify a condition that **cannot** be satisfied.

This means the **SELECT** statement in the CREATE TABLE definition **will not retrieve** any rows from the source table but will just retrieve the table structure and thus the target table will be created empty.

Example 24:
Drop the table **JOB_SPEC** if it already exists. Create a table named **JOB_SPEC** having three fields i.e. CATEGORY_ID, CATEGORY_NAME and CATEGORY_REMARKS from the source table named CATEGORY and rename the field CATEGORY_REMARKS to CATEGORY_INFO. The table JOB_SPEC should not be populated with any records.

```
DROP TABLE JOB_SPEC;
CREATE TABLE JOB_SPEC (CATEGORY_ID, CATEGORY_NAME, CATEGORY_INFO)
  AS SELECT CATEGORY_ID, CATEGORY_NAME, CATEGORY_REMARKS
     FROM CATEGORY  WHERE 1=2;
```

Output:

```
Table created.
```

Inserting Data Into A Table From Another Table

In addition to inserting data one row at a time into a table, it is quite possible to populate a table with data that already exists in another table. The syntax for doing so is as follows:

Syntax:

```
INSERT INTO <TableName>
  SELECT <ColumnName1>[, <ColumnName2>, . . . ] FROM <TableName>;
```

Example 25:
Insert data in the table **JOB_SPEC** using the table **CATEGORY** as a source of data.

```
INSERT INTO JOB_SPEC SELECT CATEGORY_ID, CATEGORY_NAME,
  CATEGORY_REMARKS FROM CATEGORY;
```

Output:

```
5 rows created.
```

Insertion Of A Data Set Into A Table From Another Table

Syntax:

```
INSERT INTO <TableName> SELECT <ColumnName1>[, <ColumnName2>, . . . ]
  FROM <TableName> WHERE <Condition>;
```

Example 26:
Populate the **JOB_SPEC** table with only those categories where category ID is below 5. Use the data from the **CATEGORY** table.

```
INSERT INTO JOB_SPEC SELECT CATEGORY_ID, CATEGORY_NAME,
  CATEGORY_REMARKS FROM CATEGORY WHERE CATEGORY_ID < 5;
```

Output:

```
4 rows created.
```

Removal Of Specific Row(s) Based On The Data Held By The Other Table

Sometimes it is desired to delete records in one table based on values in another table. Since it is not possible to list more than one table in the FROM clause while performing a delete, the EXISTS clause can be used.

Example 27:
Remove the address details of the candidate named **Chaaya**.

```
DELETE FROM CAND_ADDR WHERE EXISTS(SELECT CAND_ID FROM CANDIDATE
  WHERE CANDIDATE.CAND_ID=CAND_ADDR.CAND_ID
    AND CANDIDATE.CAND_FIRST_NAME='Chhaya');
```

Output:

```
2 rows deleted.
```

Explanation:
The above SQL sentence will delete all those records from the **CAND_ADDR** table where the CAND_ID field holds a value that matches the **CAND_ID** field of the **CANDIDATE** table. This is done using a sub-query which is responsible to extract the **CAND_ID** of the candidate named 'Chhaya' using the **WHERE** clause.

Displaying Information About User Created Objects

Table/s Created By A User

The command shown below is used to determine the tables to which a user has access. The tables created under the **currently selected** tablespace are displayed.

Example 28:

```
SELECT * FROM TAB;
```

Output:

```
TNAME                          TABTYPE  CLUSTERID
------------------------------ -------  ----------
PK_GENERATOR                   TABLE
CATEGORY                       TABLE
COUNTRY                        TABLE
LANGUAGE                       TABLE
USERS                          TABLE
PERMISSIONS                    TABLE
CLIENT                         TABLE
CLNT_CONTACT                   TABLE
CLNT_ADDR                      TABLE
CLNT_CONT_MOD                  TABLE
CLNT_ADDR_MAP                  TABLE
CANDIDATE                      TABLE
CAND_ADDR                      TABLE
CAND_DOC                       TABLE
CAND_QUAL                      TABLE
CAND_LANG                      TABLE
CAND_EMPL                      TABLE

17 rows selected.
```

Creating Synonyms

A **synonym** is an **alternative name** for objects such as tables, views, sequences, stored procedures and other database objects.

Syntax:

```
CREATE [OR REPLACE]  [PUBLIC] SYNONYM [SCHEMA .]
  SYNONYM_NAME FOR [SCHEMA .] OBJECT_NAME [@ DBLINK];
```

In the syntax,

- The **OR REPLACE** phrase allows recreating the synonym (if it already exists) without having to issue a DROP synonym command.
- The **PUBLIC** phrase means that the synonym is a public synonym and is accessible to all users

REMINDER

Remember though that the user must first have the appropriate object privileges to use the synonym

- The **SCHEMA** phrase points to an appropriate schema. If this phrase is omitted, Oracle assumes that a reference is made to the user's own schema
- The **OBJECT_NAME** phrase is the name of the object for which you are creating the synonym. It can be one of the following:
  - Table
  - Package
  - View
  - Materialized View
  - Sequence
  - Java Class Schema Object
  - Stored Procedure
  - User-Defined Object
  - Function
  - Synonym

Example 29:
Create a synonym for a table named **Candidate** held by the user DBA_PM.

To create a **PUBLIC** synonym log in as the **System DBA** using the **SYS** user.

```
CONNECT SYS/<password>@SCT as SYSDBA;
```

Output:

```
Connected.
```

```
CREATE PUBLIC SYNONYM RESOURCES FOR DBA_PM.CANDIDATE;
```

Output:

```
Synonym created.
```

Explanation:
Now, users of other schemas can reference the table **CANDIDATE**, now referred to as **RESOURCES without** having to prefix the table name with the schema named **DBA_PM**. For example a user named SHARANAM logs in via SQL *Plus and needs to access the **CANDIDATE** (i.e. **RESOURCES**) table owned by the user name DBA_PM. This can simply be done using:

```
SELECT * FROM RESOURCES;
```

Dropping Synonyms

Syntax:

```
DROP [PUBLIC] SYNONYM [SCHEMA.]SYNONYM_NAME [FORCE];
```

In the syntax,

- The **PUBLIC** phrase allows dropping a public synonym. If public is specified, then there is no need to specify a schema.
- The **FORCE** phrase will force Oracle to drop the synonym even if it has dependencies. It is probably not a good idea to use the force phrase as it can cause dependent Oracle objects to become invalid.

Example 30:
Drop the public synonym named **RESOURCES**.

To drop a **PUBLIC** synonym log in as the **System DBA** using the **SYS** user.

```
CONNECT SYS/<password>@SCT as SYSDBA;
```

Output:

```
Connected.
```

```
DROP PUBLIC SYNONYM RESOURCES;
```

Output:

```
Synonym dropped.
```

Hands On Exercises

1. Using the SQL*PLUS tool, create a tablespace named **SCT_PUB**. The name of the data file will be **Pub_Info.dat** and the size will be **25 MB**. The specification for the tablespace's storage parameters is as follows:
   INITIAL EXTENT SIZE 10k **NEXT EXTENT SIZE** 50k
   MINEXTENTS 1 **MAXEXTENTS** 499 **PCTINCREASE** 10
2. Using the SQL*PLUS tool, create a user bound to the **SCT_PUB** tablespace. The name of the Oracle user will be **DBA_PUB** and will have a password of choice.
3. Using the SQL*PLUS tool, convert the user named **DBA_PUB** into a super admin (Oracle DBA) for the **SCT_PUB** tablespace.

4. Create the tables described below:

Table Name: **CntryMstr**
Primary Key: CntryID **Foreign Key:**
Description: Used to store lookup information for name's of countries.

Column Name	Data Type	Size	Default	Attributes
CntryID	Number	3		Primary Key. Auto incremented
CntryName	VarChar2	40		NOT NULL

Table Name: **AthrMstr**
Primary Key: AthrID **Foreign Key:** CntryMstr.CntryID
Description: Used to store author's personal and postal information.

Column Name	Data Type	Size	Default	Attributes
AthrID	VarChar2	6		Primary Key. First letter must start with 'A'
Name	VarChar2	45		NOT NULL
BirthDt	Date			NOT NULL
Addr1	VarChar2	50		
City	VarChar2	20		
PinCode	VarChar2	10		
State	VarChar2	20		
CntryID	Number	3		Foreign Key. References **CntryID** in **CntryMstr**.

Table Name: **AthrCnct**
Primary Key: **Foreign Key:** AthrMstr.AthrID
Description: Used to store author's contact information.

Column Name	Data Type	Size	Default	Attributes
AthrID	VarChar2	6		Foreign Key. References **AthrID** in **AthrMstr**.
CnctType	VarChar2	1		T-Telephone, F-Facsimile, M-Mobile, E-Email, U-Website
Contact	VarChar2	50		

Table Name: **DmsnMstr**
Primary Key: DmsnID **Foreign Key:**
Description: Used to store lookup information for book's dimension.

Column Name	Data Type	Size	Default	Attributes
DmsnID	Number	3		Primary Key. Auto incremented
Name	VarChar2	15	NULL	
BkHght	Number	4, 2		NOT NULL
BkWdth	Number	4, 2		NOT NULL
MsrgUnt	VarChar2	1		C-centimeters, I-inches

Table Name: **BkMstr**
Primary Key: BkID **Foreign Key:** DmsnMstr.DmsnID
Description: Used to store list of books published.

Column Name	Data Type	Size	Default	Attributes
BkID	VarChar2	6		Primary Key. First letter must start with 'B'
Title	VarChar2	60		NOT NULL
Subject	VarChar2	255		
ISBN	VarChar2	16		NOT NULL
Price	Number	6, 2		
Pages	Number	4		
DmsnID	Number	3		Foreign Key. References **DmsnID** in **DmsnMstr**.

Table Name: **CntrctMstr**
Primary Key: CntrctID **Foreign Key:** AthrMstr.AthrID, BkMstr.BkID
Description: Used to store information related to contracts with authors on per book bases.

Column Name	Data Type	Size	Default	Attributes
CntrctID	VarChar2	6		Primary Key. First letter must start with 'C'
CntrctDt	Date			NOT NULL
AthrID	VarChar2	6		Foreign Key. References **AthrID** in **AthrMstr**.
BkID	VarChar2	6		Foreign Key. References **BkID** in **BkMstr**.
RyltPcnt	Number	6, 2		NOT NULL

Table Name: **PrntCycle**
Primary Key: PrntID **Foreign Key:** BkMstr.BkID
Description: Used to store information of various print cycles on per book bases.

Column Name	Data Type	Size	Default	Attributes
PrntID	VarChar2	6		Primary Key. First letter must start with 'P'
PrntDt	Date			NOT NULL
BkID	VarChar2	6		Foreign Key. References **BkID** in **BkMstr**.
Copies	Number	4		NOT NULL

Table Name: **Sales**
Primary Key: SalesID **Foreign Key:** BkMstr.BkID
Description: Used to store information related to contracts with authors on per book bases.

Column Name	Data Type	Size	Default	Attributes
SalesID	VarChar2	6		Primary Key. First letter must start with 'S'
SalesPrd	VarChar2	12		NOT NULL
BkID	VarChar2	6		Foreign Key. References **BkID** in **BkMstr**.
OpStck	Number	4	0	

Table Name: **Sales** (Continued)

Column Name	Data Type	Size	Default	Attributes
Cmplmtry	Number	4	0	
Sold	Number	5	0	
ClStck	Number	4	0	

Table Name: **RyltDtls**
Primary Key: **Foreign Key:** AthrMstr.AthrID, BkMstr.BkID

Description: Used to store information related to royalty payable to authors on per book bases.

Column Name	Data Type	Size	Default	Attributes
AthrID	VarChar2	6		Foreign Key. References **AthrID** in **AthrMstr**.
BkID	VarChar2	6		Foreign Key. References **BkID** in **BkMstr**.
FnclYr	VarChar2	9		NOT NULL
SalesAmt	Number	10, 2		NOT NULL
DscntAmt	Number	8, 2		
TrdAmt	Number	10, 2		
RyltAmt	Number	8, 2		

5. Insert the following data into their respective tables:

a) Data for **CntryMstr** table:

CntryID	CntryName
1	India
4	Sri Lanka
7	France

CntryID	CntryName
2	Bangladesh
5	United Kingdom
8	Canada

CntryID	CntryName
3	Pakisthan
6	U. S. A.

b) Data for **AthrMstr** table:

AthrID	Name	BirthDt	Addr1	City	Pincode	State	Cntry ID
A00001	Dr. C. K. Shah	12-03-1965	Shahpur, 5, R. P. Lane,	Mumbai	400016	Maharashtra	1
A00002	Thomas Hunt	30-01-1980	343, Apple Street,	York			5
A00003	Abdul M. Khan	05-12-1970	Hussain House, F-12, Rd. No. 14,	Islamabad			3
A00004	R. Krishna	25-06-1955		Chennai	600020	Tamil Nadu	1
A00005	Parvati Patil	21-01-1974	201, Fatima C.H.S, SV Road, Jogeshwari (W)	Mumbai	400102	Maharashtra	1
A00006	Felice DeCaper	11-02-1979		Paris			7

Data for **AthrMstr** table: **(Continued)**

AthrID	Name	BirthDt	Addr1	City	Pincode	State	Cntry ID
A00007	Hemant Rana	15-08-1970	714-W, Maple Crossing,	Toronto			8
A00008	Jonny Parker	06-10-1981		Alabama		Texas	6

c) Data for **AthrCnct** table:

AthrID	CnctType	Contact
A00001	T	+91 022 24310051
A00002	M	+44 9300458790
A00003	T	+92 015 5442542
A00004	T	+91 066 28568462
A00005	T	+91 022 32654584
A00006	T	+33 2546464
A00007	T	+1 145841656
A00008	T	+1 154648715

AthrID	CnctType	Contact
A00001	U	www.heatlhyshah.com
A00002	U	www.pacsystems.com
A00003	E	abdulkhan@hotmail.com
A00004	E	supports@ramanworks.com
A00005	E	pparvati@rediffmail.com
A00006	U	www.feliceideas.com
A00007	E	hemantrana@usa.net
A00008	E	jonnyparker@hotmail.com

d) Data for **DmsnMstr** table:

Dmsn ID	Name	BkHght	BkWdth	MsrgUnt
1	Letter	11	8.5	I
3	A5	8.27	5.83	I
5	B5 (JIS)	10.12	7.17	I
7	Crown	19	12	C

Dmsn ID	Name	BkHght	BkWdth	MsrgUnt
2	A4	11.69	8.27	I
4	Foolscap	43	34	C
6	B5 (ISO)	9.84	6.93	I
8	Royal	63	51	C

e) Data for **BkMstr** table:

BkID	Title	Subject	ISBN	Price	Pages	Dmsn ID
B00001	Fighting Fit	Health - Physical Fitness	A003-01-0001-A03	120	150	6
B00002	Business And Policies	Economics - Finances	A003-11-0030-A05	275	325	3
B00003	Concepts Of E-Business	Business Management	A003-31-0120-L35	200	150	4
B00004	Development In C++	Computers - Programming	A004-02-0080-C51	225	400	3
B00005	Development In PHP	Computers - Programming	A004-31-0120-L35	450	550	6
B00006	Stress Controls	Health - Physical Fitness	A004-05-0201-F74	150	120	7
B00007	Modern Economics	Economics - Principles	A004-62-0250-P13	275	350	3
B00008	Tales Of The Shadow - I	Fictions - Horror	A004-80-3247-X37	100	250	7
B00009	Concepts Of Networking	Business Management	A005-08-1020-C35	180	150	4
B00010	Tales Of The Shadow - II	Fictions - Horror	A005-11-0047-X17	100	250	7

f) Data for **CntrctMstr** table:

CntrctID	CntrctDt	AthrID	BkID	Rylt Pcnt
C00001	22-05-2002	A00001	B00001	10
C00003	10-01-2003	A00003	B00003	12
C00005	18-06-2003	A00005	B00004	5
C00007	27-11-2003	A00007	B00005	5
C00009	07-03-2004	A00002	B00007	12
C00011	16-11-2004	A00003	B00009	12

CntrctID	CntrctDt	AthrID	BkID	Rylt Pcnt
C00002	15-09-2002	A00002	B00002	12
C00004	18-06-2003	A00004	B00004	5
C00006	27-11-2003	A00006	B00005	5
C00008	05-01-2004	A00001	B00006	10
C00010	22-05-2004	A00008	B00008	5
C00012	22-12-2004	A00008	B00010	5

g) Data for **PrntCycle** table:

PrntID	PrntDt	BkID	Copies
P00001	15-03-2003	B00001	3000
P00003	21-08-2003	B00003	9000
P00005	31-11-2003	B00001	5000
P00007	10-01-2004	B00005	8000
P00009	14-05-2004	B00002	3000
P00011	12-07-2004	B00008	9000
P00013	21-11-2004	B00006	3000
P00015	08-01-2005	B00009	9000
P00017	07-02-2005	B00005	5000
P00019	14-03-2005	B00003	5000

PrntID	PrntDt	BkID	Copies
P00002	09-05-2003	B00002	3000
P00004	16-09-2003	B00002	8000
P00006	29-12-2003	B00004	8000
P00008	31-03-2004	B00006	3000
P00010	23-06-2004	B00007	3000
P00012	01-09-2004	B00005	5000
P00014	12-12-2004	B00008	5000
P00016	28-01-2005	B00010	9000
P00018	19-02-2005	B00002	5000
P00020	05-04-2005	B00006	3000

h) Data for **Sales** table:

SalesID	SalesPrd	BkID	OpStck	Cmplmtry	Sold	ClStck
S00001	APR03-JUN03	B00001	3000	10	440	2500
S00002	APR03-JUN03	B00002	0	25	1575	1400
S00003	JUL03-SEP03	B00001	2500	20	1270	1210
S00004	JUL03-SEP03	B00002	1400	10	2500	6890
S00005	JUL03-SEP03	B00003	0	15	1000	8985
S00006	OCT03-DEC03	B00001	1210	5	1055	5150
S00007	OCT03-DEC03	B00002	6890	0	1905	4985
S00008	OCT03-DEC03	B00003	8985	10	1775	7200
S00009	JAN04-MAR04	B00001	5150	0	950	4200
S00010	JAN04-MAR04	B00002	4985	0	975	4010
S00011	JAN04-MAR04	B00003	7200	10	2025	5165
S00012	JAN04-MAR04	B00004	8000	30	1990	5980
S00013	JAN04-MAR04	B00005	0	20	2230	8000
S00014	APR04-JUN04	B00001	4200	0	980	3220
S00015	APR04-JUN04	B00002	4010	0	2570	4440
S00016	APR04-JUN04	B00003	5165	0	1865	3300
S00017	APR04-JUN04	B00004	5980	35	1965	3980
S00018	APR04-JUN04	B00005	5750	10	3460	2280
S00019	APR04-JUN04	B00006	3000	10	990	2000

Data for **Sales** table: **(Continued)**

SalesID	SalesPrd	BkID	OpStck	Cmplmtry	Sold	ClStck
S00020	JUL04-SEP04	B00001	3220	0	820	2400
S00021	JUL04-SEP04	B00002	4440	10	2790	1640
S00022	JUL04-SEP04	B00003	3300	0	1300	2000
S00023	JUL04-SEP04	B00004	3980	20	2000	1960
S00024	JUL04-SEP04	B00005	2280	10	3070	4200
S00025	JUL04-SEP04	B00006	2000	5	1000	995
S00026	JUL04-SEP04	B00007	3000	10	540	2450
S00027	JUL04-SEP04	B00008	0	50	2950	6000
S00028	OCT04-DEC04	B00001	2400	0	800	1600
S00029	OCT04-DEC04	B00002	1640	0	1100	540
S00030	OCT04-DEC04	B00003	2000	0	1125	875
S00031	OCT04-DEC04	B00004	1960	10	1950	0
S00032	OCT04-DEC04	B00005	4200	0	3250	950
S00033	OCT04-DEC04	B00006	995	5	1250	2740
S00034	OCT04-DEC04	B00007	2450	10	590	1850
S00035	OCT04-DEC04	B00008	6000	25	3275	7700
S00036	JAN05-MAR05	B00001	1600	0	705	895
S00037	JAN05-MAR05	B00002	540	0	1575	3965
S00038	JAN05-MAR05	B00003	875	0	1075	4800
S00039	JAN05-MAR05	B00005	950	0	2950	3000
S00040	JAN05-MAR05	B00006	2740	0	1570	1170
S00041	JAN05-MAR05	B00007	1850	5	645	1200
S00042	JAN05-MAR05	B00008	7700	15	2780	4905
S00043	JAN05-MAR05	B00009	0	20	980	8000
S00044	JAN05-MAR05	B00010	0	10	590	8400
S00045	APR05-JUN05	B00001	895	0	545	350
S00046	APR05-JUN05	B00002	3965	5	1995	1965
S00047	APR05-JUN05	B00003	4800	5	1545	3250
S00048	APR05-JUN05	B00005	3000	0	2450	650
S00049	APR05-JUN05	B00006	1170	0	1670	2500
S00050	APR05-JUN05	B00007	1200	0	750	450
S00051	APR05-JUN05	B00008	4905	5	2900	2000
S00052	APR05-JUN05	B00009	8000	10	1490	6500
S00053	APR05-JUN05	B00010	8400	30	1970	6400

i) Data for **RyltDtls** table:

AthrID	BkID	FnclYr	SalesAmt	DscntAmt	TrdAmt	RyltAmt
A00001	B00001	2003-2004	447000	44700	402300	40230
A00002	B00002	2003-2004	1912625	153010	1759615	211152.8
A00003	B00003	2003-2004	960000	76800	883200	105984

Data for **RyltDtls** table: **(Continued)**

AthrID	BkID	FndYr	SalesAmt	DscntAmt	TrdAmt	RyltAmt
A00004	B00004	2003-2004	447750	44775	402975	20148.75
A00005	B00004	2003-2004	447750	44775	402975	20148.75
A00006	B00005	2003-2004	1003500	100350	903150	45157.5
A00007	B00005	2003-2004	1003500	100350	903150	45157.5
A00001	B00001	2004-2005	396600	39660	356940	35694
A00002	B00002	2004-2005	2209625	176770	2032855	243942.6
A00003	B00003	2004-2005	1073000	85840	987160	118459.2
A00004	B00004	2004-2005	1330875	133087.5	1197788	59889.38
A00005	B00004	2004-2005	1330875	133087.5	1197788	59889.38
A00006	B00005	2004-2005	5724000	572400	5151600	257580
A00007	B00005	2004-2005	5724000	572400	5151600	257580
A00001	B00006	2004-2005	721500	72150	649350	64935
A00002	B00007	2004-2005	488125	39050	449075	53889
A00008	B00008	2004-2005	900500	135075	765425	38271.25
A00003	B00009	2004-2005	176700	14112	162288	19474.56
A00008	B00010	2004-2005	59000	8150	50150	2507.5
A00001	B00001	2005-2006	65280	6528	58752	5875.2
A00002	B00002	2005-2006	548625	43890	504735	60568.2
A00003	B00003	2005-2006	309000	24720	284280	34113.6
A00006	B00005	2005-2006	1102500	110250	992250	49612.5
A00007	B00005	2005-2006	1102500	110250	992250	49612.5
A00001	B00006	2005-2006	250500	25050	225450	22545
A00002	B00007	2005-2006	206250	16500	189750	22770
A00008	B00008	2005-2006	290000	43500	246500	12325
A00003	B00009	2005-2006	268200	21456	246744	29669.28
A00008	B00010	2005-2006	197000	29550	167450	8372.5

6. Exercise on retrieving records from a table

a. Find out the names of all the authors.

b. Retrieve the entire contents of the BkMstr table.

c. Retrieve the list of names, city and the sate of all the authors.

d. List the various royalty payable to authors on books.

e. List all the authors who are located in India.

7. Exercise on updating records in a table

a. Change the city of author 'A00005' to 'Bangalore'.

b. Change the royalty of author 'A00001' to 9%.

c. Change the cost price of the book named 'Development In C++' to Rs. 200.

8. Exercise on deleting records in a table

a. Delete all sales entries from the Sales table whose sales period is in the last quarter on the year 2004.

b. Delete all contracts for book where the authors are 'A00004' and 'A00005'.

c. Delete royalty information, where discounted amount is less than 8% of sales.

9. Exercise on altering the table structure

a. Add a column called 'Profession' of data type 'varchar2' and size '20' to the AthrMstr table.

b. Change the size of SalesAmt column in RyltDtls to 12,2.

10. Exercise on renaming the table

a. Change the name of the Sales table to QtrlyBkSales.

11. Exercise on deleting the table structure along with the data

a. Destroy the all tables created for the hands on exercises.

SECTION II: WORKING WITH ORACLE 10g

Advance Interaction With SQL

Running Calculations On Table Data Using Operators

None of the SQL coding techniques used till now permits the display of data from a table **after some arithmetic** has been done with it.

The arithmetic calculations may include displaying an employee's name and the employee's salary from the Employee_Master table **along with** the **annual salary** of the employee (**i.e.** Salary*12). The multiplication operation of (Monthly Salary * 12) is an example of table data arithmetic.

Arithmetic and **logical** operators give a new dimension to SQL sentences.

Arithmetic Operators

Oracle allows the use of arithmetic operators while viewing records from a table as well as when performing Data Manipulation operations such as Insert, Update and Delete. The arithmetic operations are:

+	Addition	*	Multiplication
-	Subtraction	**	Exponentiation
/	Division	( )	Enclosed operation

Example 1:
List the candidates who have registered with the Personnel Management System along with their period of work experience. Display their **period of work experience in months**.

Synopsis:

Tables:	CANDIDATE
Columns:	CAND_USER_ID, CAND_ID, CAND_FIRST_NAME, CAND_TOT_EXP
Technique:	**Operators:** * (Multiplication) **; Others:** || (Concatenate)

Solution:

```
SELECT CAND_USER_ID, CAND_ID, CAND_FIRST_NAME,
      (CAND_TOT_EXP * 12) || ' months' FROM CANDIDATE;
```

Output:

```
CAND_US CAND_ID CAND_FIRST_NAME    (CAND_TOT_EXP*12)||'MONTHS'
------- ------- ------------------ ---------------------------
US00001 CD00001 Sharanam           36 months
US00002 CD00002 Hansel             12 months
US00003 CD00003 Ivan               360 months
US00004 CD00004 Chhaya             48 months
US00005 CD00005 Shravan            36 months
```

Explanation:
Here, **(CAND_TOT_EXP*12) ||'months'** is **not** a column in the table **CANDIDATE**. However, the arithmetic specified is done on the contents of the columns **CAND_TOT_EXP** of the table **CANDIDATE** and displayed in the output of the query.

By default, the Oracle engine will use the column names of the table **CANDIDATE** as column headers when displaying column output on the VDU screen.

Since there are no columns with the arithmetic expression applied on the table **CANDIDATE**, the Oracle engine performs the required arithmetic operation and uses the **formula** as the **default** column header when displaying output as seen above.

Renaming Columns Used With Expression Lists

Rename the default output column names with an **alias**, when required.

Syntax:

```
SELECT <ColumnName> <AliasName>, <ColumnName> <AliasName>
  FROM <TableName>;
```

Example 2:
List the candidates who have registered with the Personnel Management System along with their period of work experience. Calculate the **period of work experience in months**. Use an **ALIAS** to rename the calculated column to **Work Experience**.

Synopsis:

Tables:	CANDIDATE
Columns:	CAND_USER_ID, CAND_ID, CAND_FIRST_NAME, CAND_TOT_EXP
Technique:	**Operators:** * (Multiplication); **Others:** || (Concatenate), Alias

Solution:

```
SELECT CAND_USER_ID, CAND_ID, CAND_FIRST_NAME,
  (CAND_TOT_EXP * 12) || ' months' "Work Experience" FROM CANDIDATE;
```

Output:

```
CAND_US CAND_ID CAND_FIRST_NAME Work Experience
------- ------- --------------- ---------------
US00001 CD00001 Sharanam        36 months
US00002 CD00002 Hansel          12 months
US00003 CD00003 Ivan            360 months
US00004 CD00004 Chhaya          48 months
US00005 CD00005 Shravan         36 months
```

Explanation:
Here, **(CAND_TOT_EXP*12) ||'months'** is renamed to ALIAS "**Work Experience**".

Logical Operators

The logical operators that can be used in SQL sentences are:

The AND Operator

The AND operator allows the creation of SQL statements based on two or more conditions being met. It can be used in any valid SQL statement such as select, insert, update or delete. The AND operator requires that **each condition must be met** for the record to be included in the result set.

The Oracle engine will process all rows in a table and display the result only when **all** of the conditions specified using the **AND** operator are satisfied.

Example 3:
Display all candidates whose linguistic skills are good in both reading and writing.

Synopsis:

Tables:	CAND_LANG
Columns:	All Columns
Technique:	**Operators:** AND; **Clauses:** WHERE

Solution:

```
SELECT * FROM CAND_LANG WHERE SPOKEN='Good' AND WRITTEN='Good';
```

Output:

```
   LANG_ID CAND_ID SPOKE WRITT
---------- ------- ----- -----
         1 CD00002 Good  Good
         1 CD00004 Good  Good
         3 CD00004 Good  Good
         3 CD00005 Good  Good
```

Explanation:
Here, the AND operator is used to compare the value held in the date fields i.e. **SPOKEN** and **WRITTEN** with a constant i.e. **Good**. Only those records that satisfy this comparison are shown.

The OR Operator

The OR condition allows the creation of SQL statements where records are returned when **any one** of the conditions are met. It can be used in any valid SQL statement such as select, insert, update or delete. The OR condition requires that **any of** the conditions be met for the record to be included in the result set.

The Oracle engine will process all rows in a table and display the result only when **any of** the conditions specified using the **OR** operator is satisfied.

Example 4:
Display all candidates who have poor linguistic skills either in reading or writing.

Synopsis:

Tables:	CAND_LANG
Columns:	All Columns
Technique:	**Operators:** OR; **Clauses:** WHERE

Solution:

```
SELECT * FROM CAND_LANG WHERE SPOKEN='Poor' OR WRITTEN='Poor';
```

Output:

```
   LANG_ID CAND_ID SPOKE WRITT
---------- ------- ----- -----
         2 CD00003 Poor  Poor
         6 CD00005 Fair  Poor
```

Explanation:
Here, the **OR** operator is used to compare the value held in the fields **SPOKEN** and **WRITTEN**. This condition will only be satisfied if the value held in either the **SPOKEN** field or the **WRITTEN** field is **Poor**.

Combining The AND And OR Operator

Both the **AND** and **OR** operators can be combined in a single SQL statement. It can be used in any valid SQL statement such as select, insert, update or delete.

When combining these conditions, it is important to use brackets so that the database engine knows the order in which to evaluate each condition.

The Oracle engine will process all rows in a table and display the result only when **all** of the conditions specified using the **AND** operator are satisfied and when **any of** the conditions specified using the **OR** operator are satisfied.

Example 5:
Display all candidates whose **place of birth is Mumbai or** all candidates whose **last name** begin with the alphabet '**S**' **and** are **married**.

Synopsis:

Tables:	CANDIDATE
Columns:	CAND_ID, CAND_FIRST_NAME, CAND_SURNAME, CAND_BIRTHPLACE, CAND_MARITAL_STATUS
Technique:	**Functions:** UPPER(); **Operators:** LIKE, AND, OR; **Clauses:** WHERE; **Others:** || (Concatenate)

Solution:

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Candidate"
  FROM CANDIDATE WHERE (UPPER(CAND_BIRTHPLACE)='MUMBAI')
     OR (CAND_SURNAME LIKE 'S%' AND CAND_MARITAL_STATUS='Yes');
```

Output:

```
CAND_ID Candidate
------- ---------------
CD00001 Sharanam Shah
CD00002 Hansel Colaco
CD00003 Ivan Bayross
```

Explanation:
The above SQL query returns all those records:

1. Where the value held in the field **CAND_MARITAL_STATUS** is **Yes**
   AND
   The first character in the **CAND_SURNAME** field is '**S**'

OR

2. Where the value held in the field **CAND_BIRTHPLACE** is **MUMBAI**

The brackets determine what order the **AND** / **OR** conditions are evaluated in.

The NOT Operator

The Oracle engine will process all rows in a table and display only those records that **do not** satisfy the condition specified.

Example 6:
List the candidates who are **not** born in **Mumbai**.

Synopsis:

Tables:	CANDIDATE
Columns:	CAND_ID, CAND_FIRST_NAME, CAND_SURNAME, CAND_BIRTHPLACE
Technique:	**Functions:** UPPER(); **Operators:** NOT; **Clauses:** WHERE; **Others:** Alias

Solution:

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Candidate"
  FROM CANDIDATE WHERE NOT (UPPER(CAND_BIRTHPLACE)='MUMBAI');
```

Output:

```
CAND_ID Candidate
------- ----------------
CD00004 Chhaya Bankar
CD00005 Shravan Dhone
```

Explanation:
The above query **will not** display rows where the value of the field **CAND_BIRTHPLACE** is **Mumbai**. Hence, all those records, which satisfy the condition specified using **the NOT operator**, **will not be shown**.

IS NAN and IS INFINITE Operators

Besides the regular operators such as LIKE, IN, IS NULL and BETWEEN, Oracle 10g extends its list by providing **two new** operators to support the introduction of BINARY_FLOAT and BINARY_DOUBLE data types. These data types are capable of holding values like NaN and infinity.

The additional SQL operators are:

- IS NAN / IS NOT NAN: Matches the NaN special value (**i.e. N**ot **a N**umber)
- IS INFINITE / IS NOT INFINITE: Matches infinite value

Example 1:
The following statement displays the contents of the table named **EMPEXP**, after filtering it for **NaN** values in the **LASTSAL** column:

```
SELECT * FROM EMPEXP WHERE LASTSAL IS NAN;
```

Output:

```
     EMPNO     LASTSAL      EXPSAL
---------- ---------- ----------
         3        Nan         Nan
```

Example 2:
The following statement displays the contents of the table named **EMPEXP**, after filtering it for Infinite values in the **EXPSAL** column:

```
SELECT * FROM EMPEXP WHERE EXPSAL IS INFINITE;
```

Output:

```
     EMPNO     LASTSAL      EXPSAL
---------- ---------- ----------
         4          Inf         Inf
```

Searching For Specific Data

Range Searching

In order to select data that is **within** a range of user specified values, the **BETWEEN** operator is used. The **BETWEEN** operator allows the selection of rows that contain values within a specified lower and upper limit. The **range** specified using the **BETWEEN** operator is **inclusive**.

The **lower** value must be specified **first**. The two values in between the range must be linked with the logical operator **AND**. The **BETWEEN** operator can be used with both character and numeric data types. However, the data types cannot be mixed (**i.e.** the lower value, of a range of values from a character column and the higher value from a numeric column).

Example 7:
List the candidates whose passport shall expire in the years 2006 to 2010.

Synopsis:

Tables:	CAND_DOC
Columns:	CAND_ID, CAND_DOC_NO, CAND_DOC_NAME, CAND_DOC_EXP_DAT
Technique:	**Functions:** SUBSTRING(); **Operators:** AND, BETWEEN; **Clauses:** WHERE

Solution:

```
SELECT CAND_ID, CAND_DOC_NO, CAND_DOC_EXP_DAT FROM CAND_DOC
  WHERE CAND_DOC_NAME='Passport'
      AND SUBSTR(CAND_DOC_EXP_DAT, -4) BETWEEN 2006 AND 2010;
```

Equivalent to:

```
SELECT CAND_ID, CAND_DOC_NO, CAND_DOC_EXP_DAT FROM CAND_DOC
  WHERE CAND_DOC_NAME='Passport'
      AND SUBSTR(CAND_DOC_EXP_DAT, -4) >= 2006
      AND SUBSTR(CAND_DOC_EXP_DAT, -4) <= 2010;
```

Output:

```
CAND_ID CAND_DOC_NO          CAND_DOC_EXP
------- -------------------- ------------
CD00002 A1865183             11/08/2006
CD00005 A-4624525            21/01/2008
```

Explanation:
The first query will retrieve all those records from the **CAND_DOC** table where the value of the last four characters, held in the CAND_DOC_EXP_DAT field is 2006, 2007, 2008, 2009 or 2010. This is done by using SUBSTR() function which extracts year value from the CAND_DOC_EXP_DAT field. This is then compared using the **BETWEEN** operator.

The second query gives the same output using the **AND** operator.

Pattern Matching

The LIKE Predicate

The comparison operators discussed so far have compared one value, **exactly** to another value. Such precision may not always be desired or necessary. For this purpose Oracle provides the **LIKE** predicate.

The **LIKE** predicate allows comparison of one string value with another string value, which is **not identical** in all aspects. This is achieved by using wildcard characters. **Two** wildcard characters that are available for character data are:

- **%** allows finding a match for any string of any length (including zero length) (**i.e.** the percentage sign)
- _ allows finding a match for any single character (**i.e.** the underscore character)

Example 8:
List the candidate whose names have the second character as **a**.

Synopsis:

Tables:	CANDIDATE
Columns:	CAND_ID, CAND_FIRST_NAME, CAND_SURNAME
Technique:	**Operators:** LIKE; **Clauses:** WHERE; **Others:** || (Concatenate), Alias

Solution:

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Candidate"
  FROM CANDIDATE WHERE CAND_FIRST_NAME LIKE '_a%';
```

Output:

```
CAND_ID Candidate
------- ---------------
CD00002 Hansel Colaco
```

Explanation:
In the above example, all those records where the value held in the field **CAND_FIRST_NAME** contains the second character as **a** are displayed. The **_a** (i.e. underscore a) indicates that only one character can **precede** the character **a**. The **%** indicates that any number of characters can **follow** the letters **a**.

The IN And NOT IN Predicates

The arithmetic operator **(=)** compares a single value to another single value. In case a value needs to be compared to a list of values then the **IN** predicate is used. The IN predicate helps reduce the need to use multiple OR conditions

Example 9:
List the candidate ID, names and birthdates of the candidates named Ivan, Chhaya and Shravan.

Synopsis:

Tables:	CANDIDATE
Columns:	CAND_ID, CAND_FIRST_NAME, CAND_SURNAME, CAND_BIRTHDATE
Technique:	**Operators:** IN, **Clauses:** WHERE**; Others:** || (Concatenate), Alias

Solution:

```
SELECT CAND_ID, CAND_FIRST_NAME || ' ' || CAND_SURNAME "Candidate",
      CAND_BIRTHDATE
  FROM CANDIDATE WHERE CAND_FIRST_NAME IN('Ivan', 'Chhaya', 'Shravan');
```

Output:

```
CAND_ID Candidate       CAND_BIRT
------- --------------- ---------
CD00003 Ivan Bayross    23-JUN-52
CD00004 Chhaya Bankar   22-JUN-76
CD00005 Shravan Dhone   21-FEB-82
```

Explanation:
The above example, displays all those records where the **CAND_FIRST_NAME** field holds any one of the three specified values.

REMINDER

The **NOT IN** predicate is the opposite of the **IN** predicate. This will select all the rows where values **do not** match the values in the list.

Functions In Oracle

Oracle Functions serve the purpose of manipulating data and returning a result. Functions are also capable of accepting user-supplied variables **or** constants and working with them. Such variables or constants are called **arguments**. Any number of arguments (**or** no arguments at all) can be passed to a function in the following format:

Function_Name(argument1, argument2, ...)

Oracle Functions can be clubbed together depending upon whether they operate on a single row or a group of rows retrieved from a table. Accordingly, functions can be classified as follows:

Group Functions (Aggregate Functions)

Functions that act on a **set of values** are called **Group Functions**. For example, **SUM** is a Group function, which calculates the total of a set of numbers. A group function returns a single result row for a group of queried rows.

Scalar Functions (Single Row Functions)

Functions that act on **only one value** at a time are called **Scalar Functions**. For example, **LENGTH** is a Scalar function, which calculates the length of specific string. A single row function returns one result for every row of a queried table or view.

Single row functions can be further grouped according to the data type of the arguments they accept and the values they return. For example, the function **LENGTH** is related to the **String** Data type.

Functions can be classified according to different data types as follows:

- String Functions: For the **String** Data type
- Numeric Functions: For the **Number** Data type
- Conversion Functions: For the **Conversion** of one Data type to another.
- Date Functions: For the **Date** Data type

Aggregate Functions

AVG: Returns an average value of '**n**', **ignoring** null values in a column.

Syntax:

```
AVG ([<DISTINCT>|<ALL>] <n>)
```

Example:

```
SELECT AVG(CAND_TOT_EXP) "Average Experience" FROM CANDIDATE;
```

Output:

```
Average Experience
------------------
                8.2
```

REMINDER

In the above SELECT statement, the **AVG** function is used to calculate the average work experience of all candidates. The selected column is renamed as **Average Experience** in the output.

MIN: Returns a minimum value of **Expr**.

Syntax:

```
MIN([<DISTINCT>|<ALL>] <Expr>)
```

Example:

```
SELECT MIN(CAND_TOT_EXP) || ' years' "Min Experience" FROM CANDIDATE;
```

Output:

```
Min Experience
--------------
1 years
```

COUNT(expr): Returns the number of rows where **Expr** is not null.

Syntax:

```
COUNT([<DISTINCT>|<ALL>] <Expr>)
```

Example:

```
SELECT COUNT(CAND_ID) "No. Of Candidates" FROM CANDIDATE;
```

Output:

```
No. Of Candidates
-----------------
                5
```

COUNT(*): Returns the number of rows in the table, including duplicates and those with nulls.

Syntax:

```
COUNT(*)
```

Example:

```
SELECT COUNT(*) "No. Of Candidates" FROM CANDIDATE;
```

Output:

```
No. Of Candidates
-----------------
                5
```

MAX: Returns the maximum value of **Expr**.

Syntax:

```
MAX([<DISTINCT>|<ALL>] <Expr>)
```

Example:

```
SELECT MAX(CAND_TOT_EXP) || ' years' "Max Experience" FROM CANDIDATE;
```

Output:

```
Max Experience
--------------
4 years
```

SUM: Returns the sum of the values of '**n**'.

Syntax:

```
SUM([<DISTINCT>|<ALL>] <n>)
```

Example:

```
SELECT SUM(CAND_TOT_EXP) || ' years' "Total Experience" FROM CANDIDATE;
```

Output:

```
Total Experience
----------------
41 years
```

Numeric Functions

ABS: Returns the absolute value of '**n**'.

Syntax:

```
ABS(<n>)
```

Example:

```
SELECT ABS(-65) "Absolute" FROM DUAL;
```

Output:

```
Absolute
--------
      65
```

POWER: Returns **m** raised to the **n**th power. **n** must be an integer, else an error is returned.

Syntax:

```
POWER(<m>, <n>)
```

Example:

```
SELECT POWER(5,2) "Raised" FROM DUAL;
```

Output:

```
Raised
------
    25
```

ROUND: Returns **n**, rounded to **m** places to the right of a decimal point. If **m** is omitted, **n** is rounded to **0** places. **m** can be negative to round off digits to the left of the decimal point. **m** must be an integer.

Syntax:

```
ROUND(<n> [,<m>])
```

Example:

```
SELECT ROUND(11.37,1) "Round" FROM DUAL;
```

Output:

```
Round
-----
 11.4
```

SQRT: Returns square root of **n**. If **n**<0, NULL. SQRT returns a **real** result.

Syntax:

```
SQRT(<n>)
```

Example:

```
SELECT SQRT(100) "Square Root" FROM DUAL;
```

Output:

```
Square Root
-----------
         10
```

EXP: Returns **e** raised to the **nth** power, where **e = 2.71828183**.

Syntax:

```
EXP(<n>)
```

Example:

```
SELECT EXP(9) "Exponent" FROM DUAL;
```

Output:

```
  Exponent
----------
8103.08393
```

EXTRACT: Returns a value extracted from a date or an interval value. A DATE can be used only to extract YEAR, MONTH and DAY, while a timestamp with a time zone datatype can be used only to extract TIMEZONE_HOUR and TIMEZONE_MINUTE.

Syntax:

```
EXTRACT( {year | month | day | hour | minute | second | timezone_hour |
          timezone_minute | timezone_region | timezone_abbr}
    FROM { date_value | interval_value } )
```

Example:

```
SELECT EXTRACT(YEAR FROM DATE '2006-07-22') "Year",
  EXTRACT(MONTH FROM SYSDATE) "Month" FROM DUAL;
```

Output:

```
      Year      Month
---------- ----------
      2006          3
```

GREATEST: Returns the greatest value in a list of expressions.

Syntax:

```
GREATEST(<Expr1>[, <Expr2>, . . . , <Expr_n>])
```

where, **Expr1**, **Expr2**, ... **Expr_n** are expressions that are evaluated by the greatest function.

Example:

```
SELECT GREATEST(40, 35, 108) "Num", GREATEST('40', '35', '108') "Text"
  FROM DUAL;
```

Output:

```
       Num Text
---------- ----
       108 40
```

LEAST: Returns the least value in a list of expressions.

Syntax:

```
LEAST(<Expr1>[, <Expr2>, . . . , <Expr_n>])
```

where, **Expr1**, **Expr2**, ... **Expr_n** are expressions that are evaluated by the least function.

Example:

```
SELECT LEAST(40, 35, 108) "Num", LEAST('40', '35', '108') "Text" FROM DUAL;
```

Output:

```
       Num Text
---------- ----
        35 108
```

REMINDER

With the **GREATEST()** and **LEAST()** functions, if the datatypes of the expressions are different, all expressions will be converted to the datatype of the **first expression in the list**. If the comparison is based on a character comparison, one character is considered greater than another if it has a higher character set value in the ASCII chart.

MOD: Returns the remainder of a first number, divided by second number, passed as parameter. If the second number is zero, the result is **the same** as the first number.

Syntax:

```
MOD(<m>, <n>)
```

Example:

```
SELECT MOD(17, 8) "Mod1", MOD(17.5, 8) "Mod2" FROM DUAL;
```

Output:

```
      Mod1       Mod2
---------- ----------
         1        1.5
```

TRUNC: Returns a number truncated to a certain number of decimal places. The decimal place value must be an integer. If this parameter is omitted, the TRUNC function will truncate the number to **0** decimal places.

Syntax:

```
TRUNC(<number>, [<DecimalPlaces>])
```

Example:

```
SELECT TRUNC(25.415, 1) "Trunc1", TRUNC(25.415, -1) "Trunc2" FROM DUAL;
```

Output:

```
    Trunc1     Trunc2
---------- ----------
      25.4         20
```

FLOOR: Returns the largest integer value that is equal to or less than a number.

Syntax:

```
FLOOR(<n>)
```

Example:

```
SELECT FLOOR(13.8) "Flr1", FLOOR(24.15) "Flr2" FROM DUAL;
```

Output:

```
      Flr1       Flr2
---------- ----------
        13         24
```

CEIL: Returns the smallest integer value that is greater than or equal to a number.

Syntax:

```
CEIL(<n>)
```

Example:

```
SELECT CEIL(13.8) "Ceil1",CEIL(24.15) "Ceil2" FROM DUAL;
```

Output:

```
     Ceil1      Ceil2
---------- ----------
        14         25
```

REMINDER

Several other Numeric functions are available in Oracle. These include the following:

- **ACOS(), ASIN(), ATAN(), ATAN2(),**
- **COS(), COSH(), SIN(), SINH(), TAN(), TANH(),**
- **COVAR_POP(), COVAR_SAMP(),VAR_POP(), VAR_SAMP(),**
- **CORR(), SIGN()**

String Functions

LOWER: Returns a string with all letters in lowercase.

Syntax:

```
LOWER(<Str>)
```

Example:

```
SELECT LOWER('VAISHALI SHAH') "Lower" FROM DUAL;
```

Output:

```
Lower
-------------
vaishali shah
```

INITCAP: Returns a string with the first letter of each word in **upper case**.

Syntax:

```
INITCAP(<Str>)
```

Example:

```
SELECT INITCAP('CYNTHIA BAYROSS') "Title Case" FROM DUAL;
```

Output:

```
Title Case
---------------
Cynthia Bayross
```

UPPER: Returns a string with all letters forced to uppercase.

Syntax:

```
UPPER (<Str>)
```

Example:

```
SELECT UPPER('bRenEL CoLacO') "Capitalised" FROM DUAL;
```

Output:

```
Capitalised
-------------
BRENEL COLACO
```

SUBSTR: Returns a portion of characters, beginning at character **m** and going upto character **n**. If **n** is omitted, the result returned is upto the last character in the string. The first position of char is **1**.

Syntax:

```
SUBSTR(<Str>, <StartPosition>, [<Length>])
```

where, **Str** is the source string.
StartPosition is the position for extraction. The first position in the string is always 1.
Length is the number of characters to extract.

Example:

```
SELECT SUBSTR('AEROPLANES', 5, 6) "Substring" FROM DUAL;
```

Output:

```
Substr
------
PLANES
```

ASCII: Returns the NUMBER code that represents the character specified. If more than one character is entered, the function will return the value for the first character and ignore all of the characters after the first.

Syntax:

```
ASCII(<Char>)
```

where, **Char** is the specified character to retrieve the NUMBER code for.

Example:

```
SELECT ASCII('z') "ASCIIz", ASCII('Z') "ASCIIZ" FROM DUAL;
```

Output:

```
    ASCIIz     ASCIIZ
---------- ----------
       122         90
```

COMPOSE: Returns a Unicode string. It can be a **Char**, **VarChar2**, **NChar**, **NVarChar2**, **CLOB** or **NCLOB**.

Syntax:

```
COMPOSE(<Single>)
```

Below is a listing of **unistring** values that can be combined with other characters in the compose function.

Unistring Value	Resulting character
UNISTR('\0300')	grave accent (`)
UNISTR('\0301')	acute accent (´)
UNISTR('\0302')	circumflex (^)
UNISTR('\0303')	tilde (~)
UNISTR('\0308')	umlaut (¨)

Example:

```
SELECT 'ol' || COMPOSE('e' || UNISTR('\0301')) "Composed" FROM DUAL;
```

Output:

```
Composed
--------
olé
```

HINT

Notice the acute accent (´) is attached to the letter e.

DECOMPOSE: Accepts a Unicode string and returns a normal string.

Syntax:

```
DECOMPOSE(<single>)
```

Example:

```
SELECT DECOMPOSE('ol' || COMPOSE('e' || UNISTR('\0301'))) FROM DUAL;
```

Output:

```
Decomposed
----------
ole´
```

HINT

Notice the acute accent (´) appears after the letter e.

INSTR: Returns the location of a substring in a string.

Syntax:

```
INSTR(<Str1>, <Str2>, [<StartPosition>], [<nthAppearance>])
```

where, **Str1** is the string to search from
Str2 is the substring to search for in **Str1**
StartPosition is the position in **Str1** where the search will start. If omitted, it defaults to **1**. The first position in the string is **1**. If the **StartPosition** is negative, the function counts back **StartPosition** number of characters from the end of **Str1** and then searches towards the beginning of **Str1**.
nthAppearance is the **nth** appearance of **Str2**. If omitted, it defaults to **1**.

Example:

```
SELECT INSTR('SCT on the net', 't') "Instr1",
  INSTR('SCT on the net', 't', 1, 2) "Instr2" FROM DUAL;
```

Output:

```
 Instr1  Instr2
------- -------
      8      14
```

TRANSLATE: Replaces a sequence of characters in a string with another set of characters. However, it replaces a single character at a time. For example, it will replace the 1st character in the **StringToReplace** with the 1st character in the **ReplacementString**. Then it will replace the 2nd character in the **StringToReplace** with the 2nd character in the **ReplacementString** and so on.

Syntax:

```
TRANSLATE(<Str1>, <StringToReplace>, <ReplacementString>)
```

where, **Str1** is the string in which the replacement will take place
StringToReplace is the string that will be searched for in **Str1**
ReplacementString is the string that will be used as the replacement string in **Str1**
All characters in the **StringToReplace** will be replaced with corresponding characters in **ReplacementString**.

Example:

```
SELECT TRANSLATE('1sct523', '123', '7a9') "Change" FROM DUAL;
```

Output:

```
Change
-------
7sct5a9
```

LENGTH: Returns the length of a word.

Syntax:

```
LENGTH(<Word>)
```

Example:

```
SELECT LENGTH('SHARANAM SHAH') "Length" FROM DUAL;
```

Output:

```
    Length
----------
        13
```

LTRIM: Removes characters from the left of char with initial characters removed upto the first character not in set.

Syntax:

```
LTRIM(<Char>[, <Set>])
```

Example:

```
SELECT LTRIM('NISHA','N') "LTRIM" FROM DUAL;
```

Output:

```
LTRIM
-----
ISHA
```

RTRIM: Returns char, with final characters removed after the last character not in the set. '**Set**' is optional, it defaults to spaces.

Syntax:

```
RTRIM(<Char>[, <Set>])
```

Example:

```
SELECT RTRIM('SUNILA','A') "RTRIM" FROM DUAL;
```

Output:

```
RTRIM
-----
SUNIL
```

TRIM: Removes all specified characters either from the beginning or the ending of a string.

Syntax:

```
TRIM([<Leading> | <Trailing> | <Both> [<TrimChar> FROM ] ] <Str1> )
```

where, **Leading** - remove **TrimChar** from the front of **Str1**.
Trailing - remove **TrimChar** from the end of **Str1**.
Both - remove **TrimChar** from the front and end of **Str1**.
If none of the above option is chosen, the **TRIM** function will remove **TrimChar** from both the front and end of **Str1**.
TrimChar is the character that will be removed from **Str1**. If this parameter is omitted, the trim function will remove all leading and trailing spaces from **Str1**.
Str1 is the string to trim.

Example 1:

```
SELECT TRIM(' Chriselle ') "Trim Both" FROM DUAL;
```

Output:

```
Trim Both
---------
Chriselle
```

Example 2:

```
SELECT TRIM(LEADING 'x' FROM 'xxxHanselxxx') "Remove prefixes" FROM DUAL;
```

Output:

```
Remove pr
---------
Hanselxxx
```

Example 3:

```
SELECT TRIM(BOTH 'x' FROM 'xxxHanselxxx') "Remove prefixes N suffixes"
  FROM DUAL;
```

Output:

```
Remove
------
Hansel
```

Example 4:

```
SELECT TRIM(BOTH '1' FROM '123Hansel12111') "Remove string" FROM DUAL;
```

Output:

```
Remove str
----------
23Hanse112
```

LPAD: Returns **Char1**, left-padded to length **n** with the sequence of characters specified in **Char2**. If **Char2** is not specified Oracle uses blanks by default.

Syntax:

```
LPAD(<Char1>, <n> [, <Char2>])
```

Example:

```
SELECT LPAD(CAND_FIRST_NAME, 10, 'x') "RPAD Example" FROM CANDIDATE
  WHERE CAND_FIRST_NAME = 'Ivan';
```

Output:

```
RPAD Examp
----------
xxxxxxIvan
```

RPAD: Returns **Char1**, right-padded to length **n** with the characters specified in **Char2**. If **Char2** is not specified, Oracle uses blanks by default.

Syntax:

```
RPAD(<Char1>, <n> [, <Char2>])
```

Example:

```
SELECT RPAD(CAND_FIRST_NAME, 10, 'x') "RPAD Example" FROM CANDIDATE
  WHERE CAND_FIRST_NAME = 'Ivan';
```

Output:

```
RPAD Examp
----------
Ivanxxxxxx
```

VSIZE: Returns the number of bytes in the internal representation of an expression.

Syntax:

```
VSIZE(<Expression>)
```

Example:

```
SELECT VSIZE('Ivan Nelson Bayross') "Size" FROM DUAL;
```

Output:

```
      Size
----------
        19
```

NVL: Many a times there are records holding null values in a table. When an output of such a table is displayed it is difficult to understand the reason of null or blank values shown in the output.

The only way to overcome this problem is to replace null values with some other meaningful value while outputting the records. This can be done using the NVL function available in oracle.

Syntax:

```
NVL(<ColumnName>, <Value>)
```

It converts a NULL value into a known value. If the first parameter (i.e. the column name) holds a NULL value, the function returns the **value** specified in the second parameter. Otherwise, the **value** held by the column is returned.

Example:

```
SELECT NVL(CAND_FIRST_NAME, 'Not Specified') "First Name",
  NVL(CAND_MIDDLE_NAME, 'Not Specified') "Middle Name",
  NVL(CAND_SURNAME, 'Not Specified') "Last Name" FROM CANDIDATE;
```

Output:

```
First Name   Middle Name     Last Name
------------ --------------- ---------
Sharanam     Chaitanya       Shah
Hansel       Not Specified   Colaco
Ivan         Nelson          Bayross
Chhaya       Not Specified   Bankar
Shravan      Not Specified   Dhone
```

NVL2: Converts a NULL as well as a NOT NULL value into a desired value. If the first parameter (**i.e.** the column name) holds a value, the function returns the **value1** specified in the second parameter. Otherwise, the **value2** specified in the third parameter is returned.

Syntax:

```
NVL2(<ColumnName>, <Value1>, <Value2>)
```

Example:

```
SELECT NVL2(CAND_FIRST_NAME, 'Available', 'Not Available') "First Name",
       NVL2(CAND_MIDDLE_NAME, 'Available', 'Not Available') "Middle Name",
       NVL2(CAND_SURNAME, 'Available', 'Not Available') "Last Name"
  FROM CANDIDATE;
```

Output:

```
First Name     Middle Name    Last Name
-------------- -------------- --------------
Available      Available      Available
Available      Not Available  Available
Available      Available      Available
Available      Not Available  Available
Available      Not Available  Available
```

REPLACE: This function replaces a part of string with the one desired.

Syntax:

```
REPLACE(<Str>, <SearchString>, <ReplaceString>)
```

Example:

```
SELECT REPLACE('Hansel is a bad boy', 'bad', 'good') "Comments" FROM DUAL;
```

Output:

```
Comments
----------------------
Hansel is a good boy
```

TRANSLATE: This function converts the occurrences of characters as desired.

Syntax:

```
TRANSLATE(<Str>, <FromChar>, <ToChar>)
```

Example:
Convert the text Message: ORACLE 10G IS ROCKING into a hidden message:

```
SELECT TRANSLATE('MESSAGE: ORACLE 10G IS ROCKING',
  'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',
  'ZYXWVUTSRQPONMLKJIHGFEDCBA9876543210') "Translated Text" FROM DUAL;
```

REMINDER

The above translation reverses the order of alphabets and digits (i.e. A becomes Z, B becomes Y, while 0 becomes 9 and so on).

Output:

```
Translated Text
------------------------------
NVHHZTV: LIZXOV 89T RH ILXPRMT
```

Retrieve Records Based On Sounds

If it is desired to search all people whose name sounds like:

Neeta or Nita
Meeta or Mita
Suneel or Sunil
Pooja or Puja
Anil or Aneel
Deepa or Dipu or Dipa

This can simply be done using the **Soundex** function of Oracle.

Soundex returns a Character String containing the phonetic representation of another string. The phonetic representation system uses a simple phonetic algorithm to reduce each name to a four character alphanumeric code. The first letter of the code corresponds to the first letter of the last name. The remainder of the code consists of three digits derived from the syllables of the word.

Rules of Soundex:

- Retain the first letter of the string and removes all other occurrences of the following letters: a, e, h, i, o, u, w, y
- Assign numbers to the remaining letters (after the first) as follows:
  b, f, p, v = 1
  c, g, j, k, q, s, x, z = 2
  d, t = 3
  l = 4
  m, n = 5
  r = 6
- If two or more letters with the same number were adjacent in the original name or adjacent except for any intervening h and w, then omit all but the first
- Return the first four bytes padded with 0

Example:
To understand the use of the function, follow the steps mentioned below:

```
CREATE TABLE SCTTEAM (NAME VarChar2(15));
```

Output:

```
Table created.
```

```
INSERT INTO SCTTEAM VALUES ('Neeta');
INSERT INTO SCTTEAM VALUES ('Mita');
INSERT INTO SCTTEAM VALUES ('Dipu');
INSERT INTO SCTTEAM VALUES ('Deepu');
INSERT INTO SCTTEAM VALUES ('Dipa');
INSERT INTO SCTTEAM VALUES ('Anil');
INSERT INTO SCTTEAM VALUES ('Sunil');
```

Output: (For each of the above INSERT INTO statement)

```
1 row created.
```

```
COMMIT;
```

Output:

```
Commit complete.
```

```
SELECT * FROM SCTTEAM;
```

Output:

```
NAME
------
Neeta
Mita
Dipu
Deepu
Dipa
Anil
Sunil
```

Search for a friend whose name sounds like Nita:

```
SELECT * FROM SCTTEAM WHERE SOUNDEX(NAME) = SOUNDEX('Nita');
```

Output:

```
NAME
------
Neeta
```

Search for a friend whose name sounds like Deep:

```
SELECT * FROM SCTTEAM WHERE SOUNDEX(NAME) = SOUNDEX('Deep');
```

Output:

```
NAME
------
Dipu
Deepu
Dipa
```

To understand the working of the function issue the following:

```
SELECT SOUNDEX(NAME), NAME, SOUNDEX('DEEP') FROM SCTTEAM;
```

Output:

```
SOUNDEX(NAME) NAME            SOUNDEX(DEEP)
------------- --------------- -------------
N300          Neeta           D100
M300          Mita            D100
D100          Dipu            D100
D100          Deepu           D100
D100          Dipa            D100
A540          Anil            D100
S540          Sunil           D100

7 rows selected.
```

The output shows how Oracle converts the words into numbers followed by a comparison between the two words and finally displays the output for the matching ones.

Conversion Functions

TO_NUMBER: Converts **char**, a **CHARACTER** value expressing a number, to a NUMBER datatype.

Syntax:

```
TO_NUMBER(<Char>)
```

Example:

```
UPDATE CAND_EMPL SET SAL_ON_LEAVING
  = SAL_ON_LEAVING + TO_NUMBER(SUBSTR('$500',2,3));
```

Output:

```
7 rows updated.
```

REMINDER

Here, the value 500 will be added to every salary on leaving in the **CAND_EMPL** table.

TO_CHAR (number conversion): Converts a value of a **NUMBER** datatype to a **character** datatype, using the optional format string. TO_CHAR() accepts a number (**n**) and a numeric format (**fmt**) in which the number has to appear. If **fmt** is omitted, **n** is converted to a char value exactly long enough to hold all significant digits.

Syntax:

```
TO_CHAR (n[,fmt])
```

Example:

```
SELECT TO_CHAR(17145, '$099,999') "Char" FROM DUAL;
```

Output:

```
Char
--------
$017,145
```

TO_CHAR (Date Conversion): Converts a value of a **DATE** datatype to **CHAR** value. TO_CHAR() accepts a date, as well as the format (fmt) in which the date has to appear. **fmt** must be a date format. If **fmt** is omitted, the **date** is converted to a character value using the default date format, **i.e.** "DD-MON-YY".

Syntax:

```
TO_CHAR(date[,fmt])
```

Example:

```
SELECT TO_CHAR(SYSDATE, 'Month DD, YYYY') "New Date Format" FROM DUAL;
```

Output:

```
New Date Format
------------------
March     20, 2005
```

Translating Numeric Values To Character Equivalents

Example:
There are times when **amounts** in an application have to be represented by their character values. Specially while printing bank drafts the system gives the draft amount in **numbers** and **words**. In Oracle this can be simply done using the Julian Date conversion as follows:

Solution 1: FOR **UPPER-CASE** LETTERS

```
SELECT TO_CHAR(TO_DATE(34654, 'J'), 'JSP') FROM DUAL;
```

Output:

```
TO_CHAR(TO_DATE(34654,'J'),'JSP')
-----------------------------------------
THIRTY-FOUR THOUSAND SIX HUNDRED FIFTY-FOUR
```

Solution 2: FOR **TITLE-CASE** LETTERS

```
SELECT TO_CHAR(TO_DATE(34654, 'J'), 'JsP') FROM DUAL;
```

Output:

```
TO_CHAR(TO_DATE(34654,'J'),'JSP')
-----------------------------------------
Thirty-Four Thousand Six Hundred Fifty-Four
```

Solution 3: FOR **LOWER-CASE** LETTERS

```
SELECT TO_CHAR(TO_DATE(34654, 'J'), 'jSP') FROM DUAL;
```

Output:

```
TO_CHAR(TO_DATE(34654,'J'),'JSP')
-----------------------------------------
thirty-four thousand six hundred fifty-four
```

Note that:

- The minimum JULIAN number allowed is 1 and the maximum JULIAN number allowed is 5373484
- Amount larger then the maximum JULIAN number allowed cannot be converted to words
- This only works for integer amounts

- If rupees and paise are also required in the output then split up the amount into it's integer and decimal parts and handle the case of a zero amount, such as:

```
SELECT 'Rupees ' || DECODE(TRUNC(34654.23), 0, 'ZERO',
  TO_CHAR(TO_DATE(TRUNC(34654.23), 'J'), 'JSP')) || ' AND ' ||
  DECODE(TRUNC(MOD(34654.23, 1)*100), 0, 'ZERO',
  TO_CHAR(TO_DATE(TRUNC(MOD(34654.23,1)*100), 'J'), 'JSP')) || ' Paise'
  FROM DUAL;
```

Output:

```
'RUPEES'||DECODE(TRUNC(34654.23),0,'ZERO',TO_CHAR(TO_DATE(TRUNC
(34654.23)
------------------------------------------------------------------
----------
Rupees THIRTY-FOUR THOUSAND SIX HUNDRED FIFTY-FOUR AND TWENTY-
THREE Paise
```

SYSDATE

SYSDATE is a **pseudo** column that contains the current date and time. It requires no arguments when used with the table DUAL. It returns the current date.

Example 10:

```
SELECT SYSDATE FROM DUAL;
```

Output:

```
SYSDATE
---------
20-MAR-05
```

Date Functions

Oracle's DATE data type is used to store date and time information. It has special properties associated with it. The data type stores information about century, year, month, day, hour, minute and second for **each** date value.

Any date value stored in a column of a DATE data type **is always** stored in a specific **default** format. The default format is 'DD-MON-YY HH:MI:SS'. Hence, when a date has to be inserted in a date field, its value has to be specified in exactly the same format. The values held in DATE columns are always displayed in the **default** format when **retrieved** from the table.

If data from a date column has to be viewed in any other format (other than the default format), Oracle provides the **TO_DATE** function that can be used to specify the required format.

The same function can also be used for **storing a date** into a DATE column in a specific format other than Oracle's default. This can be done by specifying the date value and **the format** in which it is to be stored. The TO_DATE() function also allows part insertion of a DATE value into a column, for example, only the day and month portion of the date value.

To enter the time portion (i.e. HH:MI:SS) of a date, the TO_DATE function must be used with a **format mask** indicating the time portion.

To understand the working of the Date function in Oracle create a table named CAND_PRSN_INFO base on the following description:

Table Definition:

Table Name : **CAND_PRSN_INFO**
Primary Key : CAND_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_USER_ID	VARCHAR2	7	NOT NULL	
CAND_ID	VARCHAR2	7		
CAND_FIRST_NAME	VARCHAR2	35		
CAND_SURNAME	VARCHAR2	35		NULL
CAND_BIRTHDATE	DATE		NOT NULL	
CAND_BIRTHPLACE	VARCHAR2	35		NULL

The following statement will create the CAND_PRSN_INTO table:

```
CREATE TABLE CAND_PRSN_INFO (CAND_USER_ID VARCHAR2(7) NOT NULL,
  CAND_ID VARCHAR2(7), CAND_FIRST_NAME VARCHAR2(35),
  CAND_SURNAME VARCHAR2(35) DEFAULT NULL,
  CAND_BIRTHDATE DATE NOT NULL,
  CAND_BIRTHPLACE VARCHAR2(35) DEFAULT NULL,
  PRIMARY KEY (CAND_ID));
```

TO_DATE: Converts a character field to a date field.

Syntax:

```
TO_DATE(char [, fmt])
```

Example:
Insert the following information into the CAND_PRSN_INTO table:

CAND_USER_ID	CAND_ID	CAND_FIRST_NAME	CAND_SURNAME	CAND_BIRTH DATE	CAND_BIRTH PLACE
US00001	CD00001	Sharanam	Shah	03/01/1981	Mumbai
US00002	CD00002	Hansel	Colaco	01/01/1982	Mumbai
US00003	CD00003	Ivan	Bayross	23/06/1952	Mumbai
US00004	CD00004	Chhaya	Bankar	22/06/1976	Mumbai

Solution:

```
INSERT INTO CAND_PRSN_INFO VALUES ('US00001', 'CD00001', 'Sharanam',
  'Shah', TO_DATE('03/01/1981', 'DD/MM/YYYY'), 'Mumbai');
INSERT INTO CAND_PRSN_INFO VALUES ('US00002', 'CD00002', 'Hansel',
  'Colaco', TO_DATE('01/01/1982', 'DD/MM/YYYY'), 'Mumbai');
INSERT INTO CAND_PRSN_INFO VALUES ('US00003', 'CD00003', 'Ivan',
  'Bayross', TO_DATE('23/06/1952', 'DD/MM/YYYY'), 'Mumbai');
INSERT INTO CAND_PRSN_INFO VALUES ('US00004', 'CD00004', 'Chhaya',
  'Bankar', TO_DATE('22/06/1976', 'DD/MM/YYYY'), 'Mumbai');
```

Output for each of the above INSERT INTO statements:

```
1 row created.
```

To manipulate and extract values from the date column of a table Oracle provides date functions. Some are discussed below:

ADD_MONTHS: Returns a date after adding the number of months specified in the function.

Syntax:

```
ADD_MONTHS(d, n)
```

Example:

```
SELECT ADD_MONTHS(SYSDATE, 5) "Add Months" FROM DUAL;
```

Output:

```
Add Month
---------
20-AUG-05
```

LAST_DAY: Returns the last date of the month specified with the function.

Syntax:

```
LAST_DAY(d)
```

Example:

```
SELECT SYSDATE, LAST_DAY(SYSDATE) "LastDay" FROM DUAL;
```

Output:

```
SYSDATE   LastDay
--------- ---------
20-MAR-05 31-MAR-05
```

MONTHS_BETWEEN: Returns the number of months between **d1** and **d2**.

Syntax:

```
MONTHS_BETWEEN(d1, d2)
```

Example:

```
SELECT MONTHS_BETWEEN('02-MAY-04', '02-JAN-02') "Months" FROM DUAL;
```

Output:

```
    Months
----------
        28
```

NEXT_DAY: Returns the date of the first weekday described as **char** that is after the date specified in **date. char must be a day of the week**.

Syntax:

```
NEXT_DAY(date, char)
```

Example:

```
SELECT NEXT_DAY('09-APRIL-05', 'Saturday') "NEXT DAY" FROM DUAL;
```

Output:

```
NEXT DAY
---------
16-APR-05
```

ROUND: Returns a date rounded to a specific unit of measure. If the second parameter is omitted, the **ROUND** function will round the date to the nearest day.

Syntax:

```
ROUND(date, [format])
```

Below are the valid format parameters:

Unit	Format parameters	Rounding Rule
Year	SYYYY, YYYY, YEAR, SYEAR, YYY, YY, Y	Rounds up on July 1st
ISO Year	IYYY, IY, I	
Quarter	Q	Rounds up on the 16th day of the second month of the quarter
Month	MONTH, MON, MM, RM	Rounds up on the 16th day of the month
Week	WW	Same day of the week as the first day of the year
IW	IW	Same day of the week as the first day of the ISO year
W	W	Same day of the week as the first day of the month
Day	DDD, DD, J	
Hour	HH, HH12, HH24	
Start day of the week	DAY, DY, D	
Minute	MI	

Example:

```
SELECT ROUND(TO_DATE('01-DEC-04'), 'YYYY') "Year" FROM DUAL;
```

Output:

```
Year
---------
01-JAN-05
```

NEW_TIME: Returns the date after converting it from **time zone1** to a date in **time zone2**.

Syntax:

```
NEW_TIME(date, zone1, zone2)
```

Value	Description	Value	Description	Value	Description
AST	Atlantic Standard Time	ADT	Atlantic Daylight Time	BST	Bering Standard Time
BDT	Bering Daylight Time	CST	Central Standard Time	CDT	Central Daylight Time
EST	Eastern Standard Time	EDT	Eastern Daylight Time	GMT	Greenwich Mean Time
HST	Alaska-Hawaii Standard Time	HDT	Alaska-Hawaii Daylight Time	MST	Mountain Standard Time
MDT	Mountain Daylight Time	NST	Newfoundland Standard Time	PST	Pacific Standard Time
PDT	Pacific Daylight Time	YST	Yukon Standard Time	YDT	Yukon Daylight Time

Example:
The following example converts an Atlantic Standard Time into a Mountain Standard Time:

```
SELECT NEW_TIME(TO_DATE('2005/05/01 01:45', 'yyyy/mm/dd HH24:MI'), 'AST',
                'MST') "MST" FROM DUAL;
```

Output:

```
MST
---------
30-APR-05
```

REMINDER

Several other Date function are available in Oracle. These include the following:
- **DbTimeZone(), SessionTimeZone(), SysTimestamp(), Tz_Offset()**

The above Oracle date functions are **just a few** selected from the **many date** functions that are built into Oracle. These Oracle functions are commonly used in commercial application development.

Add A Day/Hour/Minute/Second To A Date Value

The **SYSDATE** pseudo-column shows the current system date and time. Adding 1 to SYSDATE will advance the date by 1 day. Using fractions to add hours, minutes or seconds to the date can advance the date by hour, minutes and seconds.

Example:

```
SELECT TO_CHAR(SYSDATE, 'DD-MON-YYYY HH:MI:SS') "Date",
  TO_CHAR(SYSDATE+1, 'DD-MON-YYYY HH:MI:SS') "By 1 Day",
  TO_CHAR(SYSDATE+1/24, 'DD-MON-YYYY HH:MI:SS') "By 1 Hour",
  TO_CHAR(SYSDATE+1/1440, 'DD-MON-YYYY HH:MI:SS') "By 1 Minute",
  TO_CHAR(SYSDATE+ 1/86400 , 'DD-MON-YYYY HH:MI:SS') "By 1 Second"
  FROM DUAL;
```

Output:

```
Date                 By 1 Day             By 1 Hour
By 1 Minute          By 1 Second
-------------------- -------------------- --------------------
-------------------- --------------------
28-JAN-2005 11:43:52 29-JAN-2005 11:43:52 28-JAN-2005 12:43:52
28-JAN-2005 11:44:52 28-JAN-2005 11:43:53
```

Some more forms of date additions:

Description	Code
Now	SYSDATE
Tomorrow/ next day	SYSDATE + 1
Seven days from now	SYSDATE + 7
One hour from now	SYSDATE + 1/24
Three hours from now	SYSDATE + 3/24
An half hour from now	SYSDATE + 1/48
10 minutes from now	SYSDATE + 10/1440
30 seconds from now	SYSDATE + 30/86400
Tomorrow at 12 midnight	TRUNC(SYSDATE + 1)
Tomorrow at 8 AM	TRUNC(SYSDATE + 1) + 8/24
Next Monday at 12:00 noon	NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 12/24
First day of the month at 12 midnight	TRUNC(LAST_DAY(SYSDATE) + 1)
The next Monday, Wednesday or Friday at 9 A.M	TRUNC(LEAST(NEXT_DAY(SYSDATE, 'MONDAY'), NEXT_DAY(SYSDATE, 'WEDNESDAY'), NEXT_DAY(SYSDATE, 'FRIDAY'))) + (9/24)

Manipulating Dates In SQL Using DATE()

A column of data type **Date** is always displayed in a default format, which is **'DD-MON-YY'**. If this default format is not used when entering data into a column of the **date** data type, Oracle **rejects the data** and returns an error message.

If a **date** has to be retrieved or inserted into a table in a format **other than** the default one, Oracle provides the **TO_CHAR** and **TO_DATE** functions to do this.

TO_CHAR

The TO_CHAR function facilitates the retrieval of data in a format different from the default format. It can also extract a part of the date, **i.e.** the date, month or the year from the date value and use it for sorting or grouping of data according to the date, month or year.

Syntax:

```
TO_CHAR(<Date Value> [,<Fmt>])
```

where **Date Value** stands for the date and **Fmt** is the specified format in which date is to be displayed.

Example 1:

```
SELECT TO_CHAR(SYSDATE, 'DD-MM-YY') FROM DUAL;
```

Output:

```
TO_CHAR(
--------
20-03-05
```

TO_DATE

TO_DATE converts a **CharValue** into a **date** value. It allows a user to insert date into a date column in any required format, by specifying the **CharValue** of the date to be inserted and its format.

Syntax:

```
TO_DATE(<CharValue>[,<Fmt>])
```

where **CharValue** stands for the value to be inserted in the date column and **Fmt** is a date format in which the **CharValue** is specified.

Example 2:

```
SELECT TO_DATE ('21/07/05', 'DD/MM/YY') FROM DUAL;
```

Output:

```
TO_DATE('
---------
21-JUL-05
```

Example 3:

List the candidates born after the year 1980. The birthdates should be displayed in **'DD/MM/YY'** format.

Synopsis:

Tables:	CAND_PRSN_INFO
Columns:	CAND_ID, CAND_FIRST_NAME, CAND_SURNAME, CAND_BIRTHDATE
Technique:	**Functions:** TO_CHAR(); **Clauses:** WHERE; Others: || (Concatenate), Alias

Solution:

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Name",
  TO_CHAR(CAND_BIRTHDATE, 'DD/MM/YY') "BirthDate" FROM CAND_PRSN_INFO
  WHERE TO_CHAR(CAND_BIRTHDATE, 'YYYY') > '1980';
```

Output:

```
CAND_ID Name              BirthDate
------- ----------------- ---------
CD00001 Sharanam Shah      03/01/81
CD00002 Hansel Colaco      01/01/82
```

Explanation:
Here the value held in the **CAND_BIRTHDATE** field is formatted using the **TO_CHAR()** function to display the date in the **DD/MM/YY** format.

Example 4:
Insert the following data in the table **CAND_PRSN_INFO**, where the **time component** has to be stored along with the date in the column **CAND_BIRTHDATE.**

```
INSERT INTO CAND_PRSN_INFO VALUES ('US00006', 'CD00005', 'Shravan',
  'Dhone', TO_DATE('21/02/1982 04:30:00', 'DD/MM/YYYY hh:mi:ss'), 'Thane');
```

Output:

```
1 row created.
```

Special Date Formats Using TO_CHAR function

Sometimes, the date value is required to be displayed in special formats, for example, instead of 01-JAN-05, displays the date as 01st of January, 2005. For this, Oracle provides **special attributes**, which can be used in the format specified with the **TO_CHAR** and **TO_DATE** functions. The significance and use of these characters are explained in the examples below.

All three examples below are based on the **CAND_PRSN_INFO** table

The query is as follows:

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Name",
  TO_CHAR(CAND_BIRTHDATE, 'DD/MM/YY') "BirthDate" FROM CAND_PRSN_INFO;
```

Output:

```
CAND_ID Name                BirthDat
------- ------------------- --------
CD00001 Sharanam Shah       03/01/81
CD00002 Hansel Colaco       01/01/82
CD00003 Ivan Bayross        23/06/52
CD00004 Chhaya Bankar       22/06/76
CD00005 Shravan Dhone       21/02/82
```

Variations in the output, mentioned earlier, can be achieved as follows:

Use Of TH In The TO_CHAR() Function

DDTH places TH, RD, ND for the date (DD), for example, 02ND, 03RD, 08TH etc

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Name",
  TO_CHAR(CAND_BIRTHDATE, 'DDTH-MON-YY') "BirthDate"
  FROM CAND_PRSN_INFO;
```

Output:

```
CAND_ID Name                BirthDate
------- ------------------ -----------
CD00001 Sharanam Shah      03RD-JAN-81
CD00002 Hansel Colaco      01ST-JAN-82
CD00003 Ivan Bayross       23RD-JUN-52
CD00004 Chhaya Bankar      22ND-JUN-76
CD00005 Shravan Dhone      21ST-FEB-82
```

Use Of SP In The TO_CHAR() Function

DDSP indicates that the date (DD) must be displayed by spelling the date such as ONE, TWELVE etc.

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Name",
  TO_CHAR(CAND_BIRTHDATE, 'DDSP') "BirthDate" FROM CAND_PRSN_INFO;
```

Output:

```
CAND_ID Name                BirthDate
------- ------------------ ------------
CD00001 Sharanam Shah      THREE
CD00002 Hansel Colaco      ONE
CD00003 Ivan Bayross       TWENTY-THREE
CD00004 Chhaya Bankar      TWENTY-TWO
CD00005 Shravan Dhone      TWENTY-ONE
```

Use Of SPTH In The TO_CHAR() Function

SPTH displays the date (DD) with **th** added to the spelling fourteen**th**, twelf**th**.

```
SELECT CAND_ID, (CAND_FIRST_NAME || ' ' || CAND_SURNAME) "Name",
  TO_CHAR(CAND_BIRTHDATE, 'DDSPTH') "BirthDate" FROM CAND_PRSN_INFO;
```

Output:

```
CAND_ID Name                BirthDate
------- ------------------ --------------
CD00001 Sharanam Shah      THIRD
CD00002 Hansel Colaco      FIRST
CD00003 Ivan Bayross       TWENTY-THIRD
CD00004 Chhaya Bankar      TWENTY-SECOND
CD00005 Shravan Dhone      TWENTY-FIRST
```

Using Time Intervals

Intervals have been recently introduced as data types that allow the storage of Time Intervals. These time intervals include periods of time such as 2 years and 10 months, 7 hours and 3 minutes, - 4 months and 4 days, and so on.

REMINDER

TIME INTERVALS are not to be confused with DATETIME or TIMESTAMPS.

The Time Interval data types are of two forms:

- **INTERVAL YEAR [(years_precision)] TO MONTH:** Stores a time interval measured in years and months. The **years_precision** variable is an optional precision for the years, which may be an integer from **0** to **9**. The **year_precision** variable determines the number of digits available for storing the years in the interval. The default value for the **years_precision** is **2**. An error is encountered, if attempts are made to store more year digits than the INTERVAL YEAR TO MONTH column can store.
- **INTERVAL DAY [(days_precision)] TO SECOND [(seconds_precision)]:** Stores a time interval measured in days and seconds. The **days_precision** variable is an optional precision for the days, an integer ranging from **0** to **9**. Its default value is **2**. Additionally, the **seconds_precision** variable is an optional precision for the fractional seconds, an integer ranging from **0** to **9**. Its default value is **6**.

Using The INTERVAL YEAR TO MONTH Type

Example 1:
The following statement creates a table named **EMPDURATION** that stores the employment details such as employer, employment date and the duration of employment:

```
CREATE TABLE EMPDURATION (CAND_ID VarChar2(10), EMPNO Number(2),
  EMPDT TIMESTAMP, DURATION INTERVAL YEAR(2) TO MONTH);
```

The format for supplying an INTERVAL YEAR TO MONTH literal value is as follows:

```
INTERVAL '[+|-] [y] [m]' [YEAR [(year_precision)]] [TO MONTH]
```

where,

- the **+** (plus) sign or the **-** (minus) sign is an optional indicator that specifies whether the time interval is positive or negative (default is positive),
- **y** is the optional number of years for the interval,
- **m** is the optional number of months for the interval. **TO MONTH** is included in instances where the years and months are included in the literal,
- **years_precision** is the optional precision for the year (default is 2).

The following statements populate the table named **EMPDURATION:**

```
INSERT INTO EMPDURATION (CAND_ID, EMPNO, EMPDT, DURATION)
  VALUES('CD0001', 1, TIMESTAMP '2005-01-01 09:45:55.1245',
     INTERVAL '2-4' YEAR TO MONTH);
INSERT INTO EMPDURATION (CAND_ID, EMPNO, EMPDT, DURATION)
  VALUES('CD0003', 2, TIMESTAMP '2005-01-21 19:05:00.0', INTERVAL '1'
YEAR);
```

The following statement describes the table named **EMPDURATION:**

```
DESC EMPDURATION;
```

Output:

```
Name        Null?    Type
----------- -------- ------
CAND_ID              VARCHAR2(10)
EMPNO                NUMBER(2)
EMPDT                TIMESTAMP(6)
DURATION             INTERVAL YEAR(2) TO MONTH
```

The following statement displays the contents of the table named **EMPDURATION:**

```
SELECT * FROM EMPDURATION;
```

Output:

```
CAND_ID EMPNO EMPDT                           DURATION
------- ----- ------------------------------- ---------
CD0001      1 01-JAN-05 09.45.55.124500 AM    +02-04
CD0003      2 21-JAN-05 07.05.00.000000 PM    +01-00
```

Using The INTERVAL YEAR TO MONTH Type

Example 2:
The following statement creates a table named **EMPDURATIONSECS** that stores the employment details such as employer, employment date and the duration of employment:

```
CREATE TABLE EMPDURATIONSECS (CAND_ID VarChar2(10), Empno Number(2),
  EMPDT TIMESTAMP, DURATION INTERVAL DAY(2) TO SECOND(6));
```

The format for supplying an INTERVAL DAY TO SECOND literal value is as follows:

```
  INTERVAL '[+|-] [d] [h[:m[:s]]]' [DAY [(days_precision)]]
     [TO HOURS | MINUTE | SECOND[(seconds_precision)]]
```

where,

- the + (plus) sign or the - (minus) sign is an optional indicator that specifies whether the time interval is positive or negative (default is positive),

- **d** is the optional number of days for the interval,
- **h** is the optional number of hours for the interval. The **TO HOUR** is included in instances where the days and hours are included in the literal,
- **m** is the optional number of minutes for the interval. The **TO MINUTES** is included in instances where the days and minutes are included in the literal,
- **s** is the optional number of seconds for the interval. The **TO SECOND** is included in instances where the days and seconds are included in the literal,
- **days_precision** is the optional precision for the days (default is 2),
- **seconds_precision** is the optional precision for the fractional seconds (default is 6).

The following statements populate the table named **EMPDURATIONSECS:**

```
INSERT INTO EMPDURATIONSECS (CAND_ID, EMPNO, EMPDT, DURATION)
  VALUES('CD0002', 3, TIMESTAMP '2005-01-05 10:00:00',
      INTERVAL '2 4:20:45.785' DAY TO SECOND);
INSERT INTO EMPDURATIONSECS (CAND_ID, EMPNO, EMPDT, DURATION)
  VALUES('CD0005', 4, TIMESTAMP '2005-01-17 19:05:00.0', INTERVAL '6'
      HOUR);
```

The following statement describes the table named **EMPDURATIONSECS:**

```
DESC EMPDURATIONSECS;
```

Output:

```
Name          Null?    Type
------------ -------- ----------------------------
CAND_ID                VARCHAR2(10)
EMPNO                  NUMBER(2)
EMPDT                  TIMESTAMP(6)
DURATION               INTERVAL DAY(2) TO SECOND(6)
```

The following statement displays the contents of the table named **EMPDURATIONSECS:**

```
SELECT * FROM EMPDURATIONSECS;
```

Output:

```
CAND_ID EMPNO EMPDT                          DURATION
------- ----- ---------------------------- ------------------
CD0002      3 05-JAN-05 10.00.00.000000 AM  +02 04:20:45.785000
CD0005      4 17-JAN-05 07.05.00.000000 PM  +00 06:00:00.000000
```

Time Interval Related Functions

Oracle offers multiple functions that allow getting and processing time intervals. They are:

NUMTODSINTERVAL (x, interval_unit): This function converts the number **x** to an INTERVAL DAY TO SECOND with the interval for **x** supplied in **interval_unit**, which may be set to DAY, HOUR, MINUTE or SECOND.

Example:
The following statement displays the use of the **NUMTODSINTERVAL()** function:

```
SELECT NUMTODSINTERVAL(2.25, 'DAY'),
  NUMTODSINTERVAL(100.34225, 'SECOND') FROM DUAL;
```

Output:

```
NUMTODSINTERVAL(2.25,'DAY')    NUMTODSINTERVAL(100.34225,'SECOND')
------------------------------ ---------------------------------
+000000002 06:00:00.000000000 +000000000 00:01:40.342250000
```

NUMTOYMINTERVAL (x, interval_unit): This function converts the number **x** to a YEAR TO MONTH with the interval for **x** supplied in **interval_unit**, which may be set to YEAR or MONTH.

Example:
The following statement displays the use of the **NUMTOYMINTERVAL()** function:

```
SELECT NUMTOYMINTERVAL(3.5, 'YEAR'),
  NUMTOYMINTERVAL(30.34225, 'MONTH') FROM DUAL;
```

Output:

```
NUMTOYMINTERVAL(3.5,'YEAR')   NUMTOYMINTERVAL(30.34225,'MONTH')
----------------------------- ---------------------------------
+000000003-06                 +000000002-06
```

System Functions

UID: This function returns an integer value corresponding to the UserID of the user currently logged in.

Syntax:

```
UID [INTO <variable>]
```

where, **variable** will now contain the id number for the user's session.

Example:

```
SELECT UID FROM DUAL;
```

Output:

```
       UID
----------
        64
```

USER: This function returns the **user name** of the user who has logged in. The value returned is in varchar2 data type.

Syntax:

```
USER
```

Example:

```
SELECT USER FROM DUAL;
```

Output:

```
USER
------------
DBA_PM
```

SYS_CONTEXT: Can be used to retrieve information about Oracle's environment.

Syntax:

```
SYS_CONTEXT (<namespace>, <parameter>, [<length>])
```

where,

- **namespace** is an Oracle namespace that has already been created. If the **namespace** of **USERENV** is used, attributes describing the current Oracle session can be returned.
- **parameter** is a valid attribute that has been set using the DBMS_SESSION.set_context procedure.
- **length** is the length of the return value in bytes. If this parameter is omitted or if an invalid entry is provided, the **SYS_CONTEXT** function will default to **256 bytes**.

The valid parameters for the namespace called **USERENV** are as follows:

Parameter	Explanation	Return Length
AUDITED_CURSORID	Returns the cursor ID of the SQL that triggered the audit	N/A
AUTHENTICATION_DATA	Authentication data	256
AUTHENTICATION_TYPE	Describes how the user was authenticated. Can be one of the following values: Database, OS, Network or Proxy	30
BG_JOB_ID	If the session was established by an Oracle background process, this parameter will return the Job ID. Otherwise, it will return NULL.	30

Parameter	Explanation	Return Length
CLIENT_IDENTIFIER	Returns the client identifier (global context)	64
CLIENT_INFO	User session information	64
CURRENT_SCHEMA	Returns the default schema used in the current schema	30
CURRENT_SQL	Returns the SQL that triggered the audit event	64
CURRENT_USER	Name of the current user	30
CURRENT_USERID	Userid of the current user	30
DB_NAME	Name of the database from the DB_NAME initialization parameter	30
ENTRYID	Available auditing entry identifier	30
EXTERNAL_NAME	External of the database user	256
HOST	Name of the host machine from which the client has connected	54
CURRENT_SCHEMAID	Returns the identifier of the default schema used in the current schema	30
DB_DOMAIN	Domain of the database from the DB_DOMAIN initialization parameter	256
FG_JOB_ID	If the session was established by a client foreground process, this parameter will return the Job ID. Otherwise, it will return NULL.	30
GLOBAL_CONTEXT _MEMORY	The number used in the System Global Area by the globally accessed context	N/A
INSTANCE	The identifier number of the current instance	30
IP_ADDRESS	IP address of the machine from which the client has connected	30
ISDBA	Returns TRUE if the user has DBA privileges. Otherwise, it will return FALSE.	30
LANG	The ISO abbreviate for the language	62
LANGUAGE	The language, territory and character of the session. In the following format: language_territory.characterset	52
NETWORK_PROTOCOL	Network protocol used	256
NLS_CALENDAR	The calendar of the current session	62
NLS_CURRENCY	The currency of the current session	62
NLS_DATE_FORMAT	The date format for the current session	62
NLS_DATE_ LANGUAGE	The language used for dates	62
NLS_SORT	BINARY or the linguistic sort basis	62
NLS_TERRITORY	The territory of the current session	62
OS_USER	The OS username for the user logged in	30
PROXY_USER	The name of the user who opened the current session on behalf of SESSION_USER	30
PROXY_USERID	The identifier of the user who opened the current session on behalf of SESSION_USER	30

Parameter	Explanation	Return Length
SESSION_USER	The database user name of the user logged in	30
SESSION_USERID	The database identifier of the user logged in	30
SESSIONID	The identifier of the auditing session	30
TERMINAL	The OS identifier of the current session	10

Example:

```
SELECT SYS_CONTEXT('USERENV', 'NLS_DATE_FORMAT') "SysContext"
  FROM DUAL;
```

Output:

```
SysContext
----------
DD-MON-RR
```

USERENV: Can be used to retrieve information about the current Oracle session. Although this function still exists in Oracle for backwards compatibility, it is recommended that the **SYS_CONTEXT** function is used instead.

Syntax:

```
USERENV(<Parameter>)
```

where, **Parameter** is the value to return from the current Oracle session.

The possible values are:

Parameter	Explanation
CLIENT_INFO	Returns user session information stored using the DBMS_APPLICATION_INFO package
ENTRYID	Available auditing entry identifier
INSTANCE	The identifier number of the current instance
ISDBA	Returns TRUE if the user has DBA privileges. Otherwise, it will return FALSE.
LANG	The ISO abbreviate for the language
LANGUAGE	The language, territory and character of the session. In the following format: language_territory.characterset
SESSIONID	The identifier of the auditing session
TERMINAL	The OS identifier of the current session

Example:

```
SELECT USERENV('LANGUAGE') FROM DUAL;
```

Output:

```
USERENV('LANGUAGE')
------------------------------
AMERICAN_AMERICA.WE8MSWIN1252
```

COALESCE: Returns the first non-null expression in the list. If all expressions evaluate to null, then the **coalesce** function will return null.

Syntax:

```
COALESCE(<expr1>, <expr2>, ... <expr_n>)
```

Example:

```
SELECT CAND_ID, CAND_ADDR_ID,
  COALESCE(CAND_MOB, 'Land Line = '|| CAND_TEL) Mobile FROM CAND_ADDR;
```

The above coalesce statement is equivalent to the following IF-THEN-ELSE statement:

```
IF CAND_MOB IS NOT NULL THEN
  Mobile := CAND_MOB;
ELSIF CAND_TEL IS NOT NULL THEN
  Mobile := Land Line = CAND_TEL;
ELSE
  Mobile := NULL;
END IF;
```

Output:

```
CAND_ID CAND_AD MOBILE
------- ------- ----------------------------------
CD00001 CA00001 Land Line = 28712020
CD00001 CA00002 Land Line = 28554099 / 28879221-25
CD00002 CA00003 9821531788
CD00003 CA00004 9821531788
CD00003 CA00005 Land Line = 2439162
CD00004 CA00006 Land Line =
CD00004 CA00007 Land Line = 022 27543418
CD00005 CA00008 Land Line = 04926 260300

8 rows selected.
```

Explanation:
In the above example, Oracle will display the mobile number **i.e.** the value held in the field **CAND_MOB**, if **CAND_MOB** field holds a value. If it does not hold a value, then Oracle will move on to the next column in the **COALESCE** function and display the value held in the next column **i.e. CAND_TEL**, if it holds a value.

In case the second column also does not hold a value, then Oracle will display **NULL** as an output.

Grouping Retrieved Data

The Concept Of Grouping

Till now, all SQL **SELECT** statements have:

- Retrieved all the rows from tables
- Retrieved selected rows from tables with the use of a **WHERE** clause, which returns only those rows that meet the conditions specified
- Retrieved unique rows from the table, with the use of **DISTINCT** clause
- Retrieved rows in the sorted order **i.e.** ascending or descending order, as specified, with the use of **ORDER BY** clause.

There are two other clauses, other than the above, which facilitate selective retrieval of rows. They are the **GROUP BY** and **HAVING** clauses. These are parallel to the **order by** and **where** clause, except that they **act on record sets** and **not on** individual records held in a table.

GROUP BY Clause

The **GROUP BY** clause is another section of the **Select** statement. This optional clause tells Oracle to group rows based on **distinct values** that exist for **specified columns**. The GROUP BY clause creates a data set, containing several sets of records **grouped together** based on a condition.

Syntax:

```
SELECT <ColumnName1>, <ColumnName2>, <ColumnNameN>,
      AGGREGATE_FUNCTION (<Expression>)
  FROM TableName WHERE <Condition>
      GROUP BY <ColumnName1>, <ColumnName2>, <ColumnNameN>;
```

Example 1:
Find out how many times each candidate has been employed.

Synopsis:

Tables:	CAND_EMPL
Columns:	CAND_ID, EMPL_ID
Technique:	**Functions:** COUNT(); **Clauses:** GROUP BY; **Others:** Alias

Solution:

```
SELECT CAND_ID "Candidate No.", COUNT(EMPL_ID) "No. Of Jobs" FROM
  CAND_EMPL GROUP BY CAND_ID;
```

Output:

```
Candida No. Of Jobs
------- -----------
CD00001           1
CD00002           2
CD00003           1
CD00004           1
CD00005           2
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_EMPL** table. Since the **number of jobs** per candidate is required, the records need to be **grouped** on the basis of field **CAND_ID** and then the **COUNT()** function must be applied to the field **EMPL_ID** which calculates the number of jobs on a per candidate basis.

Count Different Data Values In A Column

Sometimes it is required to count the value held in a column that is different.

Example 2:
Count the number of addresses provided per Candidate.

```
SELECT CAND_ID, COUNT(*) "Addresses" FROM CAND_ADDR GROUP BY
  CAND_ID;
```

Output:

```
CAND_ID  Addresses
------- ----------
CD00001          2
CD00002          1
CD00003          2
CD00004          2
CD00005          1
```

HAVING Clause

The **HAVING** clause must be used only in conjunction with a **GROUP BY** clause. **HAVING** imposes a condition on the **GROUP BY** clause, which further filters the groups created by the **GROUP BY** clause. Each column specification specified in the HAVING clause must occur within a statistical function or must occur in the list of columns named in the GROUP BY clause.

Example 5:
Find out the candidates who have changed jobs more than once.

Synopsis:

Tables:	CAND_EMPL
Columns:	CAND_ID, EMPL_ID
Technique:	**Functions:** COUNT(); **Operators:** > (Greater Than); **Clauses:** GROUP BY ... HAVING; **Others:** Alias

Solution:

```
SELECT CAND_ID "Candidate No.", COUNT(EMPL_ID) "No. Of Jobs" FROM
  CAND_EMPL GROUP BY CAND_ID HAVING COUNT(EMPL_ID) > 1;
```

Output:

```
Candida  No. Of Jobs
-------  -----------
CD00002            2
CD00005            2
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_EMPL** table. The **COUNT()** function is applied to the field **EMPL_ID**. This filtered information is then **grouped** on the basis of Candidate Id (i.e. the **CAND_ID** field). Since only those customers who have changed jobs more than once are to be retrieved, the **HAVING** clause is used to finally filter the data to retain only those records where the value calculated using the **COUNT()** function is **greater than 1**.

Rules For **Group By ... Having** Clause:

- Columns listed in the select statement have to be listed in the GROUP BY clause
- Columns listed in the GROUP BY clause need not be listed in the SELECT statement
- Only group functions can be used in the HAVING clause
- The group functions listed in the having clause need not be listed in the SELECT statement

Determining Whether Values Are Unique

The **HAVING** clause can be used to find unique values in situations to which DISTINCT does not apply.

The **DISTINCT** clause eliminates duplicates, but does not show which values actually were duplicate in the original data. The **HAVING** clause can identify which values were unique or non-unique.

Example 4:
List candidate who have been hired only once. (Unique Entries Only)

Synopsis:

Tables:	CAND_EMPL
Columns:	CAND_ID, EMPL_ID
Technique:	**Functions:** COUNT(), **Clauses:** GROUP BY ... HAVING, **Others:** Alias

Solution:

```
SELECT CAND_ID "Candidate No.", COUNT(EMPL_ID) "No. Of Jobs" FROM
  CAND_EMPL GROUP BY CAND_ID HAVING COUNT(EMPL_ID) = 1;
```

Output:

```
Candida No. Of Jobs
------- -----------
CD00001           1
CD00003           1
CD00004           1
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_EMPL** table. This table holds data related to the candidates previous employment(s). The **COUNT()** function is applied to the field **EMPL_ID** which will hold the number of times a candidate is hired. This information is then **grouped** on the basis of Candidates Identity (i.e. the **CAND_ID** field). Since only those candidate who have been hired **once** are to be retrieved, the **HAVING** clause is used to finally filter the data to retain only those records where the value calculated using the **COUNT()** function is **equal to 1**.

Advance Grouping Operations

Group By Using The ROLLUP Operator

The ROLLUP operator is used to calculate aggregates and super aggregates for expressions within a GROUP BY statement. Report writers usually use this operator to extract statistics and/or summaries from a result set.

Example 5:
Create a report on the candidates been employed, providing the **last amount paid as salary** on per employment basis. Further, on per candidate basis providing an aggregate of salary for each and all candidates.

Synopsis:

Tables:	CAND_EMPL
Columns:	CAND_ID, EMPL_ID, SAL_ON_LEAVING
Technique:	**Functions:** SUM(); **Operators:** ROLLUP(), **Clauses:** GROUP BY; **Others:** Alias

Solution:

```
SELECT CAND_ID, EMPL_ID, SUM(SAL_ON_LEAVING) "Salary" FROM CAND_EMPL
  GROUP BY ROLLUP (CAND_ID, EMPL_ID);
```

Output:

```
CAND_ID EMPL_ID     Salary
------- ------- ----------
CD00001 CE00001      10500
CD00001              10500
CD00002 CE00002       2500
CD00002 CE00003       6500
CD00002               9000
CD00003 CE00004       1200
CD00003               1200
CD00004 CE00005       1500
CD00004               1500
CD00005 CE00006       3000
CD00005 CE00007       6500
CD00005               9500
                     31700

13 rows selected.
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_EMPL** table. This table holds data related to employments associated with each candidate. The **SUM()** function is applied on the field **SAL_ON_LEAVING**. This provides the sum of salary received by a candidate on leaving the job. This information is then **grouped** on the basis of **CAND_ID** and **EMPL_ID** using the **ROLLUP** operator.

The **ROLLUP** operator is used to display the salary on leaving per employment (i.e. per EMPL_ID) and per candidate (i.e. per CAND_ID).

The ROLLUP operator first calculates the standard aggregate values for the groups specified in the GROUP BY clause for each employment (i.e. per **EMPL_ID**) then creates higher level subtotals, moving from right to left through the list of grouping columns (i.e. per **CAND_ID**).

Group By Using The CUBE Operator

The **CUBE** operator can be applied to all aggregates functions like **AVG()**, **SUM()**, **MAX()**, **MIN()** and **COUNT()** within a **GROUP BY** statement. This operator is usually used by report writers to extract cross-tabular reports from a result set. CUBE produces subtotals for all possible combinations of groupings specified in the **GROUP BY** clause along with a grand total as against the ROLLUP operator which produces only a fraction of possible subtotal combinations.

Example 6:
Find out the salary received for employment by each candidate along with a grand total.

Synopsis:

Tables:	CAND_EMPL
Columns:	CAND_ID, EMPL_ID, SAL_ON_LEAVING
Technique:	**Functions:** SUM(); **Operators:** CUBE(); **Clauses:** GROUP BY

Solution:

```
SELECT CAND_ID, EMPL_ID, SUM(SAL_ON_LEAVING) "Salary" FROM CAND_EMPL
  GROUP BY CUBE (CAND_ID, EMPL_ID);
```

Output:

```
CAND_ID EMPL_ID     Salary
------- ------- ----------
                     31700
        CE00001      10500
        CE00002       2500
        CE00003       6500
        CE00004       1200
        CE00005       1500
        CE00006       3000
        CE00007       6500
CD00001              10500
CD00001 CE00001      10500
CD00002               9000
CD00002 CE00002       2500
CD00002 CE00003       6500
CD00003               1200
CD00003 CE00004       1200
CD00004               1500
CD00004 CE00005       1500
CD00005               9500
CD00005 CE00006       3000
CD00005 CE00007       6500
20 rows selected.
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_EMPL** table. The **SUM()** function is applied to the field **SAL_ON_LEAVING** which will hold the sum of salary on leaving paid per candidate. This information is then **grouped** on the basis of **CAND_ID** and **EMPL_ID** using the **CUBE** operator.

The salary on leaving of every employment for a candidate is displayed using the group by clause. The **CUBE** operator is used to display the **Salary On Leaving per candidate** irrespective of the employments and the total **Salary On Leaving** for all the employments irrespective of the candidates involved.

The **CUBE** operator first calculates the standard aggregate values for the groups specified in the group by clause (Sum of **SAL_ON_LEAVING** for each employment) then creates higher level subtotals, moving from right to left through the list of grouping columns (Sum of **SAL_ON_LEAVING** for each Candidate). Additionally the **CUBE** operator displays the total **SAL_ON_LEAVING** per Candidate irrespective of employment and the total **SAL_ON_LEAVING** of all employments irrespective of the candidates.

Ranking Functions Introduced In Oracle 10g

The ranking functions can be used to calculate ranks, percentiles and n-tiles. Oracle provides various functions to process data and generate appropriate ranking based results.

The keyword **OVER** is required when calling the ranking functions. The ranking is done based on values specified with the **OVER** keyword.

RANK: Returns the rank of items in a group. **RANK()** leaves a gap in the sequence of rankings in the event of a tie. For example, if the result of the **RANK()** function generates two entries for the second position then the position for the third place is skipped. The next position would therefore be the fourth place instead of third.

Syntax:

```
RANK() OVER(<ColumnName>)
```

Example:
Rank the candidates of the Personnel Management System on the bases of number of languages known.

```
SELECT CAND_ID, COUNT(LANG_ID) "No. Of Languages Known",
  RANK() OVER (ORDER BY COUNT(LANG_ID)) "Rank"
  FROM CAND_LANG GROUP BY CAND_ID;
```

Output:

```
CAND_ID No. Of Languages Known        Rank
------- --------------------- ----------
CD00001                     1          1
CD00002                     1          1
CD00003                     1          1
CD00005                     2          4
CD00004                     3          5
```

DENSE_RANK: Returns the rank of items in a group. **DENSE_RANK()** doesn't leave a gap in the sequence of rankings in the event of a tie.

Syntax:

```
DENSE_RANK() OVER(<ColumnName>)
```

Example:
Ignore repetitive ranking, list the candidates of the Personnel Management System on the bases of number of languages known.

```
SELECT CAND_ID, COUNT(LANG_ID) "No. Of Languages Known",
  DENSE_RANK() OVER (ORDER BY COUNT(LANG_ID)) "Dense_Rank"
  FROM CAND_LANG GROUP BY CAND_ID;
```

Output:

```
CAND_ID No. Of Languages Known Dense_Rank
------- ---------------------- ----------
CD00001                      1          1
CD00002                      1          1
CD00003                      1          1
CD00005                      2          2
CD00004                      3          3
```

CUME_DIST: Returns the position of a specified value relative to a group of values. **CUME_DIST()** is short for cumulative distribution.

Syntax:

```
CUME_DIST() OVER(<ColumnName>)
```

Example:
List cumulative distribution for candidates of the Personnel Management System based on number of languages known.

```
SELECT CAND_ID, COUNT(LANG_ID) "No. Of Languages Known",
  CUME_DIST() OVER (ORDER BY COUNT(LANG_ID)) "Cume_Dist"
  FROM CAND_LANG GROUP BY CAND_ID ORDER BY CAND_ID;
```

Output:

```
CAND_ID No. Of Languages Known  Cume_Dist
------- ---------------------- ----------
CD00001                      1         .6
CD00002                      1         .6
CD00003                      1         .6
CD00004                      3          1
CD00005                      2         .8
```

PERCENT_RANK: Returns the percent rank of a value relative to a group of values.

Syntax:

```
PERCENT_RANK() OVER(<ColumnName>)
```

Example:
List the percentage ranking of candidates based on number of languages known.

```
SELECT CAND_ID, COUNT(LANG_ID) "No. Of Languages Known",
  PERCENT_RANK() OVER (ORDER BY COUNT(LANG_ID)) "Percent_Rank"
  FROM CAND_LANG GROUP BY CAND_ID ORDER BY CAND_ID;
```

Output:

```
CAND_ID No. Of Languages Known Percent_Rank
------- ---------------------- ------------
CD00001                      1            0
CD00002                      1            0
CD00003                      1            0
CD00004                      3            1
CD00005                      2          .75
```

NTILE: Returns n-tiles such as tertiles, quartiles and so on. The value passed as a parameter decides the number of groups in which the data will be divided.

Syntax:

```
NTILE() OVER(<ColumnName>)
```

Example:
Distribute the candidates into three part based on number of languages known.

```
SELECT CAND_ID, COUNT(LANG_ID) "No. Of Languages Known",
  NTILE(3) OVER (ORDER BY COUNT(LANG_ID)) "NTile"
  FROM CAND_LANG GROUP BY CAND_ID ORDER BY CAND_ID;
```

Output:

```
CAND_ID No. Of Languages Known      NTile
------- ---------------------- ----------
CD00001                      1          1
CD00002                      1          1
CD00003                      1          2
CD00004                      3          3
CD00005                      2          2
```

ROW_NUMBER: Returns a number with each row in a group.

Syntax:

```
ROW_NUMBER() OVER(<ColumnName>)
```

Example:
Rank the candidates based on number of languages known.

```
SELECT CAND_ID, COUNT(LANG_ID) "No. Of Languages Known",
  ROW_NUMBER() OVER (ORDER BY COUNT(LANG_ID)) "Row_Number"
  FROM CAND_LANG GROUP BY CAND_ID ORDER BY CAND_ID;
```

Output:

```
CAND_ID No. Of Languages Known Row_Number
------- ---------------------- ----------
CD00001                      1          1
CD00002                      1          2
CD00003                      1          3
CD00004                      3          5
CD00005                      2          4
```

Using The Union, Intersect And Minus Clause

Union Clause

Multiple queries can have their output combined using the **union** clause. The **Union** clause merges the output of two or more queries into a single set of rows and columns.

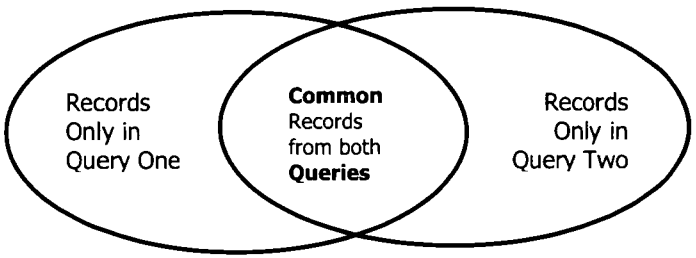


Diagram 7.1: Output of the Union Clause.

REMINDER

The output of both the queries will be as displayed above. The final output of the union clause will be: **Output** = Records from query one + records from query two + A single set of records, common in both queries.

While working with the UNION clause the following pointers should be considered:

- The number of columns and the data types of the columns being selected must be identical in all the **SELECT** statement used in the query. The names of the columns need not be identical.
- **UNION** operates over all of the columns being selected.
- **NULL** values are not ignored during duplicate checking.

- The **IN** operator has a higher precedence than the **UNION** operator.
- By default, the output is sorted in ascending order of the first column of the **SELECT** clause.

Example 1:
Using the UNION clause retrieve the candidate ID of all candidates having poor speaking or writing skills.

Synopsis:

Tables:	CAND_LANG
Columns:	CAND_ID, LANG_ID
Technique:	**Clauses:** WHERE, UNION; **Others:** Alias

Solution:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE WRITTEN = 'Poor'
UNION
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE SPOKEN = 'Poor';
```

Explanation:
Oracle executes the queries as follows:

The first query in the **UNION** example is as follows:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE WRITTEN = 'Poor';
```

The target table will be as follows:

```
CAND_ID    LANG_ID
------- ----------
CD00003           2
CD00005           6
```

The second query in the **UNION** example is as follows:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE SPOKEN = 'Poor';
```

The target table will be as follows:

```
CAND_ID    LANG_ID
------- ----------
CD00003           2
```

The **UNION** clause picks up the common records as well as the individual records in both queries. Thus, the output after applying the **UNION** clause will be:

Output:

```
CAND_ID    LANG_ID
-------  ----------
CD00003           2
CD00005           6
```

REMINDER

The alias assigned to the first query will be applied in the final output, even though an alias has been assigned to the second query it is not applicable.

The Restrictions on using a union are as follows:

- Number of columns in all the queries should be the same
- The data type of the columns in each query must be same
- Unions cannot be used in subqueries
- Aggregate functions cannot be used with union clause

Intersect Clause

Multiple queries can have their output combined using the intersect clause. The **Intersect** clause displays only those rows produced by the intersection of **both** the queries. The output in an Intersect clause will include only those rows that are retrieved **common** to both the queries.

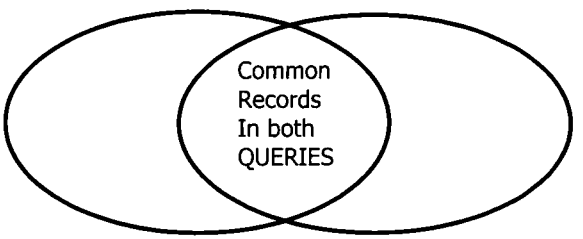


Diagram 7.2: Output of the Intersect clause.

REMINDER

The output of both the queries will be as displayed above. The final output of the Intersect clause will be: **Output** = A single set of records which are common in both queries.

While working with the INTERSECT clause the following pointers should be considered:

- The number of columns and the data types of the columns being selected by the **SELECT** statement in the queries must be identical in all the **SELECT** statements used in the query. The names of the columns need not be identical.
- Reversing the order of the intersected tables does not alter the result.
- **INTERSECT** does not ignore **NULL** values.

Example 2:
Using the INTERSECT clause retrieve the candidate ID of all candidates having poor speaking as well as writing skills.

Synopsis:

Tables:	CAND_LANG
Columns:	CAND_ID, LANG_ID
Technique:	**Clauses:** WHERE, INTERSECT; **Others:** Alias

Solution:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE WRITTEN = 'Poor'
INTERSECT
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE SPOKEN = 'Poor';
```

Explanation:
Oracle executes the queries as follows:

The first query in the **INTERSECT** example is as follows:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE WRITTEN = 'Poor';
```

The target table will be as follows:

```
CAND_ID    LANG_ID
------- ----------
CD00003           2
CD00005           6
```

The second query in the **INTERSECT** example is as follows:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE SPOKEN = 'Poor';
```

The target table will be as follows:

```
CAND_ID    LANG_ID
------- ----------
CD00003           2
```

The **INTERSECT** clause picks up records that are common in both queries. Thus, the output after applying the INTERSECT clause will be as shown in the output.

Output:

```
CAND_ID    LANG_ID
------- ----------
CD00003           2
```

Minus Clause

Multiple queries can have their output combined using the minus clause. The minus clause computes the difference of two table values. Only rows appearing in the first table and not the second will appear in the result.

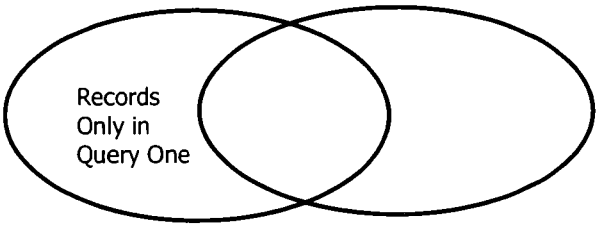


Diagram 7.3: Output of the Minus clause.

REMINDER

The output of both the queries will be as displayed above. The final output of the minus clause will be: **Output =** Records only in query one

While working with the MINUS clause the following pointers should be considered:

- The number of columns and the data types of the columns being selected by the **SELECT** statement in the queries must be identical in all the **SELECT** statements used in the query. The names of the columns need not be identical.
- All the columns in the **WHERE** clause must be in the **SELECT** clause for the **MINUS** operator to work.

Example 3:
Using the MINUS clause retrieve the candidate ID of all candidates having poor speaking skill but fair or good writing skill.

Synopsis:

Tables:	CAND_LANG
Columns:	CAND_ID, LANG_ID
Technique:	**Clauses:** WHERE, MINUS; **Others:** Alias

Solution:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE WRITTEN = 'Poor'
MINUS
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE SPOKEN = 'Poor';
```

Explanation:
Oracle executes the queries as follows:

The first query in the **MINUS** example is as follows:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE WRITTEN = 'Poor';
```

The target table will be as follows:

```
CAND_ID    LANG_ID
-------  ----------
CD00003           2
CD00005           6
```

The second query in the **MINUS** example is as follows:

```
SELECT CAND_ID, LANG_ID FROM CAND_LANG WHERE SPOKEN = 'Poor';
```

The target table will be as follows:

```
CAND_ID    LANG_ID
-------  ----------
CD00003           2
```

The **MINUS** clause picks up records in the first query after filtering the records retrieved by the second query. Thus, the output after applying the MINUS clause will be as shown below.

Output:

```
CAND_ID    LANG_ID
-------  ----------
CD00005           6
```

SUBQUERIES

A **subquery** is a form of an SQL statement that appears inside another SQL statement. It is also termed as **nested query**. The statement containing a subquery is called a **parent** statement. The parent statement uses the rows (**i.e.** the result set) returned by the subquery for further processing.

It can be used for the following:

- To insert records in a target table
- To create tables and insert records in the table created
- To update records in a target table
- To create views
- To provide values for **conditions** in WHERE, HAVING, IN and so on used with SELECT, UPDATE and DELETE statements

Example 1:
Retrieve the address details of a candidate named 'Shravan Dhone'.

Synopsis:

Tables:	CANDIDATE, CAND_ADDR
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_ADDR: CAND_ID, CAND_ADDR1, CAND_ADDR2, CAND_CITY
Technique:	**Sub-Queries; Operators:** IN; **Clauses:** WHERE; **Others:** || (Concatenate), Alias

Solution:

```
SELECT CAND_ID, (CAND_ADDR1 || ' ' || CAND_ADDR2) "Address", CAND_CITY
  FROM CAND_ADDR WHERE CAND_ID IN(SELECT CAND_ID FROM CANDIDATE
    WHERE CAND_FIRST_NAME = 'Shravan' AND CAND_SURNAME = 'Dhone');
```

Output:

```
CAND_ID Address                                  CAND_CITY
------- ---------------------------------------- ---------
CD00005 Pazhayapisharam Sreekrishnapuram         Palakkad
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_ADDR** table, which holds the postal address associated with the candidate named '**Shravan Dhone**'. This table holds address details identified by the candidate number i.e. **CAND_ID**. However, the **CAND_ADDR** table does not contain the field, which holds the candidate's name, which is required to make a comparison.

The candidate's name is available in the **CANDIDATE** table where each candidate is identified by a unique number (i.e. **CAND_ID**). So it is required to access the table **CANDIDATE** and retrieve the **CAND_ID** by supplying the Candidate's Name based on which a comparison can be made with the **CAND_ID** field held in the table **CAND_ADDR**.

Using the **CAND_ID** retrieved from the **CANDIDATE** table, it is now possible to retrieve the address details from the **CAND_ADDR** table by finding a matching value in the **CAND_ID** field in that table.

This type of processing can be done elegantly using a subquery.

In the above solution the sub-query is as follows:

```
SELECT CAND_ID FROM CANDIDATE
  WHERE CAND_FIRST_NAME = 'Shravan' AND CAND_SURNAME = 'Dhone';
```

The target table will be as follows:

```
CAND_ID
-------
CD00005
```

The outer sub-query output will simplify the solution as shown below:

```
SELECT CAND_ID, (CAND_ADDR1 || ' ' || CAND_ADDR2) "Address", CAND_CITY
  FROM CAND_ADDR WHERE CAND_ID IN('CD00005');
```

When the above SQL query is executed the resulting output is equivalent to the desired output of the SQL query using two levels of Sub-queries.

Using Sub-query In The FROM Clause

A subquery can be used in the **FROM** clause of the **SELECT** statement. The concept of using a subquery in the **FROM** clause of the **SELECT** statement is called an **inline view**. A subquery in the **FROM** clause of the **SELECT** statement defines a **data source** for that particular **Select** statement.

Example 2:
List the salary on leaving against each employment, the candidate associated with the employment and the average salary on leaving of that candidate, having salary on leaving per employment more than the average salary on leaving of that candidate.

Synopsis:

Tables:	CAND_EMPL
Columns:	CAND_ID, EMPL_ID, SAL_ON_LEAVING
Technique:	**Sub-Queries**, **Join**, **Functions:** AVG(), **Clauses:** WHERE, GROUP BY; **Others:** Alias

Solution:

```
SELECT A.CAND_ID, A.EMPL_ID, A.SAL_ON_LEAVING, B.AvgSal
  FROM CAND_EMPL A, (SELECT CAND_ID, AVG(SAL_ON_LEAVING) AvgSal
      FROM CAND_EMPL GROUP BY CAND_ID) B
        WHERE A.CAND_ID = B.CAND_ID AND A.SAL_ON_LEAVING > B.AvgSal;
```

Output:

```
CAND_ID EMPL_ID SAL_ON_LEAVING           AVGSAL
------- ------- -------------------- ----------
CD00002 CE00003 6500                       4500
CD00005 CE00007 6500                       4750
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_EMPL** table, which holds the employment details associated with the candidates. The output requirements are the **Candidate Identity**, the **Employment Identity** and the **salary on leaving** associated with the employment and the average salary on leaving of that candidate. The first three requirements can be retrieved from the **CAND_EMPL** table.

The Average salary on leaving on a per candidate basis requires use of another **SELECT** query and a **GROUP BY** clause. This means a sub query can be used, but in this case, the sub query will return a value, which will be a part of the output. Since this query is going to act as a source of data it is placed in the **FROM** clause of the outer query and given an alias **B**. Finally to produce the output a join is used to get the data on the basis of the outer query i.e. (**A.**CAND_ID = **B.**CAND_ID) followed by a **WHERE** clause which actually filters the data before producing the output.

To understand the solution the query mentioned above needs to be simplified. The inner most sub-queries should be handled first and then continued outwards.

The **first step** is to identify the Candidate and their average salary on leaving. This is done by, extracting the value held by the **CAND_ID** field from the **CAND_EMPL** table. The SQL query for this will be as follows:

```
SELECT CAND_ID, AVG(SAL_ON_LEAVING) AvgSal FROM CAND_EMPL
    GROUP BY CAND_ID;
```

The target table will be as follows:

```
CAND_ID     AVGSAL
------- ----------
CD00001      10500
CD00002       4500
CD00003       1200
CD00004       1500
CD00005       4750
```

The **second step** is to associate the data returned by the inner query with the outer. This is done by binding the Sub-query with the **FROM** clause and using join. The output shown above is treated as an individual (temporary) table. This new table is referred as **B**, the alias name specified in the main **SELECT** statement.

The **third step** is to filter the data to output only those records where the salary on leaving per employment is more then the average salary on leaving of the candidate. This is done using a **WHERE** clause (i.e. **A.**SAL_ON_LEAVING **>** **B.**AvgSal)

Finally, the **SELECT** statement is executed as a JOIN i.e. (**WHERE** **A.**CAND_ID = **B.**CAND_ID). This is explained in greater depth later in this chapter.

Using Correlated Sub-queries

A sub-query becomes correlated when the subquery references a column from a table in the parent query. A correlated subquery is evaluated once for each row processed by the parent statement, which can be either a select, delete or update.

A correlated subquery is one way of reading every row in a table and comparing values in each row against related data. It is used whenever a subquery must return a different result for each candidate row considered by the parent query.

Example 3:
List the salary on leaving against each employment, the candidate associated with the employment and the average salary on leaving of that candidate, having salary on leaving per employment more than the average salary on leaving of that candidate.

Synopsis:

Tables:	CAND_EMPL
Columns:	CAND_ID, EMPL_ID, SAL_ON_LEAVING
Technique:	**Sub-Queries, Join, Functions:** AVG(), **Clauses:** WHERE; **Others:** Alias

Solution:

```
SELECT CAND_ID, EMPL_ID, SAL_ON_LEAVING FROM CAND_EMPL A
  WHERE SAL_ON_LEAVING > (SELECT AVG(SAL_ON_LEAVING)
    FROM CAND_EMPL WHERE CAND_ID = A.CAND_ID);
```

Output:

```
CAND_ID EMPL_ID SAL_ON_LEAVING
------- ------- --------------------
CD00002 CE00003 6500
CD00005 CE00007 6500
```

Explanation:
In the above example, the data that has to be retrieved is available in the **CAND_EMPL** table, which holds the employment details associated with the candidates. The output requirements are the **Candidate Identity**, the **Employment Identity**, the **Salary On Leaving** associated with that employment and the **Average salary on leaving** of that Candidate.

These requirements can be retrieved from the **CAND_EMPL** table. However the average salary on leaving on a per candidate basis requires use of another **SELECT** query.

This means a correlated sub query can be used. The correlated sub-query specifically computes the average mileage of each candidate. Since both the queries (i.e. Outer and the Inner) use **CAND_EMPL** table an alias is allotted to the table in the outer query. It is because of this **alias** the inner query is able to distinguish the inner column from the outer column.

Using Multi Column Subquery

Example 4:
Find out all the candidate identities whose are presently residing in the same city.

Synopsis:

Tables:	CAND_ADDR
Columns:	CAND_ID, CAND_ADDR_ID, CAND_ADDR_TYPE, CAND_CITY
Technique:	**Sub_Queries, Operators:** IN, **Clauses:** WHERE

Solution:

```
SELECT A.CAND_ID, A.CAND_CITY FROM CAND_ADDR A
  WHERE (A.CAND_ADDR_TYPE, A.CAND_CITY)
     IN(SELECT B.CAND_ADDR_TYPE, B.CAND_CITY
        FROM CAND_ADDR B WHERE A.CAND_ADDR_ID <> B.CAND_ADDR_ID);
```

Output:

```
CAND_ID CAND_CITY
------- --------------
CD00002 Mumbai
CD00001 Mumbai
```

Explanation:
In the above example, each row of the outer query is compared with the values from the inner query (Multi Row and Multi Column). This means that the values held in the **CAND_ADDR_TYPE** and **CAND_CITY** fields from the outer query are compared with the values held in the **CAND_ADDR_TYPE** and **CAND_CITY** fields retrieved by the inner query.

Using Sub-query in CASE Expressions

Example 5:
List the candidates details such as user id, candidate number, birthdate and marital status (i.e. whether married or single).

Synopsis:

Tables:	CANDIDATE
Columns:	CAND_USER_ID, CAND_ID, CAND_BIRTHDATE, CAND_MARITAL_STATUS
Technique:	**Sub_Queries, Operators:** IN, **Clauses:** CASE WHEN ... THEN

Solution:

```
SELECT C.CAND_USER_ID, C.CAND_ID, C.CAND_BIRTHDATE, (CASE
  WHEN CAND_ID IN(SELECT CAND_ID FROM CANDIDATE
     WHERE CAND_MARITAL_STATUS='Yes')
  THEN 'Married' ELSE 'Single' END) "Marital Status" FROM CANDIDATE C;
```

Output:

```
CAND_US CAND_ID CAND_BIRT Marital
------- ------- --------- -------
US00001 CD00001 03-JAN-81 Married
US00002 CD00002 01-JAN-82 Single
US00003 CD00003 23-JUN-52 Married
US00004 CD00004 22-JUN-76 Single
US00005 CD00005 21-FEB-82 Single
```

Explanation:
In the above example, the inner query will return a value (**i.e.** if a record exists in the **CANDIDATE** table then the candidate is married or else the single). Based on the value returned, the outer query will display the Marital Status as either **Married** or **Single**.

Using Subquery In An ORDER BY clause

Example 6:
List the candidate's address type and city in the order of the candidate's birth date.

Synopsis:

Tables:	CAND_PRSN_INFO, CAND_ADDR
Columns:	CAND_PRSN_INFO: CAND_ID, CAND_BIRTHDATE CAND_ADDR: CAND_ID, CAND_ADDR_ID, CAND_ADDR_TYPE, CAND_CITY
Technique:	**Sub_Queries, Clauses:** ORDER BY **Others:** Alias, || (Concatenate)

Solution:

```
SELECT CAND_ID, CAND_ADDR_ID, CAND_ADDR_TYPE, CAND_CITY
  FROM CAND_ADDR A
     ORDER BY (SELECT CAND_BIRTHDATE FROM CAND_PRSN_INFO B
        WHERE A.CAND_ID = B.CAND_ID);
```

Output:

```
CAND_ID CAND_AD CAND_ADDR_TYPE       CAND_CITY
------- ------- -------------------- -------------------------
CD00003 CA00004 Permanent            Mumbai
CD00003 CA00005 Present              THIRUVANANTHAPURAM
CD00004 CA00006 Permanent            Sydney
CD00004 CA00007 Present              NAVI MUMBAI
CD00001 CA00001 Present              Mumbai
CD00001 CA00002 Present              Mira Road, Dist.Thane
CD00002 CA00003 Present              Mumbai
CD00005 CA00008 Permanent            Palakkad
8 rows selected.
```

Explanation:
In the above example, the output that needs to be ordered on the basis of candidate's birth date is available in the **CAND_ADDR** table. Since the output needs to be ordered on the basis of candidate's birth date, which is available in the **CAND_PRSN_INFO** table, there is a need of a separate query, which can return the appropriate candidate's birth date from the **CAND_PRSN_INFO** table, based on the CAND_ID. Based on the values returned from the inner query the output produced by the outer query will be ordered.

This is done, by placing the inner query, in the **ORDER BY** clause and further correlating it with the outer query on the basis of the **CAND_ID** being **the common field** in the tables **CAND_ADDR** and **CAND_PRSN_INFO**.

Using EXISTS / NOT EXISTS Operator

The **EXISTS** operator is usually used with correlated subqueries. This operator enables the testing of whether a value retrieved by the outer query exists in the results set of the values retrieved by the inner query. If the subquery returns at least one row, the operator returns **TRUE**. If the value does not exist, it returns **FALSE**.

The **EXISTS** operator ensures that the search in the inner query terminates when at least one match is found.

Similarly, the **NOT EXISTS** operator enables the testing of whether a value retrieved by the outer query is not a part of the result set of the values retrieved by the inner query.

Example 7:
List the candidates who have been employed more than once.

Synopsis:

Tables:	CANDIDATE, CAND_EMPL
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_EMPL: CAND_ID
Technique:	**Operators:** EXISTS(), **Clauses:** WHERE; **Others:** || (Concatenate), Alias

Solution:

```
SELECT CAND_ID, (CAND_FIRST_NAME||' '||CAND_SURNAME) "Name"
  FROM CANDIDATE A WHERE EXISTS(SELECT COUNT(CAND_ID)
      FROM CAND_EMPL WHERE CAND_ID = A.CAND_ID GROUP BY 1
          HAVING COUNT(CAND_ID)>1);
```

Output:

```
CAND_ID Name
------- ---------------
CD00002 Hansel Colaco
CD00005 Shravan Dhone
```

Explanation:
In the above example, the inner query is correlated to the outer query via the **CAND_ID** field. As soon as the search in the inner query retrieves the desired match, i.e. **CAND_ID = A.CAND_ID** and **COUNT(CAND_ID)>1** the search is terminated. This means that the inner query stops it's processing and the outer query then produces the output.

Example 8:
List the candidates who do not have a present address.

Synopsis:

Tables:	CANDIDATE, CAND_ADDR
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_ADDR: CAND_ID, CAND_ADDR_TYPE
Technique:	**Operators:** NOT, EXISTS(), **Clauses:** WHERE **Others:** Alias

Solution:

```
SELECT CAND_ID, (CAND_FIRST_NAME||' '||CAND_SURNAME) "Name"
  FROM CANDIDATE A
     WHERE NOT EXISTS(SELECT 'SCT' FROM CAND_ADDR
        WHERE CAND_ID = A.CAND_ID AND CAND_ADDR_TYPE = 'Present');
```

Output:

```
CAND_ID Name
------- ------------------------
CD00005 Shravan Dhone
```

Explanation:
In the above example, the inner query is correlated with the outer query via the **CAND_ID** field. Since the **NOT EXISTS** operator is used, if the inner query retrieves no rows at all i.e. the condition **CAND_ID = A.CAND_ID** fails, the outer query produces the output. This means after the inner query stops it's processing, the outer query sends the output based on the operator used. *In the case of the inner query there is no need to return a specific value, hence a constant **'SCT'** is used instead. This is useful in terms of performance as it will be faster to select a constant than a column.*

JOINS

Joining Multiple Tables (Equi Joins)

Sometimes it is necessary to work with multiple tables as though they were a single entity. Then a single SQL sentence can manipulate data from all the tables. **Joins** are used to achieve this. Tables are joined on columns that have the same **data type** and **width** in the tables.

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

The JOIN operator can be used to specify how to relate tables in the query. Types of JOIN:

- INNER
- OUTER (LEFT, RIGHT, FULL)
- CROSS

INNER JOIN: Inner joins also known as **Equi Joins** are the most common joins used in SQL*Plus. They are know as **equi joins** because the **where statement** generally compares two columns from two tables with the equivalence operator =. This type of join is by far the most commonly used. In fact, many systems use this type as the default join. This type of join can be used in situations where selecting only those rows that have values in common in the columns specified in the ON clause, is required. In short, the INNER JOIN returns all rows from both tables where there is a match.

OUTER JOIN: Outer joins are similar to inner joins, but give a bit more flexibility when selecting data from related tables. This type of join can be used in situations where it is desired, to select all rows from the table on the left, or from the table on the right, or both regardless of whether the other table has values in common and (usually) enter NULL where data is missing.

CROSS JOIN: A cross join returns what's known as a **Cartesian product**. This means that the join combines every row from the left table with every row in the right table. As can be imagined, sometimes this join produces a mess, but under the right circumstances, it can be very useful. This type of join can be used in situations where it is desired, to select all possible combinations of rows and columns from both tables. This kind of join is usually not preferred as it may run for a very long time and produce a huge result set that may not be useful.

Syntax:

ANSI-style

```
SELECT <ColumnName1>, <ColumnName2>, <ColumnName N>
  FROM <TableName1> INNER JOIN <TableName2>
     ON <TableName1>.<ColumnName1>=<TableName2>.<ColumnName2>
     WHERE <Condition>
     ORDER BY <ColumnName1>, <ColumnName2>, <ColumnNameN>
```

Theta-style

```
SELECT <ColumnName1>, <ColumnName2>, <ColumnNameN>
  FROM <TableName1>, <TableName2>
  WHERE <TableName1>.<ColumnName1> = <TableName2>.<ColumnName2>
     AND <Condition>
     ORDER BY <ColumnName1>, <ColumnName2>, <ColumnNameN>
```

In the above syntax:

- **ColumnName1** in **TableName1** is usually that table's **Primary Key**
- **ColumnName2** in **TableName2** is a **Foreign Key** in that table
- **ColumnName1** and **ColumnName2** must have the **same Data Type** and for certain data types, the same size

Inner Join

Example 9:
List the candidate details along with their address details.

Synopsis:

Tables:	CANDIDATE, CAND_ADDR
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_ADDR: CAND_ID, CAND_ADDR_TYPE, CAND_CITY
Technique:	**Join:** INNER JOIN ... ON, SIMPLE, **Clauses:** WHERE; **Others:** ||(Concatenate), Alias

Solution 1 (Ansi-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY FROM CANDIDATE A
      INNER JOIN CAND_ADDR B ON B.CAND_ID = A.CAND_ID;
```

Solution 2 (Theta-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY FROM CANDIDATE A, CAND_ADDR B
      WHERE B.CAND_ID = A.CAND_ID;
```

Output:

```
CAND_ID Name              CAND_ADDR_TYPE  CAND_CITY
------- ----------------- --------------- --------------------
CD00001 Sharanam Shah     Present         Mumbai
CD00001 Sharanam Shah     Present         Mira Road, Dist.Thane
CD00002 Hansel Colaco     Present         Mumbai
CD00003 Ivan Bayross      Permanent       Mumbai
CD00003 Ivan Bayross      Present         THIRUVANANTHAPURAM
CD00004 Chhaya Bankar     Permanent       Sydney
CD00004 Chhaya Bankar     Present         NAVI MUMBAI
CD00005 Shravan Dhone     Permanent       Palakkad
8 rows selected.
```

Explanation:
In the above example, in the **CANDIDATE** table, the **CAND_ID** column is the **primary key**, meaning that no two rows can have the same **CAND_ID**. The **CAND_ID** distinguishes two persons even if they have the same name. The data required in this example is available in two tables i.e. **CANDIDATE** and **CAND_ADDR**. This is because address details are going to be a part of the output but are not available in the **CANDIDATE** table.

Notice that:

- The **CAND_ID** column is the primary key of the **CANDIDATE** table
- The **CAND_ADDR_TYPE** column is present in the **CAND_ADDR** table
- The **CAND_ID** column in the **CANDIDATE** table is used to refer to the candidate addresses in the **CAND_ADDR** table without using their names

On the basis of the reference available in the **CANDIDATE** table i.e. the **CAND_ID** field its possible to link to the **CAND_ADDR** table and fetch the desired address details for display. This is easily possible with the use of inner join based on the condition (**B.**CAND_ID = **A.**CAND_ID).

REMINDER

If the column names on which the **join** is to be specified are the same in each table reference the columns using **TableName.ColumnName**.

Outer Join

Example 10:
List the candidate details along with their employment details. Using Left Outer Join.

Synopsis:

Tables:	CANDIDATE, CAND_EMPL
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_EMPL: EMPL_ID, EMPL_NAME, CAND_ID
Technique:	**Join:** LEFT JOIN ... ON, **Clauses:** WHERE; **Others:** || (Concatenate), Alias

Solution 1 (Ansi-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
       B.EMPL_ID, B.EMP_NAME
  FROM CANDIDATE A LEFT JOIN CAND_EMPL B ON A.CAND_ID = B.CAND_ID;
```

Solution 2 (Theta-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
       B.EMPL_ID, B.EMP_NAME
  FROM CANDIDATE A, CAND_EMPL B WHERE A.CAND_ID = B.CAND_ID (+);
```

Output:

```
CAND_ID Name                EMPL_ID EMP_NAME
------- ------------------ ------- ------------
CD00001 Sharanam Shah      CE00001 Essenpross Technologies
CD00002 Hansel Colaco      CE00002 Devassy And Devassy C.A.s
CD00002 Hansel Colaco      CE00003 NIC, Under Ministry of I.T
CD00003 Ivan Bayross       CE00004 THE RETREAT HOTEL
CD00004 Chhaya Bankar      CE00005 THE OBEROI HOTELS And RESORTS
CD00005 Shravan Dhone        CE00006 Venktron Digital System Pvt
Ltd
CD00005 Shravan Dhone      CE00007 Nitish Roy And Company

7 rows selected.
```

Explanation:
In the above example, the data required is all the candidate names along with their employment details. This means all the candidates have to be listed. The data is available in two tables i.e. CANDIDATE and CAND_EMPL.

In such a situation, the **LEFT JOIN** can be used which returns all the rows from the first table (i.e. CANDIDATE), even if there are no matches in the second table (CAND_EMPL). This means, if there are candidates entered in **CANDIDATE** table that do not have any employment information in **CAND_EMPL** table, those rows will also be listed. Notice the keyword **LEFT JOIN** in the first solution (Ansi-style) and the **(+)** in the second solution (Theta-style). This indicates that all rows from the first table i.e. CANDIDATE will be displayed even though there exists no matching rows in the second table i.e. CAND_EMPL.

Notice that:

- The **CAND_ID** column is the primary key of the **CANDIDATE** table
- The **CAND_EMPL** is a table that holds the employment information.
  In **CAND_EMPL** table:
  - **CAND_ID** column is used to refer to the candidate in the CANDIDATE table via the CAND_ID column.

To retrieve the data required, both the tables have to be linked on the basis of common columns using joins as follows:

```
A.CAND_ID = B.CAND_ID
```

This means the CAND_ID field of **CANDIDATE** table is joined with CAND_ID field of the **CAND_EMPL** table

Example 11:
List the candidate details along with their employment details. Use Right Outer Join.

Synopsis:

Tables:	CANDIDATE, CAND_EMPL
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_EMPL: EMPL_ID, EMPL_NAME, CAND_ID
Technique:	**Join:** RIGHT JOIN ... ON, **Clauses:** WHERE; **Others:** || (Concatenate), Alias

Solution 1 (Ansi-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
      B.EMPL_ID, B.EMP_NAME
  FROM CAND_EMPL B RIGHT JOIN CANDIDATE A ON B.CAND_ID = A.CAND_ID;
```

Solution 2 (Theta-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
      B.EMPL_ID, B.EMP_NAME
  FROM CAND_EMPL B, CANDIDATE A WHERE B.CAND_ID (+) = A.CAND_ID;
```

Output:

```
CAND_ID Name                EMPL_ID EMP_NAME
------- ------------------- ------- ------------
CD00001 Sharanam Shah       CE00001 Essenpross Technologies
CD00002 Hansel Colaco       CE00002 Devassy And Devassy C.A.s
CD00002 Hansel Colaco       CE00003 NIC, Under Ministry of I.T
CD00003 Ivan Bayross        CE00004 THE RETREAT HOTEL
CD00004 Chhaya Bankar       CE00005 THE OBEROI HOTELS And RESORTS
CD00005 Shravan Dhone         CE00006 Venktron Digital System Pvt
Ltd
CD00005 Shravan Dhone       CE00007 Nitish Roy And Company

7 rows selected.
```

Explanation:

In the above example, the data required is the candidate's name along with their employment details. But in this case **RIGHT JOIN** is being used. This means all the candidates have to be listed even though their corresponding employment details are not present. The data is available in two tables i.e. **CANDIDATE** and **CAND_EMPL**.

Since the **RIGHT JOIN** returns all the rows from the second table even if there are no matches in the first table, the first table in the **FROM** clause will have to be **CAND_EMPL** and the second table **CANDIDATE**. This means, if there are CANDIDATE that do not have any employment information, those rows will also be listed. Notice the keyword **RIGHT JOIN** in the first solution (Ansi-style) and the **(+)** in the second solution (Theta-style). This indicates that all rows from the second table i.e. CANDIDATE will be displayed even though there exists no matching rows in the first table i.e. CAND_EMPL.

Notice that:

- The **CAND_ID** column is the primary key of the **CANDIDATE** table
- The **CAND_EMPL** is a table that holds the employment information
  In **CAND_EMPL** table:
  - **CAND_ID** column is used to refer to the candidate in the CANDIDATE table via the CAND_ID column.

To retrieve the data required, both the tables have to be linked on the basis of common columns using joins as follows:

```
A.CAND_ID = B.CAND_ID
```

This means the CAND_ID field of **CANDIDATE** table is joined with CAND_ID field of the **CAND_EMPL** table

Cross Join

If it is desired to combine each candidate's salary expectations with an employer's salary slab table so as to analyze each candidate at different minimum and maximum salary slabs, this is elegantly done using a Cross Join.

Example 12:
Create a report using **cross join** that will display the candidates and their suitability based on their expected salary and the employer's salary slabs. *Ensure that the table EmployersSlab is created and populated with some records*.

Synopsis:

Tables:	CANDIDATE, EMPLOYERSSLAB
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME EMPLOYERSSLAB: EMPNAME, MINSAL, MAX SAL
Technique:	**Join:** CROSS JOIN, **Others:** || (Concatenate)

Prior executing the SQL statement the EMPLOYERSSLAB table has to be created and filled in with some sample data.

```
CREATE TABLE EMPLOYERSSLAB (EMPNAME Varchar2(10), MINSAL NUMBER(10),
  MAXSAL NUMBER(10));
```

Insert Statements for the table EMPLOYERSSLAB:

```
INSERT INTO EMPLOYERSSLAB VALUES('SCT', 5000, 25000);
INSERT INTO EMPLOYERSSLAB VALUES('Tata', 15000, 35000);
INSERT INTO EMPLOYERSSLAB VALUES('Telco', 20000, 45000);
INSERT INTO EMPLOYERSSLAB VALUES('Wipro', 30000, 55000);
```

Solution:

```
SELECT TRIM( (C.CAND_ID || '-' || C.CAND_FIRST_NAME || ' '
  || C.CAND_SURNAME) ) "Candidate",   C.CAND_SAL_EXPCTD "Expected",
  S.MINSAL, S.MAXSAL, S.EMPNAME, (CASE WHEN C.CAND_SAL_EXPCTD
      BETWEEN S.MINSAL AND S.MAXSAL THEN 'Suitable'
      ELSE 'Not Suitable' END) "Result"
  FROM CANDIDATE C CROSS JOIN EMPLOYERSSLAB S;
```

Output:

```
Candidate              Expected  MINSAL MAXSAL EMPNAME Result
---------------------- --------- ------ ------ ------- -------
--------
CD00001-Sharanam Shah  55000       5000  25000 SCT     Not
Suitable
CD00002-Hansel Colaco  7000        5000  25000 SCT
Suitable
CD00003-Ivan Bayross   25000       5000  25000 SCT
Suitable
CD00004-Chhaya Bankar  27000       5000  25000 SCT     Not
Suitable
CD00005-Shravan Dhone  36000       5000  25000 SCT     Not
Suitable
CD00001-Sharanam Shah  55000      15000  35000 Tata    Not
Suitable
CD00002-Hansel Colaco  7000       15000  35000 Tata    Not
Suitable
CD00003-Ivan Bayross   25000      15000  35000 Tata
Suitable
CD00004-Chhaya Bankar  27000      15000  35000 Tata
Suitable
CD00005-Shravan Dhone  36000      15000  35000 Tata    Not
Suitable
CD00001-Sharanam Shah  55000      20000  45000 Telco   Not
Suitable
CD00002-Hansel Colaco  7000       20000  45000 Telco   Not
Suitable
CD00003-Ivan Bayross   25000      20000  45000 Telco
Suitable
CD00004-Chhaya Bankar  27000      20000  45000 Telco
Suitable
CD00005-Shravan Dhone  36000      20000  45000 Telco
Suitable
CD00001-Sharanam Shah  55000      30000  55000 Wipro
Suitable
CD00002-Hansel Colaco  7000       30000  55000 Wipro   Not
Suitable
```

Output: (Continue)

```
Candidate                Expected  MINSAL MAXSAL EMPNAME Result
------------------------ --------- ------ ------ ------- -------
--------
CD00003-Ivan Bayross     25000      30000  55000 Wipro   Not
Suitable
CD00004-Chhaya Bankar    27000      30000  55000 Wipro   Not
Suitable
CD00005-Shravan Dhone    36000      30000  55000 Wipro
Suitable

20 rows selected.
```

Explanation:
In the above example, the data required is available in two tables i.e. **CANDIDATE** and **EMPLOYERSSLAB**. In the table **CANDIDATE**, there exists, **CAND_ID** and their Names. In the second table **EMPLOYERSSLAB**, there exists a list of Salary slabs comprising of minimum and maximum salary that an employer can afford.

The output is required in the form of a report, which will display analysis based on the **EMPLOYERSSLAB** table for each row held in the **CANDIDATE**. In such a situation, a CROSS JOIN can be used which will combine each record from the left table with that of the right table. In this example a **CROSS JOIN** will combine each **CANDIDATE** with the expected salary from the Candidate table with each Employer's salary slab i.e. each record in the **EMPLOYERSSLAB** table after applying some calculations. Using **CROSS JOIN**, a matrix between the tables named **CANDIDATE** table and the **EMPLOYERSSLAB** table can be created.

The above **SELECT** statement creates a record for each candidate and the expected salary with the minimum and maximum salary that an employer can afford. Based on this analysis a result is shown which displays whether that particular candidate is **SUITABLE** or **NOT SUITABLE** for that particular employer. The results are known as a Cartesian product, which combines every record in the left table i.e. **CANDIDATE** with every record in the right table i.e. **EMPLOYERSSLAB**.

Oracle versions **prior to 9i** don't support an explicit **CROSS JOIN**, but the same results can be obtained by using the following statement:

```
SELECT TRIM( (C.CAND_ID || '-' || C.CAND_FIRST_NAME || ' ' ||
  C.CAND_SURNAME)) "Candidate", C.CAND_SAL_EXPCTD "Expected", S.MINSAL,
  S.MAXSAL, S.EMPNAME, (CASE WHEN C.CAND_SAL_EXPCTD
      BETWEEN S.MINSAL AND S.MAXSAL THEN 'Suitable' ELSE 'Not Suitable'
      END) "Result" FROM CANDIDATE C, EMPLOYERSSLAB S;
```

Guidelines for Creating Joins

- When writing a select statement that joins tables, precede the column name with the table name for clarity
- If the same column name appears in more than one table, the column name must be prefixed with the table name
- The WHERE clause is the most critical clause in a join select statement. Always make sure to include the WHERE clause

Simplifying Joins With The USING Keyword

SQL/92 allows simplifying the JOIN condition with the USING clause. However, the query should satisfy the following conditions:

- The query must use an equi-join
- The columns in the equi-join must have the same name

Example 13:
List the candidate details along with their address details.

Synopsis:

Tables:	CANDIDATE, CAND_ADDR
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_ADDR: CAND_ID, CAND_ADDR_TYPE, CAND_CITY
Technique:	**Join:** INNER JOIN ... ON, SIMPLE, **Clauses:** WHERE, USING; **Others:** || (Concatenate), Alias

Solution 1 (Ansi-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY FROM CANDIDATE A
  INNER JOIN CAND_ADDR B ON B.CAND_ID = A.CAND_ID;
```

Solution 2 (Theta-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY FROM CANDIDATE A, CAND_ADDR B
  WHERE B.CAND_ID = A.CAND_ID;
```

Solution 3 (USING clause):

```
SELECT CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY FROM CANDIDATE A
  INNER JOIN CAND_ADDR B USING (CAND_ID);
```

Output:

```
CAND_ID Name              CAND_ADDR_TYPE  CAND_CITY
------- ----------------- --------------- ----------------------
CD00001 Sharanam Shah     Present         Mumbai
CD00001 Sharanam Shah     Present         Mira Road, Dist.Thane
CD00002 Hansel Colaco     Present         Mumbai
CD00003 Ivan Bayross      Permanent       Mumbai
CD00003 Ivan Bayross      Present         THIRUVANANTHAPURAM
CD00004 Chhaya Bankar     Permanent       Sydney
CD00004 Chhaya Bankar     Present         NAVI MUMBAI
CD00005 Shravan Dhone     Permanent       Palakkad
8 rows selected.
```

Explanation:
In the above example, in the **CANDIDATE** table, the **CAND_ID** column is the **primary key**, meaning that no two rows can have the same **CAND_ID**. The **CAND_ID** distinguishes two persons even if they have the same name. The data required in this example is available in two tables i.e. **CANDIDATE** and **CAND_ADDR**. This is because address details are going to be a part of the output but are not available in the **CANDIDATE** table.

Notice that:

- The **CAND_ID** column is the primary key of the **CANDIDATE** table
- The **CAND_ADDR_TYPE** column is present in the **CAND_ADDR** table
- The **CAND_ID** column in the **CANDIDATE** table is used to refer to the candidate addresses in the **CAND_ADDR** table without using their names
- On the basis of the reference available in the **CANDIDATE** table i.e. the **CAND_ID** field it is possible to link to the **CAND_ADDR** table and fetch the desired address details for display. This is easily possible with the use of inner join based on the condition (**B.**CAND_ID = **A.**CAND_ID) or the **USING** (CAND_ID) keyword. This means the **CAND_ID** field of Candidate table is joined with **CAND_ID** field of the **CAND_ADDR** table

Now since both the tables are linked using a join, data can be retrieved as if they are all in one table using the alias as:

```
A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY
```

However, if the **USING clause** is specified, then the column that is used with the USING clause (i.e. **CAND_ID**) cannot have a qualifier (i.e. **CAND_ID** cannot be preceded with **A.**). Thus in this case the data can be retrieved using the alias as:

```
CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY
```

Example 14:
List the candidate details along with their address as well as their user names and passwords details.

Synopsis:

Tables:	CANDIDATE, CAND_ADDR, USERS
Columns:	CANDIDATE: CAND_ID, CAND_FIRST_NAME, CAND_SURNAME CAND_ADDR: CAND_ID, CAND_ADDR_TYPE, CAND_CITY USERS: USER_ID, USER_USERNAME, USER_PASSWORD
Technique:	**Join:** INNER JOIN ... ON, SIMPLE, **Clauses:** WHERE, USING; **Others:** || (Concatenate), Alias

Solution 1 (Ansi-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY, C.USER_USERNAME, C.USER_PASSWORD
  FROM CANDIDATE A INNER JOIN CAND_ADDR B
        ON B.CAND_ID = A.CAND_ID INNER JOIN USERS C
           ON A.CAND_USER_ID = C.USER_ID;
```

Solution 2 (Theta-style):

```
SELECT A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY, C.USER_USERNAME, C.USER_PASSWORD
  FROM CANDIDATE A, CAND_ADDR B, USERS C
     WHERE B.CAND_ID = A.CAND_ID AND A.CAND_USER_ID = C.USER_ID;
```

Solution 3 (USING clause):

```
SELECT CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY, C.USER_USERNAME, C.USER_PASSWORD
  FROM CANDIDATE A INNER JOIN CAND_ADDR B
     USING (CAND_ID) INNER JOIN USERS C
        ON A.CAND_USER_ID = C.USER_ID;
```

Output:

```
CAND_ID Name            CAND_ADDR_TYPE  CAND_CITY
USER_USERNAME   USER_PASSWORD
------- --------------- --------------- ----------------------
--------------- -------------
CD00001 Sharanam Shah   Present         Mumbai
sharanam        sct2306
CD00001 Sharanam Shah   Present         Mira Road, Dist.Thane
sharanam        sct2306
CD00002 Hansel Colaco   Present         Mumbai
hansel          sct2306
```

Output: (Continued)

```
CAND_ID Name            CAND_ADDR_TYPE  CAND_CITY
USER_USERNAME   USER_PASSWORD
------- --------------- --------------- ----------------------
--------------- -------------
CD00003 Ivan Bayross    Permanent       Mumbai
ivan            sct2306
CD00003 Ivan Bayross    Present         THIRUVANANTHAPURAM
ivan            sct2306
CD00004 Chhaya Bankar   Permanent       Sydney
chhaya          sct2306
CD00004 Chhaya Bankar   Present         NAVI MUMBAI
chhaya          sct2306
CD00005 Shravan Dhone   Permanent       Palakkad
anil            sct2306
8 rows selected.
```

Explanation:
In the above example, the data required is available in three tables i.e. **CANDIDATE**, **CAND_ADDR** and **USERS**. These tables are linked via common fields. This is because the data is spread across the tables based on a normalized schema.

Notice that:

- The **CANDID** column is the primary key of the **CANDIDATE** table
- The **USER_ID** column is the primary key of the **USERS** table
- The **CAND_ADDR** is a table that holds the address details of the customers.
  In **CAND_ADDR** TABLE:
  - **Cand_Addr_ID** column is the primary key of that table
  - **CAND_Id column** is used to refer to the candidates in the **CANDIDATE** table
  - **CAND_USER_ID column** is used to refer to the users in the **USERS** table

To retrieve the data required these tables have to be linked on the basis of a common column using joins with one of the following techniques shown below:

```
B.CAND_ID = A.CAND_ID AND A.CAND_USER_ID = C.USER_ID
```

```
USING (CAND_ID) AND A.CAND_USER_ID = C.USER_ID
```

*(The **USING** clause cannot be used as column names are different)*

This means the **CAND_ID** field of **CANDIDATE** table is joined with **CAND_ID** field of the **CAND_ADDR** table and the **CAND_USER_ID** field of **CANDIDATE** table is joined with **USER_ID** field of the **USERS** table

Now since both the tables are linked using a join, data can be retrieved as if they are all in one table using the alias as:

```
A.CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY, C.USER_USERNAME, C.USER_PASSWORD
```

However, if the USING clause is specified, then the column that is used with the USING clause (i.e. **CAND_ID**) cannot have a qualifier (i.e. **CAND_ID** cannot be preceded with **A.**). Thus in this case the data can be retrieved using the alias as:

```
CAND_ID, (A.CAND_FIRST_NAME || ' ' || A.CAND_SURNAME) "Name",
  B.CAND_ADDR_TYPE, B.CAND_CITY, C.USER_USERNAME, C.USER_PASSWORD
```

The column CAND_USER_ID of the Candidate table and the column USER_ID of the Users table have different names and hence the using clause cannot be applied here.

SQL MODEL Clause

The SQL **MODEL** clause allows users to embed spreadsheet-like models in a **SELECT** statement, in a way that was previously the domain of dedicated multidimensional OLAP servers such as Oracle Express and Oracle 9i OLAP. The SQL **MODEL** clause brings an entirely new dimension to Oracle analytical queries and addresses a number of traditional shortcomings with the way SQL normally works.

The aim of the SQL **MODEL** clause is to give normal SQL statements the ability to create a multidimensional array from the results of a normal **SELECT** statement, carry out any number of interdependent inter-row and inter-array calculations on this array and then update the base tables with the results of the model.

Thus in short the Model clause allows treating relational data as a multidimensional array to which spreadsheet-like calculations can be applied. The result is a query that is easier to develop, understand and modify.

The **MODEL** clause defines a multidimensional array by mapping the columns of a query into three groups i.e. partitioning, dimension and measure columns.

These elements perform the following tasks:

- Partitions define logical blocks of the result set in a way similar to the partitions of the analytical functions. MODEL rules are applied to the cells of each partition.
- Dimensions identify each measure cell within a partition. These columns identify characteristics such as date, region and product name.
- Measures are analogous to the measures of a **fact** table in a **star** schema. They typically contain numeric values such as sales units or cost. Each cell is accessed within its partition by specifying its full combination of dimensions.

To create rules on these multidimensional arrays, computation rules are defined and expressed in terms of the dimension values. The rules are flexible and concise, and can use wild cards and **FOR** loops for maximum expressiveness. Calculations built with the **MODEL** clause improve on traditional spreadsheet calculations by integrating analyses into the database, improving readability with symbolic referencing and providing scalability and much better manageability.

Example 1:
Suppose it is desired to project commission earned (Revenue From Clients Of The Personnel Management System) per branch and client for the year 2005 based on the earnings of the years 2003 & 2004. A table named **CLIENTREVENUE** will be created holding the CLIENT_BRANCH, CLIENT_NAME, YEAR and the REVENUE earned in that Year.

Solution: (Table Creation)

```
CREATE TABLE CLIENTREVENUE (CLIENT_BRANCH VarChar2(15),
  CLIENT_NAME VarChar2(20), YEAR Number(4), REVENUE Number(8));
```

(Records Population)

```
INSERT INTO CLIENTREVENUE VALUES ('Mumbai', 'Summer Harvest', 2003, 4500);
INSERT INTO CLIENTREVENUE VALUES ('Mumbai', 'Summer Harvest', 2004, 9500);
INSERT INTO CLIENTREVENUE VALUES ('Shirdi', 'Summer Harvest', 2003, 8500);
INSERT INTO CLIENTREVENUE VALUES ('Shirdi', 'Summer Harvest', 2004, 12500);
INSERT INTO CLIENTREVENUE VALUES ('Bangalore', 'Summer Harvest', 2003, 6500);
INSERT INTO CLIENTREVENUE VALUES ('Bangalore', 'Summer Harvest', 2004, 10500);
INSERT INTO CLIENTREVENUE VALUES ('Mumbai', 'Avlons', 2003, 4500);
INSERT INTO CLIENTREVENUE VALUES ('Mumbai', 'Avlons', 2004, 7500);
INSERT INTO CLIENTREVENUE VALUES ('Shirdi', 'Avlons', 2003, 6900);
INSERT INTO CLIENTREVENUE VALUES ('Shirdi', 'Avlons', 2004, 6500);
INSERT INTO CLIENTREVENUE VALUES ('Bangalore', 'Avlons', 2003, 6300);
INSERT INTO CLIENTREVENUE VALUES ('Bangalore', 'Avlons', 2004, 10500);
INSERT INTO CLIENTREVENUE VALUES ('Mumbai', 'Taj India', 2003, 4500);
INSERT INTO CLIENTREVENUE VALUES ('Mumbai', 'Taj India', 2004, 5500);
INSERT INTO CLIENTREVENUE VALUES ('Shirdi', 'Taj India', 2003, 6500);
INSERT INTO CLIENTREVENUE VALUES ('Shirdi', 'Taj India', 2004, 16500);
INSERT INTO CLIENTREVENUE VALUES ('Bangalore', 'Taj India', 2003, 6500);
INSERT INTO CLIENTREVENUE VALUES ('Bangalore', 'Taj India', 2004, 12500);
```

(Revenue Projection)

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL   PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
  DIMENSION BY (YEAR)   MEASURES (REVENUE)
  RULES ( REVENUE[2005] = ROUND(REVENUE[2004] * (REVENUE[2004] /
          REVENUE[2003])) )
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

Output:

```
CLIENT_BRANCH CLIENT_NAME                YEAR    REVENUE
------------- -------------------- ---------- ----------
Bangalore     Avlons                     2003       6300
Bangalore     Summer Harvest             2003       6500
Bangalore     Taj India                  2003       6500
Mumbai        Avlons                     2003       4500
Mumbai        Summer Harvest             2003       4500
Mumbai        Taj India                  2003       4500
Shirdi        Avlons                     2003       6900
Shirdi        Summer Harvest             2003       8500
Shirdi        Taj India                  2003       6500
Bangalore     Avlons                     2004      10500
Bangalore     Summer Harvest             2004      10500
Bangalore     Taj India                  2004      12500
Mumbai        Avlons                     2004       7500
Mumbai        Summer Harvest             2004       9500
Mumbai        Taj India                  2004       5500
Shirdi        Avlons                     2004       6500
Shirdi        Summer Harvest             2004      12500
Shirdi        Taj India                  2004      16500
Bangalore     Avlons                     2005      17500
Bangalore     Summer Harvest             2005      16962
Bangalore     Taj India                  2005      24038
Mumbai        Avlons                     2005      12500
Mumbai        Summer Harvest             2005      20056
Mumbai        Taj India                  2005       6722
Shirdi        Avlons                     2005       6123
Shirdi        Summer Harvest             2005      18382
Shirdi        Taj India                  2005      41885

27 rows selected.
```

Explanation:
The calculations performed in the above example are usually used when modeling sales in a spreadsheet.

To use the **MODEL** clause, the data has to be conceptually formed into a multidimensional array. Each result set row becomes a cell in that array.

Partition Columns
Partition columns divide the result set into blocks. Rules defined in the model clause are applied independently of other partitions to each partition.

When a model is created, begin by ideating about how and whether to partition the model into separate arrays. Partitioning is optional, but it gives the database a point at which to parallelize the work and it makes formulas much easier to write.

In this example, there exists data of the revenues earned based on client branch and the clients name and each (branch and client)'s forecast together is independent of the others, so it is required to partition by branch and the client.

```
MODEL
  PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
```

Dimension columns
Dimension columns define how cells within a partition can be accessed.

This is then followed by data dimension. This means deciding which values will combine to uniquely identify a row in the array. For the revenues earned data, the data can be dimensioned by the Year column. For each branch and client, the value held in the year column uniquely identifies a row in the array.

```
MODEL
  PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
  DIMENSION BY (YEAR)
```

Measure columns
The columns defined as measures can be assigned new values in the rules section of the model clause.

Each cell in a model holds one or more values, but the forecasting formula requires just the revenue value, so a column has to be specified as a measure of calculation.

```
MODEL
  PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
  DIMENSION BY (YEAR)
  MEASURES (REVENUE)
```

Once the measure is decided its now time to create a formulae to do the actual projections for the Year 2005. Begin by writing rules, which define the calculations that is to be performed using the measure specified. In this example to forecast the revenues that will be earned in the Year 2005 the following formulae is used:

```
REVENUE[2005] = ROUND(REVENUE[2004] * (REVENUE[2004] /
                                       REVENUE[2003]))
```

The cell reference (called **Positional Referencing**) begins with the column name specified in the MEASURES clause. Next comes a list of dimension values enclosed within square brackets (In this case only one value is used as a dimension).

Finally using cell referencing the formulae is created which accesses the value held in the Year 2004 and 2003 and after performing some calculations on it transfers it to the Year 2005. This is enclosed within RULES.

REMINDER

The RULES keyword, shown in the examples at the start of the rules, is optional, but recommended for easier reading.

Example 2:
Suppose it is desired to project commission earned (Revenue From Clients Of The Personnel Management System) per branch and client for the year 2005 based on the earnings of the years 2003 & 2004. (ONLY UPDATES NO NEW RECORDS)

Solution:

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL   PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
   DIMENSION BY (YEAR)   MEASURES (REVENUE)
   RULES UPDATE( REVENUE[2005] = ROUND(REVENUE[2004] * (REVENUE[2004]
                 / REVENUE[2003])) )
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

Output:

```
CLIENT_BRANCH   CLIENT_NAME                YEAR    REVENUE
--------------- -------------------- ---------- ----------
Bangalore       Avlons                     2003       6300
Bangalore       Summer Harvest             2003       6500
Bangalore       Taj India                  2003       6500
Mumbai          Avlons                     2003       4500
Mumbai          Summer Harvest             2003       4500
Mumbai          Taj India                  2003       4500
Shirdi          Avlons                     2003       6900
Shirdi          Summer Harvest             2003       8500
Shirdi          Taj India                  2003       6500
Bangalore       Avlons                     2004      10500
Bangalore       Summer Harvest             2004      10500
Bangalore       Taj India                  2004      12500
Mumbai          Avlons                     2004       7500
Mumbai          Summer Harvest             2004       9500
Mumbai          Taj India                  2004       5500
Shirdi          Avlons                     2004       6500
Shirdi          Summer Harvest             2004      12500
Shirdi          Taj India                  2004      16500

18 rows selected.
```

Explanation:
This example is same as the example 1 but for the difference that if the data for the year 2005 doesn't exists then no new records will be inserted. Whereas if the data for the year 2005 exists then the data will be updated based on the calculations specified in the RULE.

This is done using the keyword **UPDATE** with RULE (**i.e. RULES UPDATE**). This indicates that the calculation will be applied only on the current available data based on the criteria specified and no new records will be inserted to show the projections.

Example 3:
Suppose it is desired to project commission earned (Revenue From Clients Of The Personnel Management System) per branch and client for the year 2005 & 2006 and 2007 & 2008 based on the earnings of the years 2003 & 2004 and 2005 & 2006 respectively. Use **FOR LOOP**.

Solution:

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL
   PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
   DIMENSION BY (YEAR)
   MEASURES (REVENUE)
   RULES (
      REVENUE[FOR YEAR IN(2005,2006)] = ROUND(REVENUE[2004] *
                                        (REVENUE[2004] / REVENUE[2003])),
      REVENUE[FOR YEAR IN(2007,2008)] = ROUND(REVENUE[2006] *
                                        (REVENUE[2006] / REVENUE[2005]))
   )
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

Output:

CLIENT_BRANCH	CLIENT_NAME	YEAR	REVENUE
Bangalore	Avlons	2003	6300
Bangalore	Summer Harvest	2003	6500
Bangalore	Taj India	2003	6500
Mumbai	Avlons	2003	4500
Mumbai	Summer Harvest	2003	4500
Mumbai	Taj India	2003	4500
Shirdi	Avlons	2003	6900
Shirdi	Summer Harvest	2003	8500
Shirdi	Taj India	2003	6500
Bangalore	Avlons	2004	10500
Bangalore	Summer Harvest	2004	10500
Bangalore	Taj India	2004	12500
Mumbai	Avlons	2004	7500
Mumbai	Summer Harvest	2004	9500
Mumbai	Taj India	2004	5500
Shirdi	Avlons	2004	6500
Shirdi	Summer Harvest	2004	12500
Shirdi	Taj India	2004	16500
Bangalore	Avlons	2005	17500

Output: (Continued)

CLIENT_BRANCH	CLIENT_NAME	YEAR	REVENUE
Bangalore	Summer Harvest	2005	16962
Bangalore	Taj India	2005	24038
Mumbai	Avlons	2005	12500
Mumbai	Summer Harvest	2005	20056
Mumbai	Taj India	2005	6722
Shirdi	Avlons	2005	6123
Shirdi	Summer Harvest	2005	18382
Shirdi	Taj India	2005	41885
Bangalore	Avlons	2006	17500
Bangalore	Summer Harvest	2006	16962
Bangalore	Taj India	2006	24038
Mumbai	Avlons	2006	12500
Mumbai	Summer Harvest	2006	20056
Mumbai	Taj India	2006	6722
Shirdi	Avlons	2006	6123
Shirdi	Summer Harvest	2006	18382
Shirdi	Taj India	2006	41885
Bangalore	Avlons	2007	17500
Bangalore	Summer Harvest	2007	16962
Bangalore	Taj India	2007	24038
Mumbai	Avlons	2007	12500
Mumbai	Summer Harvest	2007	20056
Mumbai	Taj India	2007	6722
Shirdi	Avlons	2007	6123
Shirdi	Summer Harvest	2007	18382
Shirdi	Taj India	2007	41885
Bangalore	Avlons	2008	17500
Bangalore	Summer Harvest	2008	16962
Bangalore	Taj India	2008	24038
Mumbai	Avlons	2008	12500
Mumbai	Summer Harvest	2008	20056
Mumbai	Taj India	2008	6722
Shirdi	Avlons	2008	6123
Shirdi	Summer Harvest	2008	18382
Shirdi	Taj India	2008	41885

54 rows selected.

Explanation:

The **MODEL** clause provides a **FOR** construct that can be used inside rules to express computations more concisely. The **FOR** construct is allowed on both sides of rules.

In the above example, a **FOR LOOP** is used to specify **RULES** for more than one year, thus avoids multiple entries for the same formulae.

If the above solution was written **without** the word **FOR:**

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL
   PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
   DIMENSION BY (YEAR)
   MEASURES (REVENUE)
   RULES (
      REVENUE[YEAR IN(2005,2006)] = ROUND(REVENUE[2004] *
                                    (REVENUE[2004] / REVENUE[2003])),
      REVENUE[YEAR IN(2007,2008)] = ROUND(REVENUE[2006] *
                                    (REVENUE[2006] / REVENUE[2005]))
   )
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

If the word **FOR** is omitted then no new rows will be added. This means that only if there exists any rows belonging to the Years 2005-2008 will be updated. In this case the output would be as seen below. As there are no rows belonging to the Years 2005-2008 only the old rows are displayed.

Output:

```
CLIENT_BRANCH   CLIENT_NAME                YEAR    REVENUE
--------------- -------------------- ---------- ----------
Bangalore       Avlons                     2003       6300
Bangalore       Summer Harvest             2003       6500
Bangalore       Taj India                  2003       6500
Mumbai          Avlons                     2003       4500
Mumbai          Summer Harvest             2003       4500
Mumbai          Taj India                  2003       4500
Shirdi          Avlons                     2003       6900
Shirdi          Summer Harvest             2003       8500
Shirdi          Taj India                  2003       6500
Bangalore       Avlons                     2004      10500
Bangalore       Summer Harvest             2004      10500
Bangalore       Taj India                  2004      12500
Mumbai          Avlons                     2004       7500
Mumbai          Summer Harvest             2004       9500
Mumbai          Taj India                  2004       5500
Shirdi          Avlons                     2004       6500
Shirdi          Summer Harvest             2004      12500
Shirdi          Taj India                  2004      16500

18 rows selected.
```

REMINDER

The **MODEL** clause has a **limit** of 10,000 rules and the virtual rules generated by **FOR** constructs are counted toward that limit. It is important to consider the total number of rules potentially generated by **FOR** constructs to avoid exceeding the rule limit.

Example 4:
Suppose it is desired to project commission earned (Revenue From Clients Of The Personnel Management System) per branch and client for every alternate year ranging from 2005 to 2010 based on the rental transactions of the years 2003 & 2004. Use **FOR LOOP FROM** With INCREMENT.

Solution:

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL
  PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
  DIMENSION BY (YEAR)
  MEASURES (REVENUE)
  RULES (
      REVENUE[FOR YEAR FROM 2005 TO 2010 INCREMENT 2]
          = ROUND(REVENUE[2004] * (REVENUE[2004]/REVENUE[2003]))
  )
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

Output:

CLIENT_BRANCH	CLIENT_NAME	YEAR	REVENUE
Bangalore	Avlons	2003	6300
Bangalore	Summer Harvest	2003	6500
Bangalore	Taj India	2003	6500
Mumbai	Avlons	2003	4500
Mumbai	Summer Harvest	2003	4500
Mumbai	Taj India	2003	4500
Shirdi	Avlons	2003	6900
Shirdi	Summer Harvest	2003	8500
Shirdi	Taj India	2003	6500
Bangalore	Avlons	2004	10500
Bangalore	Summer Harvest	2004	10500
Bangalore	Taj India	2004	12500
Mumbai	Avlons	2004	7500
Mumbai	Summer Harvest	2004	9500
Mumbai	Taj India	2004	5500
Shirdi	Avlons	2004	6500
Shirdi	Summer Harvest	2004	12500
Shirdi	Taj India	2004	16500
Bangalore	**Avlons**	**2005**	**17500**
Bangalore	**Summer Harvest**	**2005**	**16962**
Bangalore	**Taj India**	**2005**	**24038**
Mumbai	**Avlons**	**2005**	**12500**
Mumbai	**Summer Harvest**	**2005**	**20056**
Mumbai	**Taj India**	**2005**	**6722**

Output: (Continued)

CLIENT_BRANCH	CLIENT_NAME	YEAR	REVENUE
Shirdi	Avlons	2005	6123
Shirdi	Summer Harvest	2005	18382
Shirdi	Taj India	2005	41885
Bangalore	Avlons	2007	17500
Bangalore	Summer Harvest	2007	16962
Bangalore	Taj India	2007	24038
Mumbai	Avlons	2007	12500
Mumbai	Summer Harvest	2007	20056
Mumbai	Taj India	2007	6722
Shirdi	Avlons	2007	6123
Shirdi	Summer Harvest	2007	18382
Shirdi	Taj India	2007	41885
Bangalore	Avlons	2009	17500
Bangalore	Summer Harvest	2009	16962
Bangalore	Taj India	2009	24038
Mumbai	Avlons	2009	12500
Mumbai	Summer Harvest	2009	20056
Mumbai	Taj India	2009	6722
Shirdi	Avlons	2009	6123
Shirdi	Summer Harvest	2009	18382
Shirdi	Taj India	2009	41885

45 rows selected.

Explanation:
The **MODEL** clause provides a **FOR FROM INCREMENT** construct that can be used inside rules to express computations more concisely. The **FOR FROM INCREMENT** construct is allowed on both sides of rules.

In the above example, a **FOR FROM INCREMENT LOOP** is used to specify **RULES** for more than one year and for every alternate year, thus avoids multiple entries for the same formulae.

This is done using the following RULE:

```
REVENUE[FOR YEAR FROM 2005 TO 2010 INCREMENT 2]
    = ROUND(REVENUE[2004] * (REVENUE[2004]/REVENUE[2003]))
```

The above rule via the For Loop iterates through the years 2005 till 2010 but skips every next year while performing the calculations. This is because INCREMENT keyword is used with a value 2, which allows iterations to every alternate year and thereby apply calculations on the value extracted.

The **FOR LOOP** requires the range of values to iterate on. This is specified via the **FROM** keyword (i.e. **FROM 2005 TO 2010**).

Example 5:
Suppose it is desired to update the commission earned (Revenue From Clients Of The Personnel Management System) per branch and client for the year 2004 by increasing the revenue by 10%. Use Symbolic References.

Solution:

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL
   PARTITION BY (CLIENT_BRANCH, CLIENT_NAME)
   DIMENSION BY (YEAR)
   MEASURES (REVENUE)
   RULES (REVENUE[YEAR = 2004] = REVENUE[YEAR = 2004] * 1.10)
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

Output:

```
CLIENT_BRANCH   CLIENT_NAME                YEAR    REVENUE
--------------- -------------------- ---------- ----------
Bangalore       Avlons                     2003       6300
Bangalore       Summer Harvest             2003       6500
Bangalore       Taj India                  2003       6500
Mumbai          Avlons                     2003       4500
Mumbai          Summer Harvest             2003       4500
Mumbai          Taj India                  2003       4500
Shirdi          Avlons                     2003       6900
Shirdi          Summer Harvest             2003       8500
Shirdi          Taj India                  2003       6500
Bangalore       Avlons                     2004      11550
Bangalore       Summer Harvest             2004      11550
Bangalore       Taj India                  2004      13750
Mumbai          Avlons                     2004       8250
Mumbai          Summer Harvest             2004      10450
Mumbai          Taj India                  2004       6050
Shirdi          Avlons                     2004       7150
Shirdi          Summer Harvest             2004      13750
Shirdi          Taj India                  2004      18150

18 rows selected.
```

Explanation:
Symbolic references are very powerful, but they are used solely for updating existing cells. They cannot create new cells as required in case of revenue projections for future years (Refer Example 1)

In case of a symbolic reference the value for the cell reference is matched to the appropriate dimension using Boolean conditions. All the normal operators such as <,>, IN and BETWEEN can be used.

For Example:

```
REVENUE[YEAR > 2004] = REVENUE[2003] * 1.10
REVENUE[YEAR < 2004] = REVENUE[YEAR = 2004] * 1.25
REVENUE[YEAR BETWEEN 2001 AND 2004] = REVENUE[YEAR = 1999] * 2
```

(In case of multiple cells)

```
REVENUE[CLIENT_NAME='AVLONS', YEAR<2004]
    = REVENUE[CLIENT_NAME = 'AVLONS', YEAR = 2004] * 1.25
```

In this example the query looks for the Year value equal to 2004 and increases the revenue of that year by 10%.

Example 6:
Suppose it is desired to update the commission earned (Revenue From Clients Of The Personnel Management System) per branch for the years before 2004 by increasing the revenue by 10% only for the **Client** named **Summer Harvest**. Use Symbolic References and multiple dimensions.

Solution:

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL
  PARTITION BY (CLIENT_BRANCH)
  DIMENSION BY (CLIENT_NAME, YEAR)
  MEASURES (REVENUE)
  RULES (REVENUE[CLIENT_NAME = 'SUMMER HARVEST', YEAR < 2004] =
     REVENUE[CLIENT_NAME = 'SUMMER HARVEST', YEAR = 2004] * 1.10)
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

Output:

CLIENT_BRANCH	CLIENT_NAME	YEAR	REVENUE
Bangalore	Avlons	2003	6300
Bangalore	**Summer Harvest**	**2003**	**11550**
Bangalore	Taj India	2003	6500
Mumbai	Avlons	2003	4500
Mumbai	**Summer Harvest**	**2003**	**10450**
Mumbai	Taj India	2003	4500
Shirdi	Avlons	2003	6900
Shirdi	**Summer Harvest**	**2003**	**13750**
Shirdi	Taj India	2003	6500

Output: (Continued)

```
CLIENT_BRANCH   CLIENT_NAME                YEAR    REVENUE
--------------- -------------------- ---------- ----------
Bangalore       Avlons                     2004      10500
Bangalore       Summer Harvest             2004      10500
Bangalore       Taj India                  2004      12500
Mumbai          Avlons                     2004       7500
Mumbai          Summer Harvest             2004       9500
Mumbai          Taj India                  2004       5500
Shirdi          Avlons                     2004       6500
Shirdi          Summer Harvest             2004      12500
Shirdi          Taj India                  2004      16500

18 rows selected.
```

Explanation:

In this example the query looks for the Year value less than 2004 and increases the revenue of that year by 10% based on the revenue of the year 2004. This increase is only done for the client named Summer Harvest. To accomplish this a multi dimension array is used:

```
DIMENSION BY (Client_Name, Year)
```

The increase is done using the following RULE:

```
Revenue[Client_Name = 'Summer Harvest', Year < 2004] =
    Revenue[Client_Name = 'Summer Harvest', Year = 2004] * 1.10
```

This simply indicates that the Revenue value for the years less than 2004 (in this case being 2003) will be increased by 10% of the value held in the revenue column of the year 2004. In addition the update / increase is done only for the client named Summer Harvest and not for the other clients.

Example 7:

Suppose it is desired to forecast the commission earned (Revenue From Clients Of The Personnel Management System) for the year 2005 for the client named Avlons as 1000 more than the maximum revenue in the period 2003 to 2004. Use the BETWEEN clause.

Solution:

```
SELECT CLIENT_BRANCH, CLIENT_NAME, YEAR, REVENUE FROM CLIENTREVENUE
MODEL
  PARTITION BY (CLIENT_BRANCH)
  DIMENSION BY (CLIENT_NAME, YEAR)
  MEASURES (REVENUE)
  RULES (REVENUE['AVLONS', 2005]
      = 1000 + MAX(REVENUE) ['AVLONS', YEAR BETWEEN 2003 AND 2004])
ORDER BY YEAR, CLIENT_BRANCH, CLIENT_NAME;
```

Output:

```
CLIENT_BRANCH   CLIENT_NAME                YEAR    REVENUE
--------------- -------------------- ---------- ----------
Bangalore       Avlons                     2003       6300
Bangalore       Summer Harvest             2003       6500
Bangalore       Taj India                  2003       6500
Mumbai          Avlons                     2003       4500
Mumbai          Summer Harvest             2003       4500
Mumbai          Taj India                  2003       4500
Shirdi          Avlons                     2003       6900
Shirdi          Summer Harvest             2003       8500
Shirdi          Taj India                  2003       6500
Bangalore       Avlons                     2004      10500
Bangalore       Summer Harvest             2004      10500
Bangalore       Taj India                  2004      12500
Mumbai          Avlons                     2004       7500
Mumbai          Summer Harvest             2004       9500
Mumbai          Taj India                  2004       5500
Shirdi          Avlons                     2004       6500
Shirdi          Summer Harvest             2004      12500
Shirdi          Taj India                  2004      16500
Bangalore       Avlons                     2005      11500
Mumbai          Avlons                     2005       8500
Shirdi          Avlons                     2005       7900

21 rows selected.
```

Explanation:

In this example, the query projects the Revenues for the Year 2005 based on the maximum Revenue earned between in the years 2003-2004 and adds up 1000 to the maximum value extracted. This RULE is applied only to the client **Avlons**. To accomplish this multi dimensional array is used:

```
DIMENSION BY (CLIENT_NAME, YEAR)
```

The projection is done using the following RULE:

```
REVENUE['AVLONS', 2005]
    = 1000 + MAX(REVENUE) ['AVLONS', YEAR BETWEEN 2003 AND 2004]
```

This simply indicates that the **REVENUE** value for the year 2005 will be projected as the maximum revenue earned from the client named **Avlons** in the years 2003 – 2004 plus 1000.

Since it's not an update but an insertion positional reference is used i.e. REVENUE['Avlons', 2005]

Example 8:

Generate a series of dates between two dates.

Solution:

```
SELECT DT FROM (SELECT TRUNC(SYSDATE) DT FROM DUAL)
MODEL
  DIMENSION BY (0 D)
  MEASURES (DT)
  RULES ITERATE(10) (DT[ITERATION_NUMBER + 1]
     = DT[ITERATION_NUMBER] + 1);
```

Output:

```
DT
---------
23-MAR-05
24-MAR-05
25-MAR-05
26-MAR-05
27-MAR-05
28-MAR-05
29-MAR-05
30-MAR-05
31-MAR-05
01-APR-05
02-APR-05

11 rows selected.
```

Explanation:
In this example, the query generates a series of dates beginning from the current date. This is done using the system variable SYSDATE.

The array is dimensioned using current date, (i.e. value of SYSDATE given an alias as DT) as follows:

```
MODEL DIMENSION BY (0 D)
```

Here, **0** (zero) indicates the first column extracted by the query (in this case a single column **DT** is extracted). **D** indicates that an alias has been assigned to that column.

The series is generated using the **ITERATE()** function combined with the **RULES** keyword. In this case, ITERATE(10) indicates that the specified rule should be **executed ten times**.

The value of the date is generated using the following rule:

```
DT[ITERATION_NUMBER + 1] = DT[ITERATION_NUMBER] + 1
```

Here, DT is the name of an array and its first element holds the current date. The variable ITERATION_NUMBER is a running number maintained to generate a series of date. This is done by assigning a value, (i.e. current date + 1) indicated by **DT[**ITERATION_NUMBER] + 1 to the next element of the array indicated by **DT[**ITERATION_NUMBER + 1**]**.

Merging Rows Using MERGE

Recently Oracle database introduced the MERGE statement that allows merging rows from one table into another.

Example:
Assume the table **NEWCLIENT** stores details of the changes in the client's details. It will be a need to consolidate the contents of the tables **CLIENT** and **NEWCLIENT**. The structure for the **NEWCLIENT** table will be based on the following **CREATE TABLE** statement:

```
CREATE TABLE NEWCLIENT (CLNT_USER_ID VarChar2(7),
  CLNT_ID Number(4) NOT NULL, Clnt_Name VarChar2(35),
  Clnt_Code VarChar2(5), Clnt_Url VarChar2(50), Clnt_Desc VarChar2(255),
  PRIMARY KEY (CLNT_ID));
```

The contents of the **NEWCLIENT** table will be as follows:

Table Name: NEWCLIENT

CLNT_USER_ID	CLNT_ID	CLNT_NAME	CLNT_CODE	CLNT_URL	CLNT_DESC
US00006	1	SCT India	**SCT1**	**http://sctindia.com**	**IT Firm**
US00007	2	ABCL Corp	**ABCL2**	http://abcl.com	**Laptop Resellers**
US00008	3	Dell Corp	**DELL3**	**http://del.com**	**Desktop & Laptops**
US00009	**4**	**Compaq**	**COMP4**	**http://compaq.com**	**Laptops**

The records can be populated into the **NEWCANDIDATE** table using the following **INSERT INTO** statements:

```
INSERT INTO NEWCLIENT
  VALUES ('US00006',1,'SCT India', 'SCT1','http://sctindia.com','IT Firm');
INSERT INTO NEWCLIENT
  VALUES ('US00007',2,'ABCL Corp', 'ABCL2','http://abcl.com','Laptop Resellers');
INSERT INTO NEWCLIENT
  VALUES ('US00008',3,'Dell Corp', 'DELL3','http://del.com','Desktop And Laptops');
INSERT INTO NEWCLIENT
  VALUES ('US00009',4,'Compaq', 'COMP4','http://compaq.com','Laptops');
COMMIT;
```

Data Held in Client Table (Before Applying Merge)

```
CLNT_US  CLNT_ID CLNT_NAME CLNT_CODE CLNT_URL         CLNT_DESC
------- -------- --------- --------- --------------- ---------
US00006        1 SCT India
US00007        2 ABCL Corp           http://abcl.com
US00008        3 Dell Corp
```

Data Held in NewClient Table (Before Applying Merge)

```
CLNT_US  CLNT_ID CLNT_NAME CLNT_CODE CLNT_URL            CLNT_DESC
-------- -------- --------- --------- ------------------ ------------------
US00006         1 SCT India SCT1      http://sctindia.com IT Firm
US00007         2 ABCL Corp ABCL2     http://abcl.com     Laptop Resellers
US00008         3 Dell Corp DELL3     http://del.com      Desktop And Laptops
US00009         4 Compaq    COMP4     http://compaq.com   Laptops
```

On exercising the **MERGE** statement the following is expected:

- Rows that already exists in the **CLIENT** table based on a comparison with the **NEWCLIENT** table using the **CLNT_ID** column are updated with new values held by the **NEWCLIENT** table
- Rows that are not available in the **CLIENT** table but present in the **NEWCLIENT** table are inserted into the **CLIENT** table

The following statement performs the merge as defined:

```
MERGE INTO CLIENT C USING NEWCLIENT NC ON (C.CLNT_ID = NC.CLNT_ID)
WHEN MATCHED THEN
  UPDATE SET C.CLNT_USER_ID=NC.CLNT_USER_ID,
      C.CLNT_NAME=NC.CLNT_NAME, C.CLNT_CODE=NC.CLNT_CODE,
      C.CLNT_URL=NC.CLNT_URL, C.CLNT_DESC=NC.CLNT_DESC
WHEN NOT MATCHED THEN
  INSERT (C.CLNT_USER_ID, C.CLNT_ID, C.CLNT_NAME, C.CLNT_CODE,
      C.CLNT_URL, C.CLNT_DESC)
      VALUES (NC.CLNT_USER_ID, NC.CLNT_ID, NC.CLNT_NAME, NC.CLNT_CODE,
         NC.CLNT_URL, NC.CLNT_DESC);
```

Output:

```
4 rows merged.
```

Data Held in Client Table (After Applying Merge)

```
CLNT_US  CLNT_ID CLNT_NAME CLNT_CODE CLNT_URL            CLNT_DESC
-------- -------- --------- --------- ------------------ ------------------
US00006         1 SCT India SCT1      http://sctindia.com IT Firm
US00007         2 ABCL Corp ABCL2     http://abcl.com     Laptop Resellers
US00008         3 Dell Corp DELL3     http://del.com      Desktop And Laptops
US00009         4 Compaq    COMP4     http://compaq.com   Laptops
```

The following should be remembered while using the MERGE statement:

- The **MERGE INTO** clause specifies the name of the table to merge the rows into
- The **USING ... ON** clause specifies a table join
- The **WHEN MATCHED THEN** clause specifies the action to take when the **USING ... ON** clause is satisfied for a row
- The **WHEN NOT MATCHED** clause specifies the action to take when the **USING ... ON** clause is not satisfied for a row

Hierarchical Queries

Quite often data that is organized into a hierarchy is required. Such data may include information like people who work in an organization, a family tree and so on. Oracle 10g facilitates arranging such information into a hierarchy.

The elements of the hierarchy also called nodes form a hierarchy tree. Trees of nodes have technical terms associated with them such as:

- **Root node:** The root is the node at the top of the tree. In the case of the **EMPLOYEE** table, it will be the employee designated as the CEO or the President of the company
- **Parent node:** A parent is a node that has one or more nodes beneath it
- **Child node:** A child is a node that has one parent node above it
- **Leaf node:** A leaf is a node that has no children

Syntax:

```
SELECT [LEVEL], <ColumnName>, <Expression>, ... FROM <TableName>
  [WHERE <Condition>] [ [START WITH <StartCondition>]
     [CONNECT BY PRIOR <PriorCondition>] ];
```

where,

- **LEVEL** is a pseudo-column that denotes the depth of tree. It returns 1 for the root node, 2 for the child of the root and so on.
- **StartCondition** specifies where to start point for the hierarchical query. It is specified with START WITH clause.
- **PriorCondition** specifies the relationship between the parent and child rows. It is specified with the CONNECT BY PRIOR clause.

REMINDER

The **CONNECT BY** and **START WITH** clause is a requirement to perform hierarchical queries.

The **LEVEL** pseudo-column is an indication of how deep in the tree one is. Oracle can handle queries with a depth of up to **255** levels.

The **START WITH** clause is used to specify the start of the tree. More than one record can match the starting condition. The **CONNECT BY PRIOR** clause cannot be used to perform a join to other tables. The **CONNECT BY PRIOR** clause is rarely implemented in the other database offerings. Trying to achieve this programmatically will be difficult, as the top-level query has to be coded first, then, for each of the records, open a cursor to look for child nodes.

One way of working around this is to use PL/SQL, open the driving cursor with the ***CONNECT BY PRIOR*** *statement and then select matching records from other tables on a row-by-row basis, inserting the results into a temporary table for later retrieval.*

To exercise some examples on hierarchical queries create a table named **EMPLOYEE**, which will hold all those candidates who have been hired as follows:

```
CREATE TABLE EMPLOYEE(EMPNO Number(4), ENAME VarChar2(10),
  JOB VarChar2(9), MGR Number(4), HIREDATE Date, SAL Number(7, 2),
  COMM Number(7, 2), DEPTNO Number(2));
```

Populate the employee table with the following insert statements:

```
INSERT INTO EMPLOYEE VALUES ('7369', 'Anil', 'CLERK', '7902', '17-DEC-80', 800,
                             NULL, 20);
INSERT INTO EMPLOYEE VALUES ('7499', 'Rahul', 'SALESMAN', '7698', '20-FEB-
                             81', 1600, 300, 30);
INSERT INTO EMPLOYEE VALUES ('7521', 'Amit', 'SALESMAN', '7698', '22-FEB-81',
                             1250, 500, 30);
INSERT INTO EMPLOYEE VALUES ('7566', 'Chhaya', 'MANAGER', '7839', '02-APR-
                             81', 2975, NULL, 20);
INSERT INTO EMPLOYEE VALUES ('7654', 'Sunil', 'SALESMAN', '7698', '28-SEP-
                             81', 1250, 1400, 30);
INSERT INTO EMPLOYEE VALUES ('7698', 'Ashwini', 'MANAGER', '7839', '01-MAY-
                             81', 2850, NULL, 30);
INSERT INTO EMPLOYEE VALUES ('7782', 'Vaishali', 'MANAGER', '7839', '09-JUN-
                             81', 2450, NULL, 10);
INSERT INTO EMPLOYEE VALUES ('7788', 'Sharanam', 'ANALYST', '7566', '19-APR-
                             87', 3000, NULL, 20);
INSERT INTO EMPLOYEE VALUES ('7839', 'Ivan', 'PRESIDENT', NULL, '17-NOV-81',
                             5000, NULL, 10);
INSERT INTO EMPLOYEE VALUES ('7844', 'Stuti', 'SALESMAN', '7698', '08-SEP-81',
                             1500, 0, 30);
INSERT INTO EMPLOYEE VALUES ('7876', 'Pooja', 'CLERK', '7788', '23-MAY-87',
                             1100, NULL, 20);
INSERT INTO EMPLOYEE VALUES ('7900', 'Anu', 'CLERK', '7698', '03-DEC-81',
                             950, NULL, 30);
INSERT INTO EMPLOYEE VALUES ('7902', 'Hansel', 'ANALYST', '7566', '03-DEC-
                             81', 3000, NULL, 20);
INSERT INTO EMPLOYEE VALUES ('7934', 'Abhijeet', 'CLERK', '7782', '23-JAN-82',
                             1300, NULL, 10);
```

Using The CONNECT BY And START WITH Clause

Example:
Generate a hierarchy based on the information in the **EMPLOYEE** table:

```
SELECT EMPNO, MGR, ENAME FROM EMPLOYEE
  START WITH EMPNO=7839 CONNECT BY PRIOR EMPNO = MGR;
```

Output:

```
     EMPNO        MGR ENAME
---------- ---------- ----------
      7839            Ivan
      7566       7839 Chhaya
      7788       7566 Sharanam
      7876       7788 Pooja
      7902       7566 Hansel
      7369       7902 Anil
      7698       7839 Ashwini
      7499       7698 Rahul
      7521       7698 Amit
      7654       7698 Sunil
      7844       7698 Stuti
      7900       7698 Anu
      7782       7839 Vaishali
      7934       7782 Abhijeet

14 rows selected.
```

Using The LEVEL Pseudo-Column

Example:
The following query uses the LEVEL pseudo-column to display the level in the tree generated with the information in the **EMPLOYEE** table:

```
SELECT LEVEL, EMPNO, MGR, ENAME FROM EMPLOYEE START WITH EMPNO=7839
  CONNECT BY PRIOR EMPNO = MGr ORDER BY LEVEL;
```

Output:

```
     LEVEL      EMPNO        MGR ENAME
---------- ---------- ---------- ----------
         1       7839            Ivan
         2       7566       7839 Chhaya
         2       7782       7839 Vaishali
         2       7698       7839 Ashwini
         3       7788       7566 Sharanam
         3       7499       7698 Rahul
         3       7654       7698 Sunil
         3       7934       7782 Abhijeet
         3       7900       7698 Anu
         3       7844       7698 Stuti
         3       7521       7698 Amit
         3       7902       7566 Hansel
         4       7876       7788 Pooja
         4       7369       7902 Anil

14 rows selected.
```

Example:
The following query uses the **COUNT()** function and **LEVEL** to get the number of levels in the tree generated on processing the information in the **Employee** table:

```
SELECT COUNT(DISTINCT LEVEL) FROM EMPLOYEE
  START WITH EMPNO=7839 CONNECT BY PRIOR EMPNO = MGR;
```

Output:

```
COUNT(DISTINCTLEVEL)
--------------------
                   4
```

Formatting The Result From A Hierarchical Query

The results from a hierarchical query can be formatted using **LEVEL** and the **LPAD()** function.

Example:
The following query left-pads the values with spaces in order of the **LEVEL** on the hierarchy generated from the **EMPLOYEE** table:

```
COLUMN EMPLOYEE FORMAT A25
SELECT LEVEL, LPAD(' ', 2*LEVEL-1) || ENAME AS EMPLOYEE FROM EMPLOYEE
  START WITH EMPNO=7839 CONNECT BY PRIOR EMPNO = MGR;
```

REMINDER

The command **COLUMN** Employee **FORMAT A25** restricts the **EMPLOYEE** column to a length of 25 characters.

Output:

```
     LEVEL EMPLOYEE
---------- --------------
         1  Ivan
         2    Chhaya
         3      Sharanam
         4        Pooja
         3      Hansel
         4        Anil
         2    Ashwini
         3      Rahul
         3      Amit
         3      Sunil
         3      Stuti
         3      Anu
```

Output: (Continued)

```
     LEVEL EMPLOYEE
---------- --------------
         2    Vaishali
         3      Abhijeet

14 rows selected.
```

The above example generates an Employee hierarchy. This hierarchy indicates the positions held by the employees. For Example: Ivan is the boss of Chhaya, Ashwini, and Vaishali. Similarly Chhaya is the boss of Sharanam and Hansel. Further Pooja reports to Sharanam and Anil reports to Hansel and so on.

Starting At A Node Other Than The Root

The starting point for traversing the nodes of a tree can be changed by simply modifying the condition associated with the **START WITH** clause.

REMINDER

Irrespective of the root node the **LEVEL** for the starting point in the hierarchical query is always set to **1**.

Example:
The following query starts at employee named **Chhaya** while generating the hierarchical query from the **EMPLOYEE** table:

```
COLUMN EMPLOYEE FORMAT A25
SELECT LEVEL, LPAD(' ', 2*LEVEL-1) || ENAME AS EMPLOYEE FROM EMPLOYEE
  START WITH ENAME='Chhaya' CONNECT BY PRIOR EMPNO = MGR;
```

Output:

```
     LEVEL EMPLOYEE
---------- ------------
         1  Chhaya
         2    Sharanam
         3      Pooja
         2    Hansel
         3      Anil
```

Traversing Upward Through The Tree

The traversing of nodes in a tree can be reversed (i.e. from child to parent instead of parent to child). This can be achieved by inversing the child and parent column in the **CONNECT TO PRIOR** clause.

Example:
The following query starts at the employee named **Anil**, who is at the lower-most level in the employee's hierarchy tree.

```
COLUMN EMPLOYEE FORMAT A25
SELECT LEVEL, LPAD(' ', 2*LEVEL-1) || ENAME AS EMPLOYEE FROM EMPLOYEE
  START WITH EMPNO= 7369 CONNECT BY PRIOR MGR = EMPNO;
```

Output:

```
     LEVEL EMPLOYEE
---------- --------------
         1  Anil
         2    Hansel
         3      Chhaya
         4        Ivan
```

Eliminating Nodes And Branches From A Hierarchical Query

A particular node from a query tree can be eliminated by using a **WHERE** clause.

Example:
The following query eliminates the employee named Chhaya from the result of the hierarchical query generated from the **EMPLOYEE** table.

```
COLUMN EMPLOYEE FORMAT A25
SELECT LEVEL, LPAD(' ', 2*LEVEL-1) || ENAME AS EMPLOYEE FROM EMPLOYEE
  WHERE ENAME != 'Chhaya' START WITH EMPNO= 7839
    CONNECT BY PRIOR EMPNO = MGR;
```

Output:

```
     LEVEL EMPLOYEE
---------- -----------------
         1  Ivan
         3     Sharanam
         4       Pooja
         3     Hansel
         4       Anil
         2    Ashwini
         3      Rahul
         3      Amit
         3      Sunil
         3      Stuti
         3      Anu
         2    Vaishali
         3      Abhijeet
13 rows selected.
```

REMINDER

Although the entry for Chhaya is been excluded for the result, all subsequent child nodes have been included in the result.

The RETURNING Clause

In Oracle Database 10g, the **RETURNING** clause can be used to return the value from an aggregate functions such as **AVG()**, **SUM()**, **COUNT()** and so on.

Example:
The following example performs the following tasks:

- Declares a variable named **Avg_Sal**
- Increases the salary column of the rows in the Employee table (created earlier) and saves the average salary in the **avg_sal** variable using the **RETURNING** clause
- Rolls back the update
- Prints the value of the **avg_sal** variable

Solution:

```
VARIABLE avg_sal NUMBER
UPDATE EMPLOYEE SET SAL = SAL * 1.10 RETURNING AVG(SAL) INTO :avg_sal;
```

Output:

```
14 rows updated.
```

```
ROLLBACK;
```

Output:

```
Rollback complete.
```

```
PRINT avg_sal
```

Output:

```
   AVG_SAL
----------
2280.53571
```

Matrix ReportS

Example:
Create a Matrix Report that displays the Categories, Candidates and the number of candidates registered per category.

Solution:

```
SELECT * FROM (SELECT B.CAND_ID "Candidate",
  DECODE(B.CAND_CATEGORY, '1', (SELECT COUNT(CAND_ID)
     FROM CANDIDATE R WHERE R.CAND_ID IN (SELECT CAND_ID
        FROM CANDIDATE WHERE CANDIDATE.CAND_CATEGORY
           = B.CAND_Category))) "Continental Cook",
  DECODE(B.CAND_CATEGORY, '2', (SELECT COUNT(CAND_ID)
     FROM CANDIDATE R WHERE R.CAND_ID IN (SELECT CAND_ID
        FROM CANDIDATE WHERE CANDIDATE.CAND_CATEGORY
           = B.CAND_CATEGORY))) "Arabic Cook",
  DECODE(B.CAND_CATEGORY, '3', (SELECT COUNT(CAND_ID)
     FROM CANDIDATE R WHERE R.Cand_ID IN (SELECT CAND_ID
        FROM CANDIDATE WHERE CANDIDATE.CAND_CATEGORY
           = B.CAND_CATEGORY))) "Analyst Programmer",
  DECODE(B.Cand_Category, '4', (SELECT COUNT(Cand_ID)
     FROM Candidate R WHERE R.Cand_ID IN (SELECT Cand_ID
        FROM CANDIDATE WHERE CANDIDATE.CAND_CATEGORY
           = B.CAND_CATEGORY))) "Scaffolding Foreman",
  DECODE(B.CAND_CATEGORY, '5', (SELECT COUNT(CAND_ID)
     FROM CANDIDATE R WHERE R.CAND_ID IN (SELECT CAND_ID
        FROM Candidate WHERE CANDIDATE.CAND_CATEGORY
           = B.CAND_CATEGORY))) "Cabin Steward"
  FROM CATEGORY A, CANDIDATE B
     WHERE A.CATEGORY_ID = B.CAND_CATEGORY
        GROUP BY B.CAND_ID, B.CAND_CATEGORY);
```

Output:

```
Candida Continental Cook Arabic Cook Analyst Programmer
Scaffolding Foreman Cabin Steward
------- ---------------- ----------- ------------------ -------
------------ -------------
CD00001                                                 3
CD00002                1
CD00003                                                 3
CD00004                                                 3
CD00005                             1
```

In the above example, **DECODE** is used which is responsible to actually create a matrix output. This means that wherever the **CAND_CATEGORY** column returns a value other than **1 → Continental Cook** (the same applies to all the other categories in the query, i.e. Arabic Cook, Analyst Programmer and so on) a blank value is displayed. This makes it possible to create a matrix report.

DECODE is a handy value-substitution mechanism that returns plain-English equivalents for a coded field. One of its advantage is speed. It is much faster to query using the DECODE keyword than to perform a join to a lookup table, especially when using large tables.

Using A Matrix Report To Count Data Values In A Column

Example:
Create a report displaying the Candidates, the number of English, Hindi, Arabic, German speaking and a total of all categories based on their Identity.

```
SELECT CAND_ID,
  SUM(DECODE(LANG_ID, '1', 1, 0)) "English",
  SUM(DECODE(LANG_ID, '2', 1, 0)) "Arabic",
  SUM(DECODE(LANG_ID, '3', 1, 0)) "Hindi",
  SUM(DECODE(LANG_ID, '4', 1, 0)) "German",
  SUM(DECODE(LANG_ID, '5', 1, 0)) "French",
  SUM(DECODE(LANG_ID, '6', 1, 0)) "Tamil",
  SUM(DECODE(LANG_ID, '7', 1, 0)) "Urdu",
  SUM(DECODE(LANG_ID, '8', 1, 0)) "Malayalam",
  SUM(DECODE(LANG_ID, '9', 1, 0)) "Marathi",
  COUNT(LANG_ID) "TOTAL"
  FROM CAND_LANG GROUP BY CAND_ID;
```

Output:

```
CAND_ID English Arabic Hindi German French Tamil Urdu Malayalam
Marathi TOTAL
------- ------- ------ ----- ------ ------ ----- ---- ---------
------- -----
CD00001       1      0     0      0      0     0    0         0
      0     1
CD00002       1      0     0      0      0     0    0         0
      0     1
CD00003       0      1     0      0      0     0    0         0
      0     1
CD00004       1      0     1      0      1     0    0         0
      0     3
CD00005       0      0     1      0      0     1    0         0
      0     2
```

Create a CSV output

SQLPLUS can be a great tool to produce a quick report from Oracle database. As an example assume it is necessary to produce a Comma Separated Values (CSV), output file. At first glance the user might start by appending the comma character to the database fields in the select statement.

However the solution below shows that this can be done quite easily using some of the built-in SQLPLUS commands. It also demonstrates some other commands that might be useful and can be used as a skeleton script upon which to base other reports.

Example:
Generate a report on the documents submitted by the candidates.

Solution:

```
/* Suppress page headers, titles and all formatting */
SET PAGESIZE 0
/* Switch off the SQL text before/after any variable substitution
*/
SET VERIFY OFF
/* Set line size, make this as big as desired */
SET LINES 700
/* Delete any blank spaces at the end of each spooled line */
SET TRIMSPOOL ON
/* Switch off the lines number display returned by the query */
SET FEEDBACK OFF
/* Switch off SELECT output to the screen */
SET TERMOUT OFF
/* Separate each column by a comma character (CSV output) */
SET COLSEP ','
/* Put the SELECT output into a file*/
SPOOL Docs_Report.txt
SELECT      CAND_DOC_NO,      CAND_DOC_NAME,      CAND_DOC_TYPE,
CAND_DOC_ISSUE_DAT,
  CAND_DOC_EXP_DAT, CAND_DOC_ISSUE_PLACE FROM CAND_DOC;
SPOOL OFF
```

The output can be seen, by browsing to the bin directory available under **$ORACLE_HOME/bin.** Open the file **Docs_Report.txt**

Output:

```
SQL> /* Suppress page headers, titles and all formatting */
SQL> SET PAGESIZE 0
SQL>
SQL> /* Switch off the SQL text before/after any variable substitution */
SQL> SET VERIFY OFF
SQL>
SQL> /* Set line size, make this as big as desired */
SQL> SET LINES 700
SQL>
SQL> /* Delete any blank spaces at the end of each spooled line */
SQL> SET TRIMSPOOL ON
SQL>
SQL> /* Switch off the lines number display returned by the query */
SQL> SET FEEDBACK OFF
SQL>
SQL> /* Switch off SELECT output to the screen */
SQL> SET TERMOUT OFF
SQL>
SQL> /* Separate each column by a comma character (CSV output) */
SQL> SET COLSEP ','
SQL>
SQL> /* Put the SELECT output into a file*/
SQL> SPOOL Docs_Report.txt
SQL> SELECT Cand_Doc_No, Cand_Doc_Name, Cand_Doc_Type, Cand_Doc_Issue_Dat,
  2    Cand_Doc_Exp_Dat, Cand_Doc_Issue_Place FROM Cand_Doc;
E3097703            ,Passport        ,                    ,01/11/2002  ,30/10/2012  ,Trivandrum
T2110/99            ,Driving License ,Light Vehicles      ,07/09/1999  ,06/08/2009  ,Trivandrum
A1865183            ,Passport        ,                    ,12/08/1996  ,11/08/2006  ,Mumbai
MH019855835         ,Driving License ,Light Vehicles      ,27/11/1998  ,26/11/2016  ,Mumbai
a123434             ,Passport        ,                    ,12/12/2000  ,12/12/2005  ,Mumbai
qw333               ,Driving License ,Light Vehicles      ,11/12/2000  ,12/12/2005  ,Gujarat
B.3451531           ,Passport        ,                    ,01/03/2001  ,01/03/2011  ,Mumbai
A-4624525           ,Passport        ,                    ,22/01/1998  ,21/01/2008  ,Trivandrum
T/9871/1995         ,Driving License ,Light Vehicles      ,10/10/1995  ,09/10/2015  ,Trivandrum
SQL>
SQL> SPOOL OFF
SQL>
```

Assist The DBA

Changing The Oracle Password

Example:
Update the password for the Oracle user named **vaishali** to **sct005**.

Solution 1:

```
ALTER USER vaishali IDENTIFIED BY sct005;
```

Output:

```
User altered.
```

Oracle 8 onwards this can simply be done as:
Solution 2: To change the password for another User via the SQL*PLUS tool

```
Password vaishali;
```

Output:

```
Changing password for vaishali
New password: ******
Retype new password: ******
Password changed
```

Solution 3: To change the password for the Current User via the SQL*PLUS tool

```
Password;
```

Output:

```
Changing password for DBA_PM
Old password: *******
New password: *******
Retype new password: *******
Password changed
```

Query Flashback

A Query Flashback can be used to revert mistakenly committed changes and to view original records before the COMMIT was executed. If required, the results of a query flashback can be used to manually change rows back to their original values.

In addition, flashbacks can be based on a **datetime** or **System Change Number** (SCN). The database uses SCNs to track changes made to data and these can be used to flash back to a particular SCN in the database.

Granting The Privilege For Using Flashbacks

Flashbacks use the PL/SQL DBMS_FLASHBACK package, for which the EXECUTE privilege is a must.

Example:
The following example connects as the **sys** user and grants the **EXECUTE** privilege on DBMS_FLASHBACK to the **DBA_PM** user:

```
CONNECT sys/<password_for_sys> AS sysdba
GRANT EXECUTE ON SYS.DBMS_FLASHBACK TO DBA_PM;
```

Using The Time Query Flashbacks

Example:
The following example connects as **DBA_PM** and retrieves the **CAND_ID** and **CAND_SAL_EXPCTD** columns from the **CANDIDATE** table:

```
CONNECT DBA_PM/<password>
SELECT CAND_ID, CAND_SAL_EXPCTD FROM CANDIDATE;
```

Output:

```
CAND_ID CAND_SAL_EXPCTD
------- ---------------
CD00001 55000
CD00002 7000
CD00003 25000
CD00004 27000
CD00005 36000
```

The following statement reduces the **CAND_SAL_EXPCTD** of these rows, commits the change and retrieves the rows again to get the new salary expectations (i.e. **CAND_SAL_EXPCTD**):

```
UPDATE CANDIDATE SET CAND_SAL_EXPCTD=CAND_SAL_EXPCTD * 0.95;
```

Output:

```
5 rows updated.
```

```
COMMIT;
```

Output:

```
Commit complete.
```

```
SELECT CAND_ID, CAND_SAL_EXPCTD FROM CANDIDATE;
```

Output:

```
CAND_ID CAND_SAL_EXPCTD
------- ---------------
CD00001 52250
CD00002 6650
CD00003 23750
CD00004 25650
CD00005 34200
```

To understand the flaskback concept, execute the DBMS_FLASHBACK.ENABLE_AT_TIME() procedure, which enables the performance of a flashback to a particular datetime. This

procedure accepts a datetime. The example passes **SYSDATE - 10 / 1440** to the procedure, which means **ten minutes back:**

```
EXECUTE DBMS_FLASHBACK.ENABLE_AT_TIME(SYSDATE - 10 / 1440);
```

Output:

```
PL/SQL procedure successfully completed.
```

Any queries executed after the Flashback will display the rows as they were ten minutes ago. Assuming the earlier UPDATE was performed less than ten minutes ago, the following query will display the candidate's salary expectation as they were before the **CANDIDATE** table was updated.

```
SELECT CAND_ID, CAND_SAL_EXPCTD FROM CANDIDATE;
```

Output:

```
CAND_ID CAND_SAL_EXPCTD
------- ---------------
CD00001 55000
CD00002 7000
CD00003 25000
CD00004 27000
CD00005 36000
```

To disable flashback, execute **DBMS_FLASHBACK.DISABLE()**, as shown in the following example:

```
EXECUTE DBMS_FLASHBACK.DISABLE();
```

Output:

```
PL/SQL procedure successfully completed.
```

WARNING

A flashback should be disabled before it can be enabled again.

When the queries are performed after disabling the flashback, the rows will be retrieved, as they currently exist.

```
SELECT CAND_ID, CAND_SAL_EXPCTD FROM CANDIDATE;
```

Output:

```
CAND_ID CAND_SAL_EXPCTD
------- ---------------
CD00001 52250
CD00002 6650
CD00003 23750
CD00004 25650
CD00005 34200
```

Using The System Change Number Query Flashbacks

Flashbacks based on system change numbers (SCNs) can be more precise than those based on a time, because the database uses SCNs to track changes. To get the current SCN, execute the DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER() function as shown below:

```
VARIABLE current_scn NUMBER
EXECUTE :current_scn :=
  DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
```

Output:

```
PL/SQL procedure successfully completed.
```

```
PRINT current_scn
```

Output:

```
CURRENT_SCN
-----------
     787624
```

Example:
The following adds a row to the **COUNTRY** table, commits the change and retrieves the newly inserted row:

```
INSERT INTO Country VALUES(99, 'Kenya');
```

Output:

```
1 row created.
```

```
COMMIT;
```

Output:

```
Commit complete.
```

```
SELECT * FROM COUNTRY;
```

Output:

```
COUNTRY_ID COUNTRY_NAME
---------- -------------------------
         1 India
         2 United States Of America
         3 United Kingdoms
         4 United Arab Emirates
         5 Pakistan
         6 Sri Lanka
         7 Bangladesh
        99 Kenya

8 rows selected.
```

Finally, execute the DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER() procedure, which enables the performance of a flashback to an SCN. This procedure accepts an SCN, and the example passes **current_scn** to the procedure:

```
EXECUTE
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER(:current_scn);
```

Output:

```
PL/SQL procedure successfully completed.
```

Any queries executed after the Flashback will display the rows as they were at the SCN stored in **current_scn** before the **INSERT INTO** statement.

Example:
The following query will attempt to retrieve the newly inserted record for the **COUNTRY** table.

```
SELECT * FROM COUNTRY;
```

Output:

```
COUNTRY_ID COUNTRY_NAME
---------- -------------------------
         1 India
         2 United States Of America
         3 United Kingdoms
         4 United Arab Emirates
         5 Pakistan
         6 Sri Lanka
         7 Bangladesh

7 rows selected.
```

The query does not return the desired record because the new row was added after the SCN was stored in the variable **current_scn**.

To disable a flashback, execute the **DBMS_FLASHBACK.DISABLE()** function, as shown in the following example:

```
EXECUTE DBMS_FLASHBACK.DISABLE();
```

After Disabling The Flashback:

```
SELECT * FROM COUNTRY;
```

Output:

```
COUNTRY_ID COUNTRY_NAME
---------- -------------------------
         1 India
         2 United States Of America
         3 United Kingdoms
         4 United Arab Emirates
         5 Pakistan
         6 Sri Lanka
         7 Bangladesh
        99 Kenya

8 rows selected.
```

Flashback Table

Consider a situation where the Category table has been accidentally deleted.

```
SQL> DROP TABLE Category;
Table dropped.
SQL>
```

Now, with Oracle 10g the table has been moved to the recycle bin rather than actually deleted.

To check this, issue the **SHOW RECYCLEBIN** command.

```
SQL> SHOW RECYCLEBIN;
ORIGINAL NAME  RECYCLEBIN NAME                 OBJECT TYPE
DROP TIME
-------------- ------------------------------ ------------
-------------------
CATEGORY       BIN$h9iXx6X4Rk6rKRz45M6WUA==$0 TABLE
2005-03-23:10:26:59
SQL>
```

Now, to restore the table Category, issue the **FLASHBACK TABLE** <TABLE_NAME> **TO BEFORE DROP** command.

```
SQL> FLASHBACK TABLE Category TO BEFORE DROP;
Flashback complete.
```

If instead it is desired to clear the table down completely and thus release the space taken up by the table, issue the **PURGE RECYCLEBIN** command.

```
SQL> DROP TABLE Category;
Table dropped.
```

Verify It as:

```
SQL> SHOW RECYCLEBIN;
ORIGINAL NAME    RECYCLEBIN NAME                OBJECT TYPE
DROP TIME
---------------- ------------------------------ ------------
-------------------
CATEGORY         BIN$QoUs+n9XTU6NtVONmEMmnQ==$0 TABLE
2005-03-23:10:29:16
```

Purge The RecycleBin as:

```
SQL> PURGE RECYCLEBIN;
Recyclebin purged.
```

Verify It as:

```
SQL> SHOW RECYCLEBIN;
SQL>
```

What actually happens is that the table being dropped is actually renamed to a system defined name, starting with BIN$ rather than the table being actually dropped. The flashback table feature is intelligent enough to be able to manage multiple stored copies of the same table, if the table is dropped and recreated with a different structure. It also purges old copies of recycled tables if the database starts to run short of space.

If desired to drop the table Category without sending it to the RecycleBin then issue **DROP TABLE** <TABLE_NAME> **PURGE** to drop the table and skip the RecycleBin in the first place. *This is something similar to using **SHIFT+DEL** Key Stroke to delete a file permanently.*

Drop the table without sending it to RecycleBin:

```
SQL> DROP TABLE Category PURGE;
Table dropped.
```

Verify It as:

```
SQL> SHOW RECYCLEBIN;
SQL>
```

Oracle Sample Schemas

Other then the SCOTT schema, Oracle 10g now provides many sample schemas to work with. These are:

```
SQL> SELECT USERNAME FROM DBA_USERS;
```

Output:

```
USERNAME
------------------------------
SYSTEM
SYS
. . .
HR
OE
SH
PM
IX
BI
SCOTT
. . .
32 rows selected.
```

By default these accounts are inactive and locked. To use them, these accounts need to be unlocked and assigned a new password. This can be done as follows:

```
ALTER USER "HR" IDENTIFIED BY "<password>" ACCOUNT UNLOCK;
```

Output:

```
User altered.
```

```
ALTER USER "OE" IDENTIFIED BY "<password>" ACCOUNT UNLOCK;
```

Output:

```
User altered.
```

```
ALTER USER "SH" IDENTIFIED BY "<password>" ACCOUNT UNLOCK;
```

Output:

```
User altered.
```

```
ALTER USER "PM" IDENTIFIED BY "<password>" ACCOUNT UNLOCK;
```

Output:

```
User altered.
```

```
ALTER USER "IX" IDENTIFIED BY "<password>" ACCOUNT UNLOCK;
```

Output:

```
User altered.
```

Once these user accounts are unlocked these schemas, can be used by logging in via SQL *PLUS with their appropriate user name and password.

Assuming SQL *PLUS is open type in the following commands to connect to different user accounts and display the tables held within them.

Connecting To User Named HR:

```
SQL> CONNECT HR/<password>@<HostStringIfAny>;
SQL> SELECT * FROM TAB;
```

Output:

```
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
REGIONS                        TABLE
COUNTRIES                      TABLE
LOCATIONS                      TABLE
DEPARTMENTS                    TABLE
JOBS                           TABLE
EMPLOYEES                      TABLE
JOB_HISTORY                    TABLE
EMP_DETAILS_VIEW               VIEW

8 rows selected.
```

Connecting To User Named OE:

```
SQL> CONNECT OE/<password>@<HostStringIfAny>;
SQL> SELECT * FROM TAB;
```

Output:

```
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
CUSTOMERS                      TABLE
WAREHOUSES                     TABLE
ORDER_ITEMS                    TABLE
ORDERS                         TABLE
INVENTORIES                    TABLE
PRODUCT_INFORMATION            TABLE
PRODUCT_DESCRIPTIONS           TABLE
PROMOTIONS                     TABLE
COUNTRIES                      SYNONYM
LOCATIONS                      SYNONYM
DEPARTMENTS                    SYNONYM
JOBS                           SYNONYM
EMPLOYEES                      SYNONYM
JOB_HISTORY                    SYNONYM
PRODUCTS                       VIEW
SYDNEY_INVENTORY               VIEW
BOMBAY_INVENTORY               VIEW
TORONTO_INVENTORY              VIEW
PRQDUCT_PRICES                 VIEW
ACCOUNT_MANAGERS               VIEW
CUSTOMERS_VIEW                 VIEW
ORDERS_VIEW                    VIEW
DEPTVIEW                       VIEW
PURCHASEORDERS                 TABLE
STYLESHEET_TAB                 TABLE
CATEGORIES_TAB                 TABLE
PRODUCT_REF_LIST_NESTEDTAB     TABLE
SUBCATEGORY_REF_LIST_NESTEDTAB TABLE
OC_INVENTORIES                 VIEW
OC_PRODUCT_INFORMATION         VIEW
OC_CUSTOMERS                   VIEW
OC_CORPORATE_CUSTOMERS         VIEW
OC_ORDERS                      VIEW

33 rows selected.
```

Connecting To User Named SH:

```
SQL> CONNECT SH/<password>@<HostStringIfAny>;
SQL> SELECT * FROM TAB;
```

Output:

```
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
SALES                          TABLE
COSTS                          TABLE
TIMES                          TABLE
PRODUCTS                       TABLE
CHANNELS                       TABLE
PROMOTIONS                     TABLE
CUSTOMERS                      TABLE
COUNTRIES                      TABLE
SUPPLEMENTARY_DEMOGRAPHICS     TABLE
MVIEW$_EXCEPTIONS              TABLE
CAL_MONTH_SALES_MV             TABLE
FWEEK_PSCAT_SALES_MV           TABLE
DR$SUP_TEXT_IDX$I              TABLE
DR$SUP_TEXT_IDX$K              TABLE
DR$SUP_TEXT_IDX$R              TABLE
DR$SUP_TEXT_IDX$N              TABLE
PROFITS                        VIEW
SALES_TRANSACTIONS_EXT         TABLE

18 rows selected.
```

Connecting To User Named PM:

```
SQL> CONNECT PM/<password>@<HostStringIfAny>;
SQL> SELECT * FROM TAB;
```

Output:

```
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
ONLINE_MEDIA                   TABLE
PRINT_MEDIA                    TABLE
TEXTDOCS_NESTEDTAB             TABLE
```

Connecting To User Named IX:

```
SQL> CONNECT IX/<password>@<HostStringIfAny>;
SQL> SELECT * FROM TAB;
```

Output:

```
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
ORDERS_QUEUETABLE              TABLE
AQ$_ORDERS_QUEUETABLE_S        TABLE
AQ$_ORDERS_QUEUETABLE_T        TABLE
AQ$_ORDERS_QUEUETABLE_H        TABLE
SYS_IOT_OVER_49905             TABLE
AQ$_ORDERS_QUEUETABLE_G        TABLE
AQ$_ORDERS_QUEUETABLE_I        TABLE
STREAMS_QUEUE_TABLE            TABLE
AQ$_STREAMS_QUEUE_TABLE_S      TABLE
AQ$_STREAMS_QUEUE_TABLE_T      TABLE
AQ$_STREAMS_QUEUE_TABLE_H      TABLE
SYS_IOT_OVER_49922             TABLE
AQ$_STREAMS_QUEUE_TABLE_G      TABLE
AQ$_STREAMS_QUEUE_TABLE_I      TABLE
AQ$ORDERS_QUEUETABLE_S         VIEW
AQ$ORDERS_QUEUETABLE_R         VIEW
AQ$ORDERS_QUEUETABLE           VIEW
AQ$STREAMS_QUEUE_TABLE_S       VIEW
AQ$STREAMS_QUEUE_TABLE_R       VIEW
AQ$STREAMS_QUEUE_TABLE         VIEW

20 rows selected.
```

As can be seen from the above output each schema, i.e. the two letter name stands for the following:

HR	Human resources, basic topics, supports Oracle Internet Directory
OE	Order entry, intermediate topics, various datatypes
SH	Sales history, large amount of data, analytic processing
PM	Product media, used for multimedia datatypes
IX	Queued shipping, shows advanced queuing

By introducing these sample schemas, students learning Oracle and instructors teaching Oracle now have better resources at hand to use and explore when it comes to testing out different Oracle capabilities. These schemas can be used to practice SQL and PL/SQL.

Hands On Exercises

Using the tables created previously generate the SQL statements for the operations mentioned below.

1. Perform the following computations on table data:

a. List the names of all authors having 'a' as the second letter in their names.

b. List the authors who stay in a city whose First letter is 'M'.

c. List all authors who stay in 'Mumbai' or 'Chennai'.

d. List names of all books whose number of copies sold more than Rs. 1000 in the first quarter for the year 2004.

e. List all information on print cycles for books printed in second half of 2004.

f. List the royalty information for authors 'A00004' and 'A00005'.

g. List books whose selling price is greater than 200 and less than or equal to 300.

h. List books whose selling price is less than 300. Calculate and display a new selling price as original selling price plus 15%. Rename the new column in the output of the above query as new_price.

i. List the names, city and state of authors who are not born in the months of January, June or July.

j. Count the total number of copies sold and give the result a proper name.

k. Calculate the average price of all the books.

l. Determine the maximum and minimum prices. Rename the output as max_price and min_price respectively.

m. Count the number of authors having their royalty percent less than or equal to 8.

n. List all the all the book IDs whose copies on hand have gone below 1000.

2. Exercise on Date Manipulation:

a. List the book ID number and day on which the contract for the book was made.

b. List the month (in alphabets) , date and book ID when the print cycle have commenced.

c. List the author's date of birth in the format 'DD-Month-YY'. e.g. 12-February-02.

d. List the date, 15 days after today's date.

3. Exercises on using Having and Group By Clauses:

a. Print the book ISBN and total copies sold for each manuscript.

b. List the authors and the royalty they are entitled to in the financial year 2004-2005.

c. List names of all books whose sales are more than Rs. 300000 in the last quarter for the year 2004.

d. Find out the total sales and royalty for books sold in the period JAN05-MAR05.

e. Count the total number of copies sold on a per book basis. Give a proper heading for the aggregate.

4. Exercises on Joins and Correlation:

a. Find out the books, which have been authored by persons resident in India.

b. Find out the books and number of copies that have been printed in the current month.

c. List the ISBN and names of books, which have sold at least 2500 copies in a single quarter of financial year 2004-2005.

d. Find the names of books authors by 'Felice DeCaper'.

e. List the periods in which more than 3000 copies of 'Tales Of The Shadow - I' have been sold.

f. Find the ISBN and total copies sold for the books authored by 'Thomas Hunt' and 'Abdul M. Khan'.

5. Exercise on Sub-queries:

a. Find the ISBN and names of non-moving books (i.e. books not being sold) for the first quarter of 2005.

b. List the customer Name, Address, City and Pincode for the author(s) who has authored 'Development In PHP'.

c. List the names of books that have contracts made before May'03.

d. List all authors who stay in 'United Kingdom' or 'U. S. A.'

e. List the royalty information for authors 'R. Krishna' and 'Parvati Patil'.

f. List the names, city and state of authors who are not in the country of 'India'.

g. List the authors, total sales of all book authored and total royalty received by authors of bestseller, ranged in the price of Rs.150 to 350. A book is considered a best seller when it sales more than 10000 copies.

SECTION II: WORKING WITH ORACLE 10g

Solutions To Hands On Exercises

6. BASIC INTERACTION WITH SQL

1. SQL Statement for creating the Tablespace:

```
CREATE TABLESPACE SCT_PUB DATAFILE 'Pub_Info.dat' SIZE 5M
    DEFAULT STORAGE(INITIAL 10k NEXT 50k MINEXTENTS 1 MAXEXTENTS 499
                    PCTINCREASE 10)
ONLINE;
```

2. SQL Statement for creating the User:

```
CREATE USER "DBA_PUB" PROFILE "DEFAULT" IDENTIFIED BY "sct2306"
    DEFAULT TABLESPACE "SCT_PUB" TEMPORARY TABLESPACE "TEMP"
    ACCOUNT UNLOCK;
```

3. SQL Statement for granting the user permission of an Oracle DBA:

```
GRANT "DBA" TO "DBA_PUB" WITH ADMIN OPTION;
```

4. SQL Statement for creating the tables:

a) **Table Name:** CntryMstr

```
CREATE TABLE CntryMstr(CntryID  Number(3) PRIMARY KEY,
    CntryName VarChar2(40) NOT NULL);
```

b) **Table Name:** AthrMstr

```
CREATE TABLE AthrMstr(AthrID VarChar2(6) PRIMARY KEY,
    Name VarChar2(45) NOT NULL, BirthDt Date NOT NULL, Addr1 VarChar2(50),
    City VarChar2(20), PinCode VarChar2(10), State VarChar2(20),
    CntryID Number(3) REFERENCES CntryMstr);
```

c) **Table Name:** AthrCnct

```
CREATE TABLE AthrCnct(AthrID VarChar2(6) REFERENCES AthrMstr,
    CnctType VarChar2(1) CHECK(CnctType IN('T', 'F', 'M', 'E' , 'U')),
    Contact VarChar2(50));
```

d) **Table Name:** DmsnMstr

```
CREATE TABLE DmsnMstr(DmsnID Number(3) PRIMARY KEY,
    Name VarChar2(15) DEFAULT NULL, BkHght Number(4, 2) NOT NULL,
    BkWdth Number(4, 2) NOT NULL,
    MsrgUnt VarChar2(1) CHECK(MsrgUnt IN('C', 'I')));
```

e) **Table Name:** BkMstr

```
CREATE TABLE BkMstr(BkID VarChar2(6) PRIMARY KEY,
    Title VarChar2(60) NOT NULL, Subject VarChar2(255),
    ISBN VarChar2(16) NOT NULL, Price Number(6, 2), Pages Number(4),
    DmsnID Number(3) REFERENCES DmsnMstr);
```

f) **Table Name:** CntrctMstr

```
CREATE TABLE CntrctMstr(CntrctID VarChar2(6) PRIMARY KEY,
    CntrctDt Date NOT NULL, AthrID VarChar2(6) REFERENCES AthrMstr,
    BkID VarChar2(6) REFERENCES BkMstr, RyltPcnt Number(6, 2) NOT NULL);
```

g) **Table Name:** PrntCycle

```
CREATE TABLE PrntCycle(PrntID VarChar2(6) PRIMARY KEY, PrntDt Date NOT NULL,
    BkID VarChar2(6) REFERENCES BkMstr, Copies Number(4) NOT NULL);
```

h) **Table Name:** Sales

```
CREATE TABLE Sales(SalesID VarChar2(6) PRIMARY KEY,
    SalesPrd VarChar2(12) NOT NULL, BkID VarChar2(6) REFERENCES BkMstr,
    OpStck Number(4) DEFAULT 0, Cmplmtry Number(4) DEFAULT 0,
    Sold Number(5) DEFAULT 0, ClStck Number(4) DEFAULT 0);
```

i) **Table Name:** RyltDtls

```
CREATE TABLE RyltDtls(AthrID VarChar2(6) REFERENCES AthrMstr,
    BkID VarChar2(6) REFERENCES BkMstr, FnclYr VarChar2(9) NOT NULL,
    SalesAmt Number(10, 2) NOT NULL, DscntAmt Number(8, 2),
    TrdAmt Number(10, 2), RyltAmt Number(8, 2));
```

5. SQL Statement for inserting into their respective tables:

a) Data for **CntryMstr** table:

```
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(1, 'India');
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(2, 'Bangladesh');
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(3, 'Pakisthan');
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(4, 'Sri Lanka');
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(5, 'United Kingdom');
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(6, 'U. S. A.');
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(7, 'France');
INSERT INTO CntryMstr (CntryID, CntryName) VALUES(8, 'Canada');
```

b) Data for **AthrMstr** table:

```
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00001', 'Dr. C. K. Shah', TO_DATE('12-03-1965', 'DD-MM-YYYY'),
            'Shahpur, 5, R. P. Lane,', 'Mumbai', '400016', 'Maharashtra', 1);
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00002', 'Thomas Hunt', TO_DATE('30-01-1980', 'DD-MM-YYYY'),
            '343, Apple Street,', 'York', '', '', 5);
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00003', 'Abdul M. Khan', TO_DATE('05-12-1970', 'DD-MM-YYYY'),
            'Hussain House, F-12, Rd. No. 14,', 'Islamabad', '', '', 3);
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00004', 'R. Krishna', TO_DATE('25-06-1955', 'DD-MM-YYYY'), '',
            'Chennai', '600020', 'Tamil Nadu', 1);
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00005', 'Parvati Patil', TO_DATE('21-01-1974', 'DD-MM-YYYY'), '201,
            Fatima C.H.S, S. V. Road, Jogeshwari (W)', 'Mumbai', '400102',
            'Maharashtra', 1);
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00006', 'Felice DeCaper', TO_DATE('11-02-1979', 'DD-MM-YYYY'), '',
            'Paris', '', '', 7);
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00007', 'Hemant Rana', TO_DATE('15-08-1970', 'DD-MM-YYYY'),
            '714-W, Maple Crossing,', 'Toronto', '', '', 8);
INSERT INTO AthrMstr (AthrID, Name, BirthDt, Addr1, City, Pincode, State,
                      CntryID)
    VALUES ('A00008', 'Jonny Parker', TO_DATE('06-10-1981', 'DD-MM-YYYY'), '',
            'Alabama', '', 'Texas', 6);
```

c) Data for **AthrCnct** table:

```
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00001', 'T', '+91 022 24310051');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00001', 'U', 'www.heatlhyshah.com');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00002', 'M', '+44 9300458790');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00002', 'U', 'www.pacsystems.com');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00003', 'T', '+92 015 5442542');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00003', 'E', 'abdulkhan@hotmail.com');
```

Data for **AthrCnct** table: **(Continued)**

```
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00004', 'T', '+91 066 28568462');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00004', 'E', 'supports@ramanworks.com');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00005', 'T', '+91 022 32654584');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00005', 'E', 'pparvati@rediffmail.com');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00006', 'T', '+33 2546464');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00006', 'U', 'www.feliceideas.com');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00007', 'T', '+1 145841656');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00007', 'E', 'hemantrana@usa.net');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00008', 'T', '+1 154648715');
INSERT INTO AthrCnct (AthrID, CnctType, Contact)
    VALUES ('A00008', 'E', 'jonnyparker@hotmail.com');
```

d) Data for **DmsnMstr** table:

```
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(1, 'Letter', 11, 8.5, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(2, 'A4', 11.69, 8.27, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(3, 'A5', 8.27, 5.83, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(4, 'Foolscap', 43, 34, 'C');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(5, 'B5 (JIS)', 10.12, 7.17, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(6, 'B5 (ISO)', 9.84, 6.93, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(7, 'Crown', 19, 12, 'C');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
    VALUES(8, 'Royal', 63, 51, 'C');
```

e) Data for **BkMstr** table:

```
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00001', 'Fighting Fit', 'Health - Physical Fitness', 'A003-01-0001-
            A03', 120, 150, 6);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00002', 'Business And Policies', 'Economics - Finances', 'A003-11-
            0030-A05', 275, 325, 3);
```

Data for **BkMstr** table: **(Continued)**

```
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00003', 'Concepts Of E-Business', 'Business Management', 'A003-
            31-0120-L35', 200, 150, 4);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00004', 'Development In C++', 'Computers - Programming', 'A004-
            02-0080-C51', 225, 400, 3);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00005', 'Development In PHP', 'Computers - Programming', 'A004-
            31-0120-L35', 450, 550, 6);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00006', 'Stress Controls', 'Health - Physical Fitness', 'A004-05-0201-
            F74', 150, 120, 7);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00007', 'Modern Economics', 'Economics - Principles', 'A004-62-
            0250-P13', 275, 350, 3);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00008', 'Tales Of The Shadow - I', 'Fictions - Horror', 'A004-80-
            3247-X37', 100, 250, 7);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00009', 'Concepts Of Networking', 'Business Management', 'A005-
            08-1020-C35', 180, 150, 4);
INSERT INTO BkMstr (BkID, Title, Subject, ISBN, Price, Pages, DmsnID)
    VALUES ('B00010', 'Tales Of The Shadow - II', 'Fictions - Horror', 'A005-11-
            0047-X17', 100, 250, 7);
```

f) Data for **CntrctMstr** table:

```
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00001', TO_DATE('22-05-2002', 'DD-MM-YYYY'), 'A00001', 'B00001',
            10);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00002', TO_DATE('15-09-2002', 'DD-MM-YYYY'), 'A00002', 'B00002',
            12);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00003', TO_DATE('10-01-2003', 'DD-MM-YYYY'), 'A00003', 'B00003',
            12);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00004', TO_DATE('18-06-2003', 'DD-MM-YYYY'), 'A00004', 'B00004',
            5);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00005', TO_DATE('18-06-2003', 'DD-MM-YYYY'), 'A00005', 'B00004',
            5);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00006', TO_DATE('27-11-2003', 'DD-MM-YYYY'), 'A00006', 'B00005',
            5);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00007', TO_DATE('27-11-2003', 'DD-MM-YYYY'), 'A00007', 'B00005',
            5);
```

Data for **CntrctMstr** table: **(Continued)**

```
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00008', TO_DATE('05-01-2004', 'DD-MM-YYYY'), 'A00001', 'B00006',
            10);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00009', TO_DATE('07-03-2004', 'DD-MM-YYYY'), 'A00002', 'B00007',
            12);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00010', TO_DATE('22-05-2004', 'DD-MM-YYYY'), 'A00008', 'B00008',
            5);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00011', TO_DATE('16-11-2004', 'DD-MM-YYYY'), 'A00003', 'B00009',
            12);
INSERT INTO CntrctMstr (CntrctID, CntrctDt, AthrID, BkID, RyltPcnt)
    VALUES ('C00012', TO_DATE('22-12-2004', 'DD-MM-YYYY'), 'A00008', 'B00010',
            5);
```

g) Data for **PrntCycle** table:

```
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00001', TO_DATE('15-03-2003', 'DD-MM-YYYY'), 'B00001', 3000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00002', TO_DATE('09-05-2003', 'DD-MM-YYYY'), 'B00002', 3000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00003', TO_DATE('21-08-2003', 'DD-MM-YYYY'), 'B00003', 9000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00004', TO_DATE('16-09-2003', 'DD-MM-YYYY'), 'B00002', 3000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00005', TO_DATE('31-10-2003', 'DD-MM-YYYY'), 'B00001', 5000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00006', TO_DATE('29-12-2003', 'DD-MM-YYYY'), 'B00004', 8000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00007', TO_DATE('10-01-2004', 'DD-MM-YYYY'), 'B00005', 8000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00008', TO_DATE('31-03-2004', 'DD-MM-YYYY'), 'B00006', 3000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00009', TO_DATE('14-05-2004', 'DD-MM-YYYY'), 'B00002', 3000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00010', TO_DATE('23-06-2004', 'DD-MM-YYYY'), 'B00007', 3000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00011', TO_DATE('12-07-2004', 'DD-MM-YYYY'), 'B00008', 9000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00012', TO_DATE('01-09-2004', 'DD-MM-YYYY'), 'B00005', 5000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00013', TO_DATE('21-11-2004', 'DD-MM-YYYY'), 'B00006', 3000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00014', TO_DATE('12-12-2004', 'DD-MM-YYYY'), 'B00008', 5000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00015', TO_DATE('08-01-2005', 'DD-MM-YYYY'), 'B00009', 9000);
```

Data for **PrntCycle** table: **(Continued)**

```
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00016', TO_DATE('28-01-2005', 'DD-MM-YYYY'), 'B00010', 9000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00017', TO_DATE('07-02-2005', 'DD-MM-YYYY'), 'B00005', 5000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00018', TO_DATE('19-02-2005', 'DD-MM-YYYY'), 'B00002', 5000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00019', TO_DATE('14-03-2005', 'DD-MM-YYYY'), 'B00003', 5000);
INSERT INTO PrntCycle (PrntID, PrntDt, BkID, Copies)
    VALUES ('P00020', TO_DATE('05-04-2005', 'DD-MM-YYYY'), 'B00006', 3000);
```

h) Data for **Sales** table:

```
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00001', 'APR03-JUN03', 'B00001', 3000, 10, 440, 2500);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00002', 'APR03-JUN03', 'B00002', 0, 25, 1575, 1400);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00003', 'JUL03-SEP03', 'B00001', 2500, 20, 1270, 1210);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00004', 'JUL03-SEP03', 'B00002', 1400, 10, 2500, 6890);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00005', 'JUL03-SEP03', 'B00003', 0, 15, 1000, 8985);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00006', 'OCT03-DEC03', 'B00001', 1210, 5, 1055, 5150);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00007', 'OCT03-DEC03', 'B00002', 6890, 0, 1905, 4985);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00008', 'OCT03-DEC03', 'B00003', 8985, 10, 1775, 7200);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00009', 'JAN04-MAR04', 'B00001', 5150, 0, 950, 4200);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00010', 'JAN04-MAR04', 'B00002', 4985, 0, 975, 4010);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00011', 'JAN04-MAR04', 'B00003', 7200, 10, 2025, 5165);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00012', 'JAN04-MAR04', 'B00004', 8000, 30, 1990, 5980);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00013', 'JAN04-MAR04', 'B00005', 0, 20, 2230, 8000);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00014', 'APR04-JUN04', 'B00001', 4200, 0, 980, 3220);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00015', 'APR04-JUN04', 'B00002', 4010, 0, 2570, 4440);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00016', 'APR04-JUN04', 'B00003', 5165, 0, 1865, 3300);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00017', 'APR04-JUN04', 'B00004', 5980, 35, 1965, 3980);
```

Data for **Sales** table: **(Continued)**

```
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00018', 'APR04-JUN04', 'B00005', 5750, 10, 3460, 2280);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00019', 'APR04-JUN04', 'B00006', 3000, 10, 990, 2000);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00020', 'JUL04-SEP04', 'B00001', 3220, 0, 820, 2400);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00021', 'JUL04-SEP04', 'B00002', 4440, 10, 2790, 1640);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00022', 'JUL04-SEP04', 'B00003', 3300, 0, 1300, 2000);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00023', 'JUL04-SEP04', 'B00004', 3980, 20, 2000, 1960);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00024', 'JUL04-SEP04', 'B00005', 2280, 10, 3070, 4200);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00025', 'JUL04-SEP04', 'B00006', 2000, 5, 1000, 995);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00026', 'JUL04-SEP04', 'B00007', 3000, 10, 540, 2450);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00027', 'JUL04-SEP04', 'B00008', 0, 50, 2950, 6000);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00028', 'OCT04-DEC04', 'B00001', 2400, 0, 800, 1600);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00029', 'OCT04-DEC04', 'B00002', 1640, 0, 1100, 540);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00030', 'OCT04-DEC04', 'B00003', 2000, 0, 1125, 875);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00031', 'OCT04-DEC04', 'B00004', 1960, 10, 1950, 0);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00032', 'OCT04-DEC04', 'B00005', 4200, 0, 3250, 950);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00033', 'OCT04-DEC04', 'B00006', 995, 5, 1250, 2740);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00034', 'OCT04-DEC04', 'B00007', 2450, 10, 590, 1850);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00035', 'OCT04-DEC04', 'B00008', 6000, 25, 3275, 7700);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00036', 'JAN05-MAR05', 'B00001', 1600, 0, 705, 895);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00037', 'JAN05-MAR05', 'B00002', 540, 0, 1575, 3965);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00038', 'JAN05-MAR05', 'B00003', 875, 0, 1075, 4800);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00039', 'JAN05-MAR05', 'B00005', 950, 0, 2950, 3000);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00040', 'JAN05-MAR05', 'B00006', 2740, 0, 1570, 1170);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00041', 'JAN05-MAR05', 'B00007', 1850, 5, 645, 1200);
```

Data for **Sales** table: **(Continued)**

```
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00042', 'JAN05-MAR05', 'B00008', 7700, 15, 2780, 4905);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00043', 'JAN05-MAR05', 'B00009', 0, 20, 980, 8000);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00044', 'JAN05-MAR05', 'B00010', 0, 10, 590, 8400);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00045', 'APR05-JUN05', 'B00001', 895, 0, 545, 350);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00046', 'APR05-JUN05', 'B00002', 3965, 5, 1995, 1965);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00047', 'APR05-JUN05', 'B00003', 4800, 5, 1545, 3250);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00048', 'APR05-JUN05', 'B00005', 3000, 0, 2450, 650);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00049', 'APR05-JUN05', 'B00006', 1170, 0, 1670, 2500);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00050', 'APR05-JUN05', 'B00007', 1200, 0, 750, 450);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00051', 'APR05-JUN05', 'B00008', 4905, 5, 2900, 2000);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00052', 'APR05-JUN05', 'B00009', 8000, 10, 1490, 6500);
INSERT INTO Sales (SalesID, SalesPrd, BkID, OpStck, Cmplmtry, Sold, ClStck)
    VALUES ('S00053', 'APR05-JUN05', 'B00010', 8400, 30, 1970, 6400);
```

i) Data for **RyltDtls** table:

```
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00001', 'B00001', '2003-2004', 447000, 44700, 402300, 40230);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00002', 'B00002', '2003-2004', 1912625, 153010, 1759615,
211152.8);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00003', 'B00003', '2003-2004', 960000, 76800, 883200, 105984);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00004', 'B00004', '2003-2004', 447750, 44775, 402975, 20148.75);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00005', 'B00004', '2003-2004', 447750, 44775, 402975, 20148.75);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00006', 'B00005', '2003-2004', 1003500, 100350, 903150,
45157.5);
```

Data for **RyltDtls** table: **(Continued)**

```
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00007', 'B00005', '2003-2004', 1003500, 100350, 903150,
            45157.5);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00001', 'B00001', '2004-2005', 396600, 39660, 356940, 35694);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00002', 'B00002', '2004-2005', 2209625, 176770, 2032855,
            243942.6);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00003', 'B00003', '2004-2005', 1073000, 85840, 987160,
            118459.2);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00004', 'B00004', '2004-2005', 1330875, 133087.5, 1197788,
            59889.38);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00005', 'B00004', '2004-2005', 1330875, 133087.5, 1197788,
            59889.38);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00006', 'B00005', '2004-2005', 5724000, 572400, 5151600,
            257580);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00007', 'B00005', '2004-2005', 5724000, 572400, 5151600,
            257580);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00001', 'B00006', '2004-2005', 721500, 72150, 649350, 64935);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00002', 'B00007', '2004-2005', 488125, 39050, 449075, 53889);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00008', 'B00008', '2004-2005', 900500, 135075, 765425,
            38271.25);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00003', 'B00009', '2004-2005', 176700, 14112, 162288, 19474.56);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00008', 'B00010', '2004-2005', 59000, 8150, 50150, 2507.5);
```

Data for **RyltDtls** table: **(Continued)**

```
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00001', 'B00001', '2005-2006', 65280, 6528, 58752, 5875.2);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00002', 'B00002', '2005-2006', 548625, 43890, 504735, 60568.2);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00003', 'B00003', '2005-2006', 309000, 24720, 284280, 34113.6);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00006', 'B00005', '2005-2006', 1102500, 110250, 992250,
            49612.5);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00007', 'B00005', '2005-2006', 1102500, 110250, 992250,
            49612.5);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00001', 'B00006', '2005-2006', 250500, 25050, 225450, 22545);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00002', 'B00007', '2005-2006', 206250, 16500, 189750, 22770);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00008', 'B00008', '2005-2006', 290000, 43500, 246500, 12325);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00003', 'B00009', '2005-2006', 268200, 21456, 246744, 29669.28);
INSERT INTO RyltDtls (AthrID, BkID, FnclYr, SalesAmt, DscntAmt, TrdAmt,
                      RyltAmt)
    VALUES ('A00008', 'B00010', '2005-2006', 197000, 29550, 167450, 8372.5);

COMMIT;
```

6. SQL Statement for retrieving records from a table:

a. Find out the names of all the authors.

```
SELECT Name FROM AthrMstr;
```

b. Retrieve the entire contents of the BkMstr table.

```
SELECT * FROM BkMstr;
```

c. Retrieve the list of names, city and the state of all the authors.

```
SELECT Name, City, State FROM AthrMstr;
```

d. List the various royalty payable to authors on books.

```
SELECT RyltPcnt, AthrID, BkID FROM CntrctMstr;
```

e. List all the authors who are located in India.

```
SELECT AthrID, Name FROM AthrMstr WHERE CntryID=1;
```

7. SQL Statement for updating records in a table:

a. Change the city of author 'A00005' to 'Bangalore'.

```
UPDATE AthrMstr SET City='Bangalore' WHERE AthrID='A00005';
```

b. Change the royalty of author 'A00001' to 9%.

```
UPDATE CntrctMstr SET RyltPcnt=9 WHERE AthrID='A00001';
```

c. Change the cost price of the book named 'Development In C++' to Rs. 200.

```
UPDATE BkMstr SET Price=200 WHERE Title='Development In C++';
```

8. SQL Statement for deleting records in a table:

a. Delete all sales entries from the Sales table whose sales period is in the last quarter on the year 2004.

```
DELETE Sales WHERE SalesPrd='OCT04-DEC04';
```

b. Delete all contracts for book where the authors are 'R. Krishna' and 'Parvati Patil'.

```
DELETE CntrctMstr WHERE AthrID='A00004' OR AthrID='A00005';
```

c. Delete royalty information, where discounted amount is less than 8% of sales.

```
DELETE RyltDtls WHERE DscntAmt<(SalesAmt*8/100);
```

9. SQL Statement for altering the table structure:

a. Add a column called 'Profession' of data type 'varchar2' and size '20' to the AthrMstr table.

```
ALTER TABLE AthrMstr ADD Profession VarChar2(20);
```

b. Change the size of SalesAmt column in RyltDtls to 12,2.

```
ALTER TABLE RyltDtls MODIFY (SalesAmt Number(12,2));
```

10. SQL Statement for renaming the table

a. Change the name of the Sales table to QtrlyBkSales.

```
RENAME Sales TO QtrlyBkSales;
```

11. SQL Statements for deleting the table structure along with the data

a. Destroy the all tables created for the hands on exercises.

```
DROP TABLE RyltDtls;
DROP TABLE QtrlyBkSales;
DROP TABLE PrntCycle;
DROP TABLE CntrctMstr;
DROP TABLE BkMstr;
DROP TABLE DmsnMstr;
DROP TABLE AthrCnct;
DROP TABLE AthrMstr;
DROP TABLE CntryMstr;
```

7. ADVANCE INTERACTION WITH SQL

1. Generate SQL Statements to perform the following computations on table data:

a. List the names of all authors having 'a' as the second letter in their names.

```
SELECT Name FROM AthrMstr WHERE Name like '_a%';
```

b. List the authors who stay in a city whose First letter is 'M'.

```
SELECT AthrID, Name FROM AthrMstr WHERE UPPER(City) LIKE 'M%';
```

c. List all authors who stay in 'Mumbai' or 'Chennai'.

```
SELECT AthrID, Name FROM AthrMstr WHERE City IN('Chennai', 'Mumbai');
```

d. List names of all books whose number of copies sold more than Rs. 1000 in the first quarter for the year 2004.

```
SELECT BkID, SalesID FROM Sales WHERE Sold > 1000
    AND SalesPrd='JAN04-MAR04';
```

e. List all information on print cycles for books printed in second half of 2004.

```
SELECT * FROM PrntCycle WHERE PrntDt
    BETWEEN TO_DATE('01-07-2004','DD-MM-YYYY')
        AND TO_DATE('31-12-2004','DD-MM-YYYY');
```

f. List the royalty information for authors 'A00004' and 'A00005'.

```
SELECT * FROM RyltDtls WHERE AthrID IN('A00004', 'A00005');
```

g. List books whose selling price is greater than 200 and less than or equal to 300.

```
SELECT BkID, Title, ISBN, Price FROM BkMstr
    WHERE Price > 200 AND Price <= 300;
```

h. List books whose selling price is less than 300. Calculate and display a new selling price as original selling price plus 15%. Rename the new column in the output of the above query as new_price.

```
SELECT BkID, Title, ISBN, Price, Price*1.15 new_price FROM BkMstr
    WHERE Price < 300;
```

i. List the names, city and state of authors who are not born in the months of January, June or July.

```
SELECT Name, City, State FROM AthrMstr
    WHERE TO_CHAR(BirthDt, 'MON') NOT IN('JAN', 'JUN', 'JUL');
```

j. Count the total number of copies sold and give the result a proper name.

```
SELECT SUM(Sold) "No. Of copies sold" FROM Sales;
```

k. Calculate the average price of all the books.

```
SELECT AVG(Price) "Avg. Price of books" FROM BkMstr;
```

l. Determine the maximum and minimum prices. Rename the output as max_price and min_price respectively.

```
SELECT MAX(Price) max_price, MIN(Price) min_price FROM BkMstr;
```

m. Count the number of authors having their royalty percent less than or equal to 8.

```
SELECT COUNT(AthrID) "Royalty < or = 10%" FROM CntrctMstr
    WHERE RyltPcnt <= 8;
```

n. List all the book IDs whose copies on hand have gone below 1000.

```
SELECT DISTINCT BkID, SUBSTR(SalesPrd, 7) "Stock Period Ending", ClStck
    FROM Sales WHERE ClStck < 1000;
```

2. SQL Statements for Date Manipulation:

a. List the book ID number and day on which the contract for the book was made.

```
SELECT BkID, TO_CHAR(CntrctDt, 'Day') FROM CntrctMstr;
```

b. List the month (in alphabets), date and book ID when the print cycle have commenced.

```
SELECT TO_CHAR(PrntDt, 'Month'), PrntDt, BkID FROM PrntCycle
    ORDER BY TO_CHAR(PrntDt, 'Month');
```

c. List the author's date of birth in the format 'DD-Month-YY'. e.g. 12-February-02.

```
SELECT AthrID, Name, TO_CHAR(BirthDt, 'DD-Month-YY') FROM AthrMstr;
```

d. List the date, 15 days after today's date.

```
SELECT SYSDATE + 15 FROM DUAL;
```

3. SQL statements for using Having and Group By Clauses:

a. Print the book ISBN and total copies sold for each manuscript.

```
SELECT BkMStr.BkID, ISBN, SUM(Sold) "No. of Copies Sold" FROM BkMstr, Sales
    WHERE BkMstr.BkID=Sales.BkID GROUP BY BkMstr.BkID, ISBN;
```

b. List the authors and the royalty they are entitled to in the financial year 2004-2005.

```
SELECT R.AthrId, Name, Sum(RyltAmt) Royalty FROM RyltDtls R, AthrMstr A
    WHERE A.AthrID=R.AthrID GROUP BY R.AthrId, Name;
```

c. List names of all books whose sales are more than Rs. 300000 in the last quarter for the year 2004..

```
SELECT B.BkID, Title, Price, Sold, Price*Sold FROM BkMstr B, Sales S
    WHERE B.BkID=S.BkID AND (Price*Sold) > 300000
        AND SalesPrd='OCT04-DEC04';
```

d. Find out the total sales and royalty for books sold in the period JAN05-MAR05.

```
SELECT B.BkID, Title, Price*Sold Sales, Price*(RyltPcnt/100)*Sold "Total Royalty"
    FROM BkMstr B, Sales S, CntrctMstr C
        WHERE B.BkID=S.BkID AND C.BkID=S.BkID
            AND SalesPrd='JAN05-MAR05';
```

e. Count the total number of copies sold on a per book basis. Give a proper heading for the aggregate.

```
SELECT B.BkID, Title, SUM(Sold) "Total Copies Sold" FROM BkMstr B, Sales S
    WHERE B.BkID=S.BkID GROUP BY B.BkID, Title;
```

4. SQL statements for exercises on Joins and Correlation:

a. Find out the books, which have been authored by persons resident in India.

```
SELECT DISTINCT B.BkID, Title
    FROM CntryMstr T, AthrMstr A, CntrctMstr C, BkMstr B
        WHERE UPPER(T.CntryName)='INDIA' AND A.CntryID=T.CntryID
            AND C.AthrID=A.AthrID AND B.BkID=C.BkID;
```

b. Find out the books and number of copies that have been printed in the current month.

```
SELECT B.BkID, Title, SUM(P.Copies) Copies FROM PrntCycle P, BkMstr B
    WHERE P.BkID=B.BkID
        AND TO_CHAR(PrntDt, 'MON-YYYY')=TO_CHAR(SYSDATE, 'MON-YYYY')
            GROUP BY B.BkID, Title;
```

c. List the ISBN and names of books, which have sold at least 2500 copies in a single quarter of financial year 2004-2005.

```
SELECT ISBN, Title, SalesPrd, Sold FROM BkMstr B, Sales S
    WHERE B.BkID=S.BkID AND Sold>=2500
        AND SalesPrd IN('APR04-JUN04', 'JUL04-SEP04', 'OCT04-DEC04', 'JAN05-
MAR05');
```

d. Find the names of books authors by 'Felice DeCaper'.

```
SELECT B.BkID, Title, Subject FROM CntrctMstr C, BkMstr B, AthrMstr A
    WHERE A.Name='Felice DeCaper' AND C.AthrID=A.AthrId
        AND B.BkID=C.BkID;
```

e. List the periods in which more than 3000 copies of 'Tales Of The Shadow - I' have been sold.

```
SELECT B.BkID, SalesPrd FROM Sales S, BkMStr B
    WHERE B.Title='Tales Of The Shadow - I' AND S.BkID=B.BkID
        AND Sold>3000;
```

f. Find the ISBN and total copies sold for the books authored by 'Thomas Hunt' and 'Abdul M. Khan'.

```
SELECT ISBN, SUM(Sold) "Total Copies Sold"
    FROM Sales S, BkMstr B, CntrctMstr C, AthrMstr A
        WHERE S.BkID=B.BkID AND B.BkID=C.BkID AND C.AthrId=A.AthrId
            AND A.Name IN('Thomas Hunt', 'Abdul M. Khan') GROUP BY ISBN;
```

5. SQL statements for exercise on Sub-queries:

a. Find the ISBN and names of non-moving books (i.e. books not being sold) for the first quarter of 2005.

```
SELECT ISBN, Title FROM BkMstr
    WHERE BkID NOT IN(SELECT BkID FROM Sales
        WHERE Sold>0 AND SalesPrd IN('JAN05-MAR05'));
```

b. List the customer Name, Address, City and PinCode for the author(s) who has authored 'Development In PHP'.

```
SELECT Name, Addr1, City, Pincode FROM AthrMstr
    WHERE AthrID IN(SELECT AthrID FROM CntrctMstr
        WHERE BkID IN (SELECT BkID FROM BkMstr
            WHERE Title='Development In PHP'));
```

c. List the names of books that have contracts made before May'03.

```
SELECT BkID, Title FROM BkMstr WHERE BkID IN(SELECT BkID FROM CntrctMstr
    WHERE CntrctDt < TO_DATE('03-2003', 'MM-YYYY'));
```

d. List all authors who stay in 'United Kingdom' or 'U. S. A.'.

```
SELECT AthrID, Name FROM AthrMstr WHERE CntryID IN(SELECT CntryID
    FROM CntryMstr WHERE CntryName='United Kingdom'
        OR CntryName='U. S. A.');
```

e. List the royalty information for authors 'R. Krishna' and 'Parvati Patil'.

```
SELECT FnclYr, ISBN, Title, RyltAmt FROM RyltDtls R, BkMstr B
    WHERE B.BkID=R.BkID AND AthrID IN(SELECT AthrID FROM AthrMstr
        WHERE Name IN('R. Krishna', 'Parvati Patil'));
```

f. List the names, city and country of authors who are not in the country of 'India'.

```
SELECT Name, City, CntryName Country FROM AthrMstr A, CntryMstr C
    WHERE C.CntryID=A.CntryID AND A.CntryID NOT IN(SELECT CntryID
        FROM CntryMstr WHERE UPPER(CntryName)='INDIA');
```

g. List the authors, total sales of all book authored and total royalty received by authors of bestseller, ranged in the price of Rs.150 to 350. A book is considered a best seller when it sales more than 10000 copies.

```
SELECT Name, SUM(R.SalesAmt) "Sales", SUM(R.RyltAmt) "Royalty Received"
    FROM AthrMstr A, RyltDtls R WHERE A.AthrID=R.AthrID
        AND R.BkID IN(SELECT BkID FROM BkMstr WHERE Price
            BETWEEN 150 AND 350)
        AND R.BkID IN(SELECT BkID FROM Sales GROUP BY BkID
            HAVING SUM(Sold)>10000) GROUP BY Name;
```

SECTION III: WORKING WITH PHP

The Language PHP

What Is PHP?

PHP is a Hypertext Pre Processor, created by Rasmus Lerdorf sometime in the year 1994/95.

Hypertext, is just another word for Web Pages. Hence a Hypertext, Pre Processor is actually an interpreter for PHP that has been created for Web servers. This interpreter will process PHP codespec at the Web server and finally dispatch pure HTML code to a browser. Thus PHP is a Web server-side scripting language. The fact that PHP runs on the server has several benefits and a few drawbacks.

PHP is a young language, so its developers have learned from other language's mistakes and have done their best to implement strengths in the PHP interpreter run at the Web Server. PHP code spec is actually simple language, despite its great powers.

Much of PHP syntax is based on **C**. This can be seen in its conditional statements, loop-structures, Boolean operators and the assignment of variables. C is probably the most common programming language today and this should make PHP easy to pick up. Even without any previous experience with C, anyone should be able to learn PHP quickly. Later, the techniques learned from PHP can be applied backwards if programming in C is required.

PHP really has a very simple way of working with different types of memory variables. There's no need to declare a memory variable of a specific type before assigning a value to it. If a number is assigned to a variable, then it just works. When the variable is later outputted to the browser, it just works. PHP takes care of converting the variable from an integer-type to a string-type, **on the fly** and **automatically**. To simplify variable creation and use, memory variables do not even have to be declared, just assign a value to them and then they are ready to use on demand.

Programmers may feel that when variables are declared as above, control of these variables is lost, as even their variable type is not known. This really does not matter as the **value** contained in the **variable** is of use in program processing, **not its type**. If essential, a variable's type can be ascertained, this is useful when checking to see if a parameter passed to a function is really of a given type or not.

Because PHP and HTML codespec are meant to work well together, it has a lot of built in functions to deal with text. PHP is specially designed to deal with Web Pages and does so quickly and efficiently. Most of its built in functions are straightforward to use.

Being web-oriented, PHP also contains built in functions to do things on the Internet. There are functions for connecting to remote Web Servers, checking mail via POP3, IMAP or URL encoding strings to protect special characters and so on.

The History Of PHP

PHP/FI 2.0

The first popular version of PHP was called PHP/FI 2.0, for Personal Home Page / Form Interpreter. It faced some parsing inconsistencies but still managed to attract many programmers as converts.

Issues in PHP/FI 2.0:

- The PHP/FI parser was largely hand-written and so users often encountered scripting errors
- The parser was absolutely tied to Apache web server

These types of issues were finally resolved in version 3, when Zeev Suraski and Andi Gutmans re-wrote PHP from the ground up using standard compiler tools like Flex and Bison.

PHP 3

PHP 3 also made the codespec extensible. This was a serious shortcoming in all prior versions. Particularly keen developers were able to write their own modules for the language, adding functionality to its core.

Issues in PHP 3:

- Code that worked on PHP/FI would no longer work after upgrading

With PHP 3, the language had gained limited object-oriented support. This added a lot to PHP's growth. By the time PHP 3 was replaced in the middle of 2000, it was installed on over 2,500,000 web-site domains, as compared to 250,000 just 18 months before.

PHP 4

PHP 4 was released, in the middle of 2000. This version had major differences from PHP 3 in all aspects. Extensive work had been done to ensure that backwards compatibility with older PHP scripts would remain. *Upgrading from PHP 3 to PHP 4 was much smoother than the PHP/FI to PHP 3 upgrade.*

Perhaps the most important change made for PHP 4 was the switch to what is called the **Zend Engine**. The Zend Engine, created by Zend, a company founded by Zeev Suraski and Andi Gutmans. The name Zend is a contraction of **ZE**ev and a**ND**i. The Zend engine permitted the greatest flexibility than ever before.

Improvements in PHP 4:

- The Zend engine took over the core of PHP and introduced reference counting
- All resources used in scripts such as database connections, files and so on are tracked automatically by the engine and freed when no longer used to minimize memory usage and ensure there were no memory leaks
- PHP now runs on:
  - Apache 1.3.x
  - Apache 2
  - Microsoft's IIS
  - Zeus
  - AOLServer and many more Web servers

  Now more than 40% of the people, who used a Web server other than Apache, began to use PHP.
- Performance increased due to:
  - The execution paradigm in PHP was changed from prior versions. PHP 3 and earlier used an **execute while interpreting** paradigm which meant that PHP read a line of source code, interpreted it, executed it, read another, interpreted it, executed it, read another and so on. *This meant that code was often re-read and re-interpreted twice or more, entirely unnecessarily*
  - PHP 4, with its new **compile first, execute later** paradigm read the entire script and compiled it to byte code before execution, which gave a big boost in speed. *The average speed increase was about 100%, with some benchmarks showing up to a fifty-fold increase in speed when PHP 4 was pushed to its limits.* Because PHP 4 compiled the entire script before executing it, it became possible to optimize and cache the compiled code before execution
  - PHP 4 introduced multi-threading, which essentially allows particularly lengthy, but non-critical functions to be run independently from the main script process, further streamlining execution

PHP 5

PHP 5 was a big step forward for the language, although admittedly not as big as the jump from PHP 3 to PHP 4. This release focused on language maturity and offers a lot of new functionality that has simply been missing from previous versions because the language was a little too simple to properly support really large projects.

Improvements in PHP 5:

- Introduced Object-oriented scripts using which developers are now able to declare how their objects may be used, which makes it easier for one developer to work with another's code
- Wide variety of functions are made available for objects that make them more flexible and easy to work with
- Introduction of error checking in the form of **try/catch**. This is something programmers from other languages had been enjoying for a long time
- Objects are now always handled as references
- SimpleXML which is a fast and easy-to-learn way to interact with XML documents
- The flat-file database API SQLite, a new SOAP extension and much more

Benefits In Running PHP As A Web Server Side Scripts

A PHP script can accomplish the retrieval of data from a database table, its processing and formatting as HTML, prior being dispatched to the client browser. The script can sort through large amounts of table data with ease before this data is downloaded to the client browser.

The PHP script is always invisible to as client browser as the Web server sends only the output of the PHP script to the client browser, **not** the script itself. Hence, PHP scripts are browser neutral **i.e.** they do not depend on some capability **internal** to a browser to run successfully. This also means that no one can steal a carefully crafted PHP script via a browser's 'View Source' capabilities. It's not like JavaScript, everybody is able to read / copy / alter JavaScript using the browser's 'View Source' capabilities for themselves

PHP has no complied binaries it is a **parsed** language. Every time a client browser requests a page with PHP code, the parser looks through the codespec of the page and executes any PHP statements it might find. Fortunately this is a blazingly fast process.

Additionally, JavaScript can still be embedded in the HTML codespec, as usual. This provides a very powerful combination between Web server and client browser, scripting.

Drawbacks In Running PHP As A Server Side Scripts

PHP does consume a lot of Web server resources because everything execute on the Web server. With many concurrent requests and large complex scripts, the server might not be able to respond swiftly enough. Having said that, this is not a matter of grave concern since the parser in PHP is blazingly fast.

HTML codespec with embedded PHP codespec cannot do anything itself, Web server resources must be consumed.

PHP attributes can be summarized as follows:

- PHP stands for Hypertext Preprocessor
- PHP is a Web server-side scripting language
- PHP scripts are executed exclusively on the Web server, their output, which is straight HTML is returned to a client browser
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC and so on)
- PHP is an open source software (OSS)
- PHP is free to download and use

Thus, the benefits of using PHP clearly outweigh its drawbacks. And an increasing number of websites are switching to using PHP.

What Is A PHP File?

A PHP file is an ASCII file, created using any ASCII editor (Notepad is an excellent example) and stored on the HDD. The file contains a mixture of PHP and HTML which instruct the Web server what requires to be done moment to moment.

- PHP files may contain text, HTML tags and scripts
- The output of the processing of PHP files is returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3" or ".phtml"

Why PHP?

- PHP runs on different O/s platforms (Windows, Linux, Unix and so on) seamlessly being an interpreted scripting language
- PHP is compatible with almost all Web servers used today (Apache, IIS and so on)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on any compatible Web server

Extensive help is available via its excellent manual (also freely downloadable). If stuck, the PHP Internet community can be counted on to help. There has been a lot written about PHP on the net and there are countless mailing lists that can be subscribed to.

Extending PHP is pretty easy. PHP has evolved to a full-fledged tool consisting of a few megabytes of source code. It is best to use '**A learn by doing**' approach with PHP. This is not the most scientific and professional approach, but is the method that's the most fun and possibly gives the best end results.

What Is Zend? and What Is PHP?

PHP's language engine is named **Zend** this refers to PHP's central core. The term PHP refers to the complete system as it appears to the outside world. While this might sound a bit confusing, it's not that complicated (Refer diagram 8.1).

To run a PHP script at any Web server, **three components** are required:

1. An interpreter that analyzes PHP code snippets, translates them to something the CPU understands, checks the translation for any kind of error and finally passes it to the CPU for execution
2. A functionality section of the interpreter, which implements the functionality of PHP (**i.e.** binds the Keywords and built in Functions used in PHP code spec to their actual signatures)
3. An interface section that talks to the Web server

Zend handles part 1 completely and a bit of part 2.
PHP handles the rest of part 2 and part 3.

Together they form the complete PHP package. **Zend** itself forms only the language core. Implementing PHP at its very basic with some predefined functions. **PHP** contains all the modules that actually create the language's outstanding capabilities.

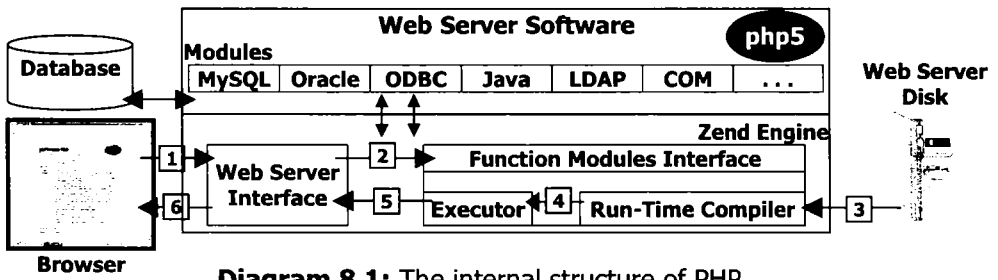


Diagram 8.1: The internal structure of PHP.

Getting Started

What Is Needed?

If a Web server supports PHP nothing extra is needed. Just create .php files in a Virtual Host directory and the Web server will parse and execute them. Most Companies that offer web site hosting also offer PHP support. Generally for **free**.

However, if the Web server does not have built in support for PHP, then PHP must be downloaded, installed and finally bound to the Web server.

Download PHP

Download PHP for free from:

http://www.php.net/downloads.php

After installing PHP, test the installation by executing simple PHP code, to test whether the installation was successful.

The following example, written in PHP, generates the HTML page, which displays a message that the PHP installation has succeeded.

```
<HTML>
<HEAD><TITLE>PHP Successfully Installed</TITLE></HEAD>
<BODY><P>
  <?php
  // This is a comment that extends to the end of the line.
  # This is a comment that extends to the end of the line.
  /* This is multi-line comment. Comments are ignored by PHP, but
  provide valuable information to people reading the source. */
      echo "Congratulation! The PHP Installation Is A Success\n";
  ?>
  </P>
</BODY>
</HTML>
```

Use any ASCII editor to write the above code, then save it to the hard disk as **testinstall.php**.

REMINDER

If Apache refuses to execute **testinstall.php**, then try other extensions like .php3 or .phtml. If the right extension cannot be figured out, then ask whoever installed the Web server for the correct extension to use. Alternatively, have a look at the file **httpd.conf** if Apache is being used as the Web server of choice.

The httpd.conf file should contain a line that looks like this:

AddType application/x-httpd-php.php

This informs Apache to execute **.php** programs and the useable filename extension is **.php**

The Start And End Of A Block Of PHP Statements

In testinstall.php, most of the code is just plain HTML. Only the part between **<?php** and **?>** is PHP codespec. Either use **<?php** (or just **<?**) to mark the start of PHP code. **?>** marks the end of PHP code. The PHP parser executes everthing within these two tags. Put any number of PHP statements, each of which must end with a semicolon (**;**) between the **<?php** and **?>** which marks the PHP code block.

In the example shown (between the two tags) there is one single-line and one multi-line **comment** and one PHP **statement** which is:

```
  echo "Congratulation! The PHP Installation Is A Success\n";
```

The comments will be completely ignored and the statement just prints the string **Congratulation! The PHP Installation Is A Success** to the web page using browser defaults. After the parser has processed **testinstall.php**, the output sent to the browser looks like:

```
<HTML><HEAD>
  <TITLE>PHP Successfully Installed</TITLE>
</HEAD><BODY>
  <P>Congratulation! The PHP Installation Is A Success</P>
</BODY></HTML>
```

There's no trace of PHP code in the final output delivered to the client browser.

REMINDER

In some of the following code snippets all the HTML code may not be included, but remember that HTML has to be used if the PHP codespec has to work in a client browser.

PHP codespec can be visualized as a normal web page written in HTML, since most of the page is still just HTML codespec, but between the PHP **start** and **end** tags (**<?php** or **<?** and **?>**) parsed PHP codespec can spice the HTML page up with (**possibly**) **dynamic content**.

Adding Comments To PHP Code

In PHP, **//** or # is used to mark a single-line comment or **/*** and ***/** to mark a large comment block.

```
<HTML><HEAD><TITLE>Using Comments</TITLE></HEAD><BODY>
  <?php
  // This is a single line comment
  #  This is a single line comment
```

```
  /*
     This is a block of comment statements
  */
  ?>
</BODY></HTML>
```

The First PHP-Enabled Page

Create a file named **testinstall.php** and put it in the Virtual domain created earlier under /var/www/sct/phptraining directory. **testinstall.php** will contain:

```
<HTML><HEAD><TITLE>PHP Successfully Installed</TITLE></HEAD><BODY>
  <?php echo '<P>Congratulation! The PHP Installation Is A Success</P>'; ?>
</BODY></HTML>
```

Using a browser access the file using the Web server's URL in its address bar, but ending with the "**/testinstall.php**" file reference. **eg.** http://sct.phptraining.com/testinstall.php.

Congratulation! The PHP Installation Is A Success

Diagram 8.2: Output for testinstall.php.

If everything is configured correctly, **testinstall.php** will be parsed by PHP and its output will be sent to the browser**: (Refer diagram 8.2)**

```
<HTML><HEAD>
  <TITLE>PHP Successfully Installed</TITLE>
</HEAD><BODY>
  <P>Congratulation! The PHP Installation Is A Success</P>
</BODY></HTML>
```

REMINDER

PHP code is not like CGI scripts. The PHP file does not need to be executable or special in any way. Think of it as a normal HTML file, which happens to have a set of special tags available which do a lot of interesting things.

This program is extremely simple. There is really no need to use PHP to create a page like this. All it does is display the words **Congratulation! The PHP Installation Is A Success** in the browser using the PHP **echo()** statement.

HINT

If this example is tried and nothing was seen in the browser, instead the browser prompted the user to download the PHP file **or** if the whole file is seen as plain text, the chances are that the Web server **does not have PHP enabled**.

Anyways, the point of this example is to show the **special PHP tag** format. In this example **<?php** was used to indicate the start of PHP code spec. Then when PHP codespec was complete PHP mode was exited by the closing tag **?>**. PHP mode can be jumped **in and out** of within HTML code spec exactly like this when required.

Once this PHP codespec works successfully, make a call to a built-in PHP function **phpinfo()**. A lot of useful information about the system, its setup parameters, such as available predefined variables, loaded PHP modules and configuration settings will be displayed as a formatted HTML page. This is the functionality of **phpinfo()**.

Escape Sequences In PHP

Like in most programming languages, PHP uses **** (backslash) as the escape character. A combination of the escape character (i.e. ****) and a letter is called an Escape sequence.

The following is a list of Escape sequences available in PHP:

\"	Print the next character as a double quote, not a string closer
\'	Print the next character as a single quote, not a string closer
\n	Print a new line character
\t	Print a tab character
\r	Print a carriage return (Not used very often)
\$	Print the next character as a dollar, not as part of a variable
****	Print the next character as a backslash, not an escape character

Escape sequences are used to signify that the character after the escape character should be treated specially.

Example:
To store the string "The seller said, "I've sold them for $2!" grinning happily.", escape characters are required because the string consist of a $ (dollar sign), ' (single quote) and a pair of " (double quotes) all placed inside double quotes.

The following code spec, stored in **testescape.php**, illustrates the same:

```
<?php
  $double = "A pair of \"double quotes\" are displayed.";
  $single = "A pair of \'single quotes\' are displayed.";
  $dollor = "A \$ indicates a dollar.";
  $new =  "A new line appears as \n.";
  $backslash "A double backslash allows \\ to be displayed.";
?>
```

The escape characters are just to make sure PHP can read the string correctly - once the data has been read, it is converted into the original format.

WARNING

Escape sequences only work in double-quoted strings. If singles quotes are used as in 'Hello!\n\n\n', PHP will actually print out the characters \n\n\n rather than converting them to new lines.

REMINDER

When stored in a variable or a file, Escape characters are stored as one character (e.g. **RETURN key**), however PHP understands them as two characters (e.g. **\n**) as otherwise it is difficult to type the ASCII characters using a normal keyboard.

Example:
\n will become ASCII character representing the **RETURN key**

Displaying Large Text Information In PHP

Sometimes it is essential to display large amounts of text from within PHP. In such situations, the text may contain several instances of single or double quotes, causing repetitive use of escape sequences. This adds to the programming nightmare.

The **heredoc** syntax has been introduced to tackle the above problem. It allows the programmers to define their own string delimiter, which can be something other than a double or single quote. A normal string, used as a delimiter, will allow the free use of single or double quotes within the body of the text. The string only ends when a user-defined delimiter is encountered.

WARNING

While using the **heredoc** syntax, the string delimiter needs to be by itself on a line, in the very first column. Spacing or tabs cannot be placed around the string used as the delimiter.

The following example defines the word END as a delimiter in the heredoc syntax:

```
echo <<<END
    <A HRef="http://sct.phptraining.com"
                Target="anywhere">sct.phptraining</A>
    Any amount of textual information can be placed here.
    It will not terminate until "END;" is encountered.
END;
```

The following points should be bared in mind while working with **heredoc:**

- The string used as a delimiter is of the users choice
- The symbol **<<<** is required before the delimiter to indicate that the heredoc mode is entered

- Variable substitution is used in PHP, which means that there is a need to escape dollar symbols - if not done, PHP will attempt variable replacement
- The delimiter can be used anywhere in the text, but not in the first column of a new line
- At the end of the string, just type the delimiter with no spaces around it, followed by a semi-colon to end the statement

Not only is it intuitive (does what a user thinks it should), jumping out of PHP, rather than using an echo, **is apparently faster**.

Alternative Method

Even jumping from PHP to HTML and back **within a function definition** is possible. Consider the following code stored in the file **testfunction.php:**

```
<?    function func() {
        echo "Entering function..."; ?>
Some HTML inside function.
<?    echo " Leaving function."; }   ?>
<HTML><BODY>
  <? func(); ?>
</BODY></HTML>
```

Output: (Refer diagram 8.3)

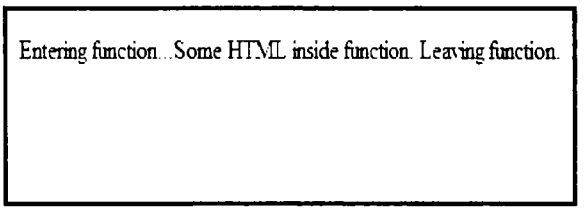


Diagram 8.3: Output for testfunction.php.

Whitespace

In PHP, characters like spaces, tabs and new lines in between statements have no effect on how the code is executed.

Example:
This next script is the same as the previous script, even though it is laid out quite different:

```
<?
  function func() {
       echo "Entering function...";
?>

Some HTML inside function.

<?
  echo " Leaving function.";
  }
?>
<HTML><BODY>
  <? func(); ?>
</BODY></HTML>
```

The use of whitespaces is recommended to separate the code into clear blocks. This segregation helps a programmer understand the code by simple visual inspection.

Including Other Files

A powerful aspect of PHP is its ability to build templates and code libraries that can be easily inserted into new scripts. In the long run, using code libraries can drastically minimize time and errors when common functionality needs to be reused across Web sites. Those with backgrounds in other languages (such as C, C++ or Java) will be familiar with this concept of code libraries and including them in a program to extend the programs functionality.

Basic templates can be easily built by including one or a series of files using two of PHP's predefined functions, **require()** and **include()**. Each function has a specific application. The source of reusable code snippets could either be internal (functions contained within the same file) or external (functions contained within other files). Sharing functionality is one of the basic operations in PHP.

The include Keyword

The **include** keyword allows embedding scripts written in some other file in the current PHP file. The keyword is followed by the path and filename (specified as a string placed within a pair of single quotes or double quotes) to locate the file containing the script, which must be included.

Syntax:

```
include "<Path To File>";
```

Example:
The following code is stored in the **testinclude.php** file:

```
<?   print 'Processing started.<BR>';
  include "fileincluded.php";
  print 'Processing completed.<BR>';   ?>
```

The following code is what is in **fileincluded.php:**

```
<?php   print 'Contents of the Included File.<BR>'; ?>
```

While executing the **testinclude.php** file, PHP executes the **print** statement and then proceeds towards the file inclusion. If the file specified in the include statement exists, the file is loaded and its contents are embedded into the **testinclude.php** file. The **testinclude.php** file (after the file inclusion) would appear as shown below:

```
<?   print 'Processing started.\n';   ?>
<?php   print 'Contents of the Included File\n';     ?>
<?   print 'Processing completed.\n';   ?>
```

Output: (Refer diagram 8.4)

After embedding the contents of the file included, PHP continues processing by performing the operations as stated by the newly embedded code spec.

```
Processing started.
Contents of the Included File.
Processing completed.
```

Diagram 8.4: Output for testinclude.php.

REMINDER

Whenever PHP includes another file, it drops out of PHP mode, then re-enters PHP mode as soon as it returns from the file being included.

PHP includes a file only if the **include** statement is executed. If the **include** statement is preceded by a conditional statement, the file specified for inclusion will be embedded only when the condition is satisfied.

Example:
The following code would never include **cndtinclude.php:**

```
<?php
  print "Conditional Process
        Started.<BR>";
  if (1 == 2) {   include "fileincluded";
  print "Conditional Process
        Completed.<BR>";
?>
```

```
Conditional Process Started.
Conditional Process Completed.
```

Diagram 8.5: Output for cndtinclude.php.

Output: (Refer diagram 8.5)

WARNING

In situation when PHP cannot locate the included file, a warning message is generated and displayed on the client's browser. The execution of the remaining lines of code is continued after the warning.

If the file specified for inclusion is stored at a different location, then the **relative path** to the file must be specified in the **include** statement. For example, if the file for inclusion was stored one level higher in the directory's hierarchical tree, then the **include** statement would read as:
include "../fileincluded.php";

The require Keyword

PHP also has the **require** keyword, which functions on the lines of the **include** keyword. The only notable difference being that if called on a file that does not exist, it will **halt script execution** with a **fatal error**.

HINT

The use of the **require** keyword should be limited to situations where the included file is mandatory for the program to execute successfully. If program process flow is not severely dependent on the file marked for inclusion, use the **include** keyword instead.

Example:
The following blocks of code describe the difference in functionality between the **include** and the **require** keywords. *In order to get an error or a warning (as the case may be) the file embedding is invalid.*

The code stored in the **cmprincluded.php:**

```
<?    print 'Processing started.<BR>';
   include "../fileincluded.php";
   print 'Processing completed.<BR>';
?>
```

Output:

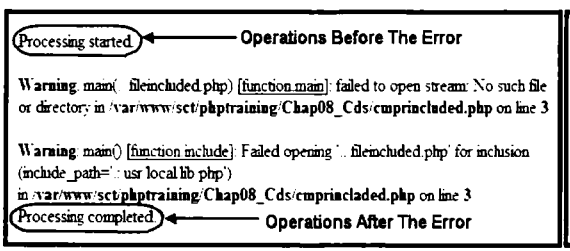


Diagram 8.6.1: Output for cmprincluded.php.

The code stored in the **cmprrequired.php:**

```
<?    print 'Processing started.<BR>';
   require "../fileincluded.php";
   print 'Processing completed.<BR>';
?>
```

Output:

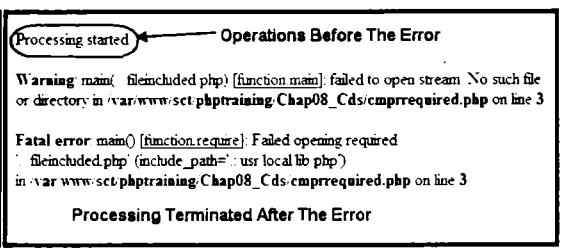


Diagram 8.6.2: Output for cmprrequired.php.

As seen in the earlier examples, the PHP execution continues even after the warning issued by the **include** statement. While in case of the **require** statement the error causes program execution to terminate.

WARNING

In older versions of PHP, **require** keyword was the equivalent of an **unconditional include**. If **require** was placed inside a conditional statement, the file would be included even if the conditional statement evaluated to false. This is no longer valid in PHP 5.

HINT

Included files are commonly used as storage for common functions, object definitions and layout code. For instance, if pages on a site have the same header HTML, a repetition in coding can be reduced by creating a header page and then including it in other pages.

Other Keywords

Sometimes there is a repetition in including files, resulting redundant processing. The keywords **include_once** and **require_once** are used to prevent such repeated file inclusion. These keywords will only include a file once, even if attempts are made to include it several times.

WARNING

The keywords **include_once** and **require_once** share the same list of "already included" files, but it is important to note that operating systems that are case sensitive are able to include a file more than once if the programmer uses varying cases while specifying the filename.

PHP needs to compile the file each time **include()** or **require()** is used. If code cache is used (by editing the php.ini file) this problem is avoided. If not, PHP has to compile the same file several times.

HINT

To prevent a case sensitive operating system for repeating included file, use lower case filenames.

The functions **get_included_files()** or **get_required_file()** can be used to retrieve the file names which have already been included.

REMINDER

If PHP cannot find a file specified using **include()** or **require()** in the same directory as the script that is running, it will also look in its standard include path. This is defined in the system's **php.ini** file as the **include_path** directive.

SECTION III: WORKING WITH PHP

PHP Basics

All programming environments provide syntax, used to declare and initialize variables, which are used to store user values, for later use. PHP permits the creation and use of different variables.

For example, the cost price of a product may have to be referenced often by program code, it would be nice to define **cp** (i.e. cost price) as a variable once, load it with a value and just refer to its contents, whenever required. When the products cost price changes, simply update the value held in **cp** created earlier. Subsequently, all other calculations that refer to **cp** will automatically provide the correct output.

Data Types in PHP

PHP has seven types of variables. All hold a specific class of information, **except one**.
The seven variable types are:

- **String** – String variables hold characters. Usually a set of characters such as "S", "Vaishali", "PHP is my favorite language" and so on. Strings can be as short or as long as desired. *There is **no limit** to size*
- **Integer** – Integer variables hold whole numbers, either positive or negative such as 1, -50, 22937932 and so on. *There is a **maximum limit** to the size of integers.* Any numbers lower than -2147483647 and higher than 2147483647 are automatically converted to floats as this datatype can hold a much larger range of values
- **Float** – Float variables hold fractional numbers as well as very high integer numbers such as 3.5, 1.00000001, 2147483647000 and so on
- **Boolean** – Boolean variables hold **TRUE** or **FALSE**. Behind the scenes these are just integers. PHP considers the number **0** to be **FALSE** and **everything else** to be **TRUE**
- **Array** – Array variables are a special variable type, which hold multiple values. They are a great way of storing variables that naturally collect together such as colors, days of the week, members of a team or items in a shopping basket
- **Object** – Object variables are complex variables that have multiple values. They also have their own functions (i.e. methods) for accessing and/or manipulating their content.

- **Resource** – Resource variables are variables that hold anything that is **not** PHP data. This could be picture data loaded from a file, the result of an SQL query and so on. A Resource type variable is used like any other variable with the key difference being that they should be freed up when not required

Naming And Loading Variables In PHP

PHP variable is declared with the help of **$** character followed by a letter. The rest of the letters can consist of both alphabets and numbers. The following are three examples of valid names for variables**:** $city, $address2 and $age_30. The _ (underscore) character can also be used in variable names. It is used as a **replacement for space** as space is a **character** that **cannot** be used in naming a PHP variable.

Once a variable is named, it is empty until assigned a value. To assign a value to variable**:**

```
$city = 'Mumbai';
```

The **named variable** is on the **left** side of **=** (**i.e.** an assignment operator) and the **value** held by the variable is on its **right**. Here, $age is assigned a value of 24**:**

```
$age = 24;
```

Different types of values can be assigned to variables. Integer and String types have already been demonstrated. No special instructions need to be passed at the time of declaring a variable. This is because PHP does not have a concept of a variable being bound to a specific data type at the time of its declaration. Irrespective of the value a variable holds in the future, PHP takes care of integer and string conversions on the fly. Hence, the same variable can hold different types of user values, at different instances in time.

Variable scope

Each variable has an area in which it exists, known as its scope. It is technically possible for a PHP script to have several variables called **$a** in existence at one point in time, however there's only one active $a at any one time.

All variables set outside a function or an object are considered **global**. This means, they are accessible from anywhere else in the PHP script.

Numbers

It's really simple to deal with numbers in PHP. Just use them as required. All the normal rules about number precedence apply.

```
$a = 4;
$b = 7;
$c = 2 + 3 * $a + 5 * $b;  // This is evaluated as 2+(3*4)+(5*7) = 49
```

When a variable is referenced by its name, PHP knows that it's the value held in the variable that must be processed and not the variable itself. This is **automatically** done **before** a new value is stored in the variable on the **left** hand side of the assignment operator. Hence, something like this can be done:

```
$a = 5;
$b = 10;
$a = (2 * $a + 5)/$b;       // Evaluated as $a = (2*5+5)/10 = 1.5
```

Strings

When assigning a String value to a variable it **must be** enclosed in quotes. Either single quotes (') or double quotes (") can be used. There is a difference between the two types of quotes. The following code demonstrates this difference:

```
$first_name = 'Vaishali';
$greeting1  = "Hello, my first name is $first_name.";
echo $greeting1;
$last_name  = 'Shah';
$greeting2  = 'Hello, my last name is $last_name.';
echo $greeting2;
```

This code produces the following output:

```
Hello, my first name is Vaishali.
Hello, my last name is $last_name.
```

When **double quotes** are used, PHP performs **variable expansion**, this means that PHP substitutes the value of **$first_name** wherever a reference to $first_name is encountered. The result is that the string value stored in the variable **$greeting1** is **Hello, my first name is Vaishali.**

When a value is assigned to the variable $greeting2 and **single quotes** are used then PHP does not perform any variable expansion and hence $greeting2 ends up with **Hello, my last name is $last_name**, This means that anything enclosed in **single quotes** is treated as a **string constant** and is not to be changed in anyway by PHP when processed.

Double quotes are also used for expanding other **special characters**, such as the **newline** character (**\n**).

The following string:

```
echo "Vaishali\nShah";
```

Will look like this when rendered in a browser:

```
Vaishali
Shah
```

The **echo** statement simply dispatches the string after it to the Web browser. There's no trace of the PHP code, but there is a **newline character** placed between the two words.

The **newline** character can be seen only in the source code of the PHP based page because the Web browser treats the newline character as a normal space. **Notice** that there's **no space before** or **after** the **\n** character.

This is because the (****) **backslash** is used to indicate that the **next character is special.** In PHP code spec if a single backslash itself is required, two backslashes must be written (****). The second backslash will then be treated as a string literal and seize to have any special meaning. The backslash (****) is called an **escape character**.

Similarly to display a **$** character within a string enclosed in double quotes, it will have to be preceded by a backslash. This is because PHP looks for a variable, since all variables start with a $.

This string will give a little problem:

```
$name = "Sharanam is an "Analyst Designer" residing in Mumbai";
```

If tried out, an error pops up. That's because the parser expects the string to stop after the second double quote, just before the word **Analyst Designer**. The rest of the line is ignored by the PHP parser. The following string is correct:

```
$name = "Sharanam is an \"Analyst Designer\" residing in Mumbai";
echo $name would produce: Sharanam is an "Analyst Designer" residing in Mumbai
```

The same applies if single quotes were used throughout the string.

Another way to resolve this issue is to enclose the string within **single quotes** and then use as many double quotes within the string as required.

Or

Do the inverse, enclose the string in double quotes and use as many single quotes in the string as required. This solves problem when it's necessary to output HTML code, which requires a lot of double quotes within the structure of the string:

```
echo '<A HRef="mailto:sharanams@ivanbayross.com">
    <IMG Src="image.gif" Width="16" Height="16" Alt="me" Border="0"></A>';
```

If the string following echo, was enclosed within double quotes, all the other double quotes will require escape characters before them. On the other hand, in the above string PHP would not do any variable substitution, i.e. PHP is being forced to deal with the string as though it was a string literal.

It's only when expressions are evaluated (**i.e.** such as in assignments and echo statements) that the difference between single and double quotes come into the picture. After the assignment is done, no one can tell how the string was produced.

Consider this code:

```
$name = 'Vaishali Shah is a ';
$post = "Programmer";
$var1 = "$name $post";
$var2 = 'Vaishali Shah is a Programmer';
```

Variables $var1 and $var2 will both contain the string Vaishaii Shah is a Programmer after evaluation, PHP will not make any distinction between the values stored in the two variables.

Joining Strings In PHP

To concatenate (add together) strings use the (.) period character (**i.e.** a dot or full stop).

```
$name = 'Vaishali Shah is a';
$post = 'Programmer';
$data = $name.$post;
```

Now $data will contain the string "Vaishali Shah is aProgrammer". That's probably not what is wanted. It would be nice with a space between the words **a** and **Programmer**. To do this, execute the following code:

```
$data = $name . ' ' . $post;
```

Notice that the **space** between the variable **$name** and the dot is ignored, it's the **string with the space (within single quotes) that's important**.

This could be solved using variable substitution. It is often easier to read, just remember to use double-quotes around the string. This PHP code gives the same results as concatenating two variables using a period:

```
$data = "$name $post";
```

When using variable substitution a tiny problem will be encountered. The problem occurs with substitution of a variable in a string, which requires having text immediately after the variable. The variable name has to be **delimited** somehow.

Using curly braces is the solution as shown:

```
$noun = 'cat';
echo "A $noun, two ${noun}s";
```

The above code shows playing around with English words in plural form. The first occurrence of $noun is substituted correctly, even when it's followed immediately by a comma because a comma cannot be a part of a name. But certainly there can be a variable called $nouns, so curly braces can be used to indicate that the variable called **$noun** is being referenced and not some nonexistent variable called $nouns.

Variable Variables

Variable variables allow accessing the contents of a variable without knowing its name directly. This is like indirectly referring a variable. Consider the following code spec:

```
<?php
   $linux = 3.0;
   $OS = "linux";
?>
```

There are two ways to retrieve the value of $linux.

- Use **print $linux**, which is quite straightforward

Or

- Use **print $$OS** (Note the use of **two dollar** signs when using the variable variables technique)

By using $$OS, PHP will look up the contents of $OS, convert it to a string, then look up the variable of the same name and return its value.

In the above example, $OS contains the string "linux" and so PHP will look up the variable named $linux and output its value in this case 3.0.

Example:
(OperatingServer.php)

```
<?php
   $AdvanceServer = "3.0\n";
   $RedHat = "AdvanceServer";
   $Linux = "RedHat";
   $OS = "Linux";
   print $AdvanceServer;
   print $$RedHat;
   print $$$Linux;
   print $$$$OS;
?>
```

Output: (Refer diagram 9.1)

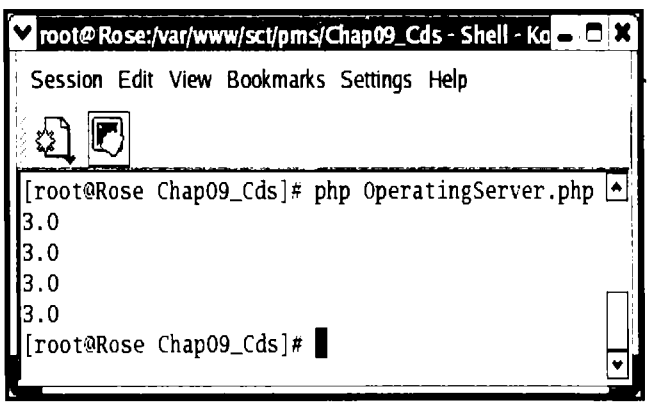


Diagram 9.1: Output for OperatingServer.php

References

When the = (equals) operator is used, PHP performs a **copy operation.** It takes the value from operand two and copies it into operand one. While this is fine for most purposes, it doesn't work when it is desired to change operand two later on and have operand one change as well.

In such a scenario, references are helpful as they allow having two variables pointing to the same data. Using references, setting operand one equal to operand two is instantaneous whether or not operand two is a simple integer or a 10MB array of data.

Furthermore, once two variables are pointing to the same data, either of the variables can be changed and the other one will also be updated.

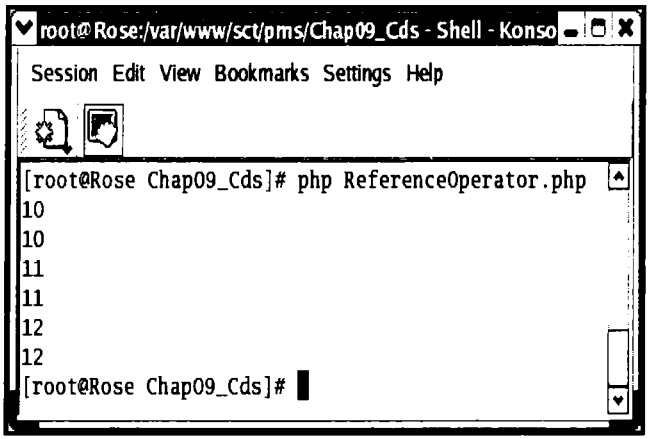


Diagram 9.2: Output for ReferenceOperator.php

To assign by reference use the reference operator (&) after the equals operator as **=&**. Refer diagram 9.2.

Example: (ReferenceOperator.php)

```
<?php
   $a = 10; $b =& $a;
   print $a."\n";
   print $b."\n";
   ++$a;
   print $a."\n";
   print $b."\n";
   ++$b;
   print $a."\n";
   print $b."\n";
?>
```

Here the reference operator is used to make $b point to the same value as $a, as can be seen in the first two print statements. After incrementing $a, both variables are printed out again and both are 11, as expected. Finally, to prove that the relationship is two-way, $b is incremented and again both $a and $b have been updated.

Reference assigning is actually slightly slower than copy assigning if large variables are not used, so it is best to stick with normal copy assigning unless there is a good reason to switch.

When it comes to functions, references can also be used to allow a function to work directly on a variable rather than on a copy.

Operators And Expressions

This section lists different operators used in PHP. These are used to operate on values held in variables or constants.

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Assignment Operators

Operator	Description	Example	Result
+	Addition	x=3 x+3	 6
-	Subtraction	x=3 5-x	 2
*	Multiplication	x=5 x*5	 25
/	Division	20/2 7/2	10 3.5
%	Modulus (division remainder)	5%3 7%2 20%2	2 1 0
++	Increment	x=8 x++	 9
--	Decrement	x=8 x--	 7

Comparison Operators

A number of different comparison operators are available to compare the value held in one variable against the value, which is held in another:

Operator	Meaning	Example
==	Is equal to	3==9 returns false
!=	Is not equal to	3!=9 returns true
<	Is less than	3<9 returns true
<=	Is less than or equal to	3<=9 returns true
>	Is greater than	15>18 returns false
>=	Is greater than or equal to	15>=18 returns false

Logical Operators

Operator	Description	Example
&&, and	The first **and** the second is true	(15!=18) && (15==18) returns false
¦¦, or, ¦¦	The first **or** the second or both is true	(25!=28) ¦¦ (25==28) returns true
!	Not	!(25==28) returns true

Example: (Operator_Exp.php)

```
<?php
  print $myvar = 5 + 5 . "\n";
  print $myvar = 5 - 5 . "\n";
  print $myvar = 5 + 5 - (5 + 5) . "\n";
  print $myvar = 5 * 5 . "\n";
  print $myvar = 10 * 5 - 5 . "\n";
  print ($myvar = 5) . "\n";
  print $myvar++ . "\n";
  print $myvar-- . "\n";
  print $myvar . "\n";
  print ++$myvar . "\n";
  print $myvar = $myvar . "appended to end.\n";
?>
```

Output: Refer diagram 9.3

The above example shows how to assign values to variables.

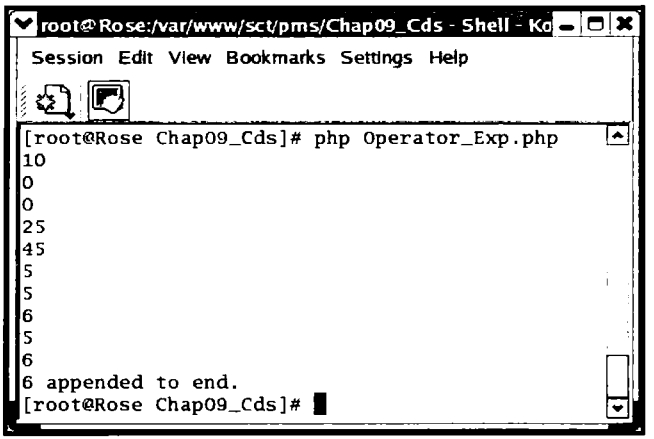


Diagram 9.3: Output for Operator_Exp.php

Arrays

Arrays are a single variable that can hold a number of different values at the same time. To store/access different values use square brackets and their numeric position in the array after the array name as follows:

```
$name[0] = 'Sharanam';
$name[1] = 'Shah';
```

Now an array has been created and its first two slots have been filled. Notice that the numbering starts at **zero** and **not** at **one**. Recall the values held as follows:

```
echo $name[0] . ' ' . $name[1];
```

If it is not important as to which slots data is assigned in the array, then use a pair of empty brackets after its name when loading it with data as follows:

```
$name[] = 'Sharanam';
$name[] = 'Chaitanya';
$name[] = 'Shah';
```

This filled the first three slots of the array, which are numbered 0, 1 and 2. When this has to be retrieved specify which slot to access.

Array slots can also be identified by **name**. This makes the code spec more readable and easy to understand as follows:

```
$personal_data['first name'] = 'Sharanam';
$personal_data['post'] = 'Analyst Designer';
$personal_data['last name'] = 'Shah';
$personal_data['age'] = 25;
echo 'Hello ' . $personal_data['first name'] .
    '! We know that you are ' . $personal_data['age'] .
    ' years old.';
```

The PHP parser will ignore all whitespace and the result is:

```
Hello Sharanam! We know that you are 25 years old.
```

This is often the easiest way to deal with associative arrays. Another possibility is to use variable substitution with curly braces like this:

```
echo "Hello {$personal_data['first name']}! " .
    "We know that you are {$personal_data['age']} years old.";
```

This code gives the same result. More complex expressions can be used inside such a set of curly braces. This includes accessing elements in multidimensional arrays and object properties.

It is recommended to use descriptive names for the slots in the arrays. It makes the code spec clear and helps to indicate what the different slots are intended for.

SECTION III: WORKING WITH PHP

PHP Conditional Statements And Iterations

Conventional Conditional Statements

The linear, hierarchic, top down, execution of a program, is controlled by 'Control Structures' embedded within program code. PHP code often has to do several things, this means that PHP code has to have decision making abilities, based on the decision made the PHP code will do some work **or** otherwise do something else entirely. PHP provides programmers with Conditional statements to achieve this.

There are two types of conditional statements:

- **if (...else) statement** – This statement is used, if a block of code must test a specific condition, if the condition tested returns a 'TRUE' then the code executes. If the condition tested returns 'FALSE' another block of code executes
- **switch statement** – This statement is used, to choose and execute **one** code block from among multiple code blocks for execution

The if Statement

Use the if-statement to execute different code snippets of PHP code in a program, based on what is returned (**i.e.** either **TRUE** or **FALSE**) when a condition is evaluated.

Syntax:

```
if (<Condition>)
    <Code Block To Be Executed If Condition Returns TRUE>;
else
    <Code Block To Be Executed If Condition Returns FALSE>;
```

The following example, stored in **iftest.php**, will give the output **The first number is greater than the second number** if the number1 i.e. 50 is greater than number 2 i.e. 25, otherwise it will give the output Number 2 is greater than Number 1:

```
<HTML><HEAD><TITLE>Testing If Statement </TITLE></HEAD><BODY>
<?php
  $number1=50;
  $number2=25;
```

```
  if ($number1 > $number2) {
      echo "The first number is greater than the second number";    }
  else {
      echo "The second number is greater than the first number 1"; }
?>
</BODY></HTML>
```

Output: (Refer diagram 10.1)

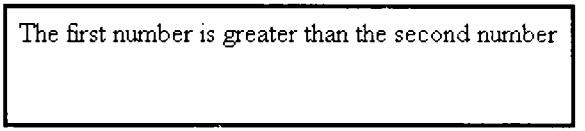


Diagram 10.1: Output for **iftest.php**.

In the above example, there are two variables i.e. number 1 and number 2 holding 50 and 25 respectively. If the number1 holding 50 is greater than number 2 holding 25 then appropriate message is displayed.

Executing Multiple Statements

Example:

When multiple lines of code require execution, when a condition is TRUE, these lines should be enclosed within **curly** brackets. Consider the following code, stored in **multiline.php:**

```
<HTML><HEAD>
  <TITLE>Executing Multiple Lines Via
        If statement</TITLE>
</HEAD><BODY>
<?php
  $weekDay = date("D");
  if ($ctr == "SUN") {
  // Start of code block
      echo "Hello<BR>";
      echo "The day is a Sunday";
  }  // End of code block
  else {
  // Start of code block
      echo "Hello<BR>";
      echo "The day is not a Sunday";
  }  // End of code block
?>
</BODY></HTML>
```

Diagram 10.2.1: Multi-line output when the condition is satisfied.

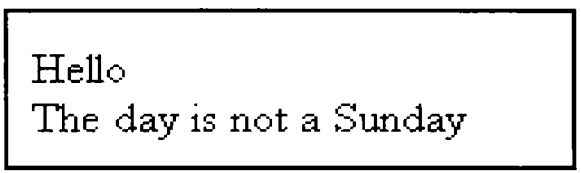


Diagram 10.2.2: Multi-line output when the condition is not satisfied.

Output: (Refer diagram 10.2.1 and diagram 10.2.2)

In the above example, the current weekday is assigned to the variable **$weekday**, then a check is made as to whether variable **$weekday** holds value **SUN** (**i.e.** whether today is a Sunday). When the day is a Sunday, the **if** statement returns **TRUE**. This allows the code immediately following the **if** statement to be executed.

On any other day, the if statement returns **FALSE**. This allows the code immediately following the **else** statement to be executed.

However, as multiple statements have to be executed, the **if / else** statements are followed by a opening curly braces **{** (indicating the start of the code block). The statements that should be executed are place after the opening curly braces. Finally, the closing curly braces **}** (indicating the end of the code block) is placed.

The pair of curly braces **{}** after the condition, contains the code block that should be executed when condition returns **TRUE**. Similarly, the pair of curly braces **{}** after the **else** clause, contains the code block that should be executed when the condition returns **FALSE**.

All kinds of statements can be used between the parenthesis and the curly braces. PHP is a language where almost every statement has a value. A value can be used together with other values to build complex expressions.

Take an assignment for example. The statement $b = $a, indicates that $b now holds the value held by $a. This means that this value can be used in another assignment, like this:

```
$c = ($b = $a);
```

The assignment in the parenthesis has the value of $a, which is then assigned to $c. The parenthesis can actually be dropped and the result will be the same. All three variables have the value of $a. This explains the following statement:

```
/* Initialize counters: */
$i = $j = $k = $l = 0;
```

Using comparison operators any kind of Boolean expression can be constructed. Take for example the following **if** statement:

```
if ($a <= $b && ($array[$a] <= $array[$b]
    || $array[$a+$b] > $array[$a-$b])) {
/* Do something useful - this is a comment, it won't be
executed */
}
```

This **if** statement, first tests to see if the value held by variable $a is smaller or equal to the value held by variable $b. If TRUE, the first part of the second section (i.e. after the &&) is executed to see if it is also TRUE. This happens because the **&&** is being used, which means that the first section and the second section should both be TRUE.

The first part of the second section (i.e. after the **&&**) uses $a and $b as indices in an array. If the first part of the second section (i.e. after the **&&**) is TRUE, then the second part of the second section, (i.e. after the **||**) will not be executed. This happens because the **||** is being used, which means either of the parts in the second section need to be true.

The above said is similar to the following condition:

```
oci_connect() or die('Could not connect to the Oracle database!');
```

The function **oci_connect()** tries to make a connection to a Oracle database. It returns TRUE if it was able to make a connection. PHP evaluates the line as a Boolean expression. That means that it starts by executing **oci_connect()**. If that returns TRUE then PHP skips the next statement because the **or** (**| |**) operator is used. The second statement is executed which halts the processing with a user defined error message, if the first statement fails.

The elseif Clause

Consider the following code:

```
<?php
  $d=date("D");
  if ($d=="Sun") {
      print "Today is a Sunday."; }
  else { if ($d=="Mon") {
          print "Today is a Monday."; }
      else { if ($d=="Tue") {
              print "Today is a Tuesday.";     }
          else { if ($d=="Wed") {
                  print "Today is a Wednesday.";     }
              else { if ($d=="Thu") {
                      print "Today is a Thursday.";   }
                  else { if ($d=="Fri") {
                          print "Today is a Friday."; }
                      else { print "Today is a Saturday.";    }
  }  }  }  }  }
?>
```

The objective of the above code is to identify the weekday of the current date of the machine serving PHP pages. The technique used is a very crude manner of working with the nested **if** statements.

The code block starts by extracting the weekday and storing it into the variable **$d**. A check is made if the extracted weekday is Sunday. If the condition returns TRUE, a message indicates that the day is Sunday. If the condition returns FALSE, the code block specified under the **else** clause is executed.

The code block under the **else** clause consist of a nested **if** statement. This **if** statement checks whether the extracted weekday is Monday. Depending on the value returned by the condition, an appropriate code block is executed (i.e. either a message indicating the day or a **else** clause). The code block for the inner **else** clause is another set of if statements. Thus the level of nested if statements increase to six to check for all possible values for the variable **$d**.

The end result is a code, which is difficult to manage, basically because PHP insists that every code block beginning with an opening curly braces be terminated with a closing curly braces.

Another method for achieving the same functionality is the use of the **elseif** clause. The **elseif** clause allows working with nested if statements in a slightly more intelligent way.

Syntax:

```
if (<Condition>)
    <Code Block To Be Executed If Condition Returns TRUE>;
elseif (<Condition>)
    <Code Block To Be Executed If Second Condition Returns TRUE>;
else
    <Code Block To Be Executed If BOTH Conditions Returns FALSE>;
```

The following code, stored in **nestedif.php**, would replace the above example:

```
<?php
  $d=date("D");
  if ($d=="Sun") {
      print "Today is a Sunday."; }
  elseif ($d=="Mon") {
          print "Today is a Monday."; }
  elseif ($d=="Tue") {
      print "Today is a Tuesday.";    }
  elseif ($d=="Wed") {
      print "Today is a Wednesday."; }
  elseif ($d=="Thu") {
      print "Today is a Thursday.";
  }
  elseif ($d=="Fri") {
      print "Today is a Friday."; }
  else { print "Today is a
Saturday."; }
?>
```

Output: (Refer diagram 10.3)

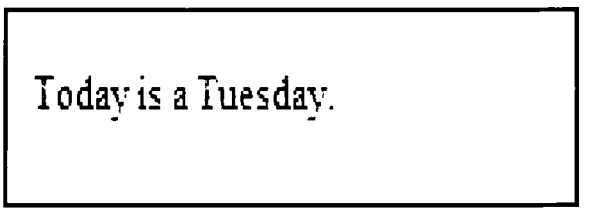


Diagram 10.3: Output for the code block with nested **if** statements.

REMINDER

The braces could be avoided if only one line of code has to be executed for a condition. Even though there is only the tiniest fraction of a speed difference between using braces and not using braces, it's usually down to a readability issue.

The same effect can be achieved by having lots of **if** statements as shown below:

```
<?php
  $d=date("D");
  if ($d=="Sun") {
      print "Today is a Sunday."; }
  if ($d=="Mon") {
      print "Today is a Monday."; }
  if ($d=="Tue") {
      print "Today is a Tuesday.";    }
  if ($d=="Wed") {
      print "Today is a Wednesday."; }
  if ($d=="Thu") {
      print "Today is a Thursday.";   }
  if ($d=="Fri") {
      print "Today is a Friday.";  }
  if ($d=="Sat") {
      print "Today is a Saturday.";   }
?>
```

However, the use of the **elseif** clause is a slightly neater way of working with nested **if** statements. The downside of this system is that the $d variable needs to be checked repeatedly.

The switch Statement

When the same variable is checked for different possible values, it is advisable to use **switch** statement, (i.e. if only one of many code blocks has to be selected for execution).

Syntax:

```
  switch (<Expression>) {
  case <Label1>:
      <Code To Be Executed If Expression = Label1>;
      break;
  case <Label2>:
      <Code To Be Executed If Expression = Label2>;
      break;
  default:
      <Code To Be Executed If Expression Is Different From Both Label1
      And Label2>;
  }
```

In a **switch** statement, a single **expression** (most often a variable) is evaluated once. The value of the expression is then compared with the values held in the **labels** of each **case** in the structure. If there is a match, the block of code associated with that case is executed.

Use **break** to **prevent** the program execution from automatically running into the next **case** after the selected **case** completes its execution.

The **default** clause is used if none of the cases return **TRUE**.

Example:
The following code (stored in **testswitch.php**,) obtains the functionality of the nested **if** statements, by using the **switch** statement:

```
<HTML>
  <HEAD><TITLE> Using Switch Statement</TITLE></HEAD>
  <BODY>
<?php
  $d=date("D");
  switch ($d) {
  case "Sun":
      print "Today is a Sunday.";
      break;
  case "Mon":
      print "Today is a Monday.";
      break;
  case "Tue":
      print "Today is a Tuesday.";
      break;
  case "Wed":
      print "Today is a Wednesday.";
      break;
  case "Thu":
      print "Today is a Thursday.";
      break;
  case "Fri":
      print "Today is a Friday.";
      break;
  case "Sat":
      print "Today is a Saturday.";
      break;
  default:
      print "None of the days are valid.";
  }
?>
  </BODY>
</HTML>
```

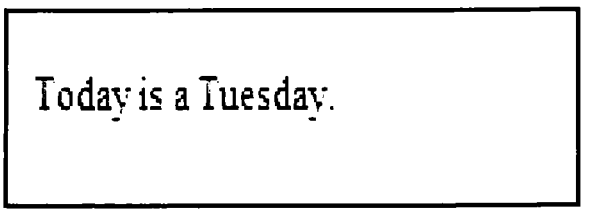


Diagram 10.4: Output for the code block with the **switch** statement.

Output: (Refer diagram 10.4)

The code block starts by extracting the weekday and storing it into the variable **$d**. A check is made to identify the value stored in **$d**. When the first case clause is encountered and the value stored in the variable matches the value following the case keyword (in this case 'Sun'), the block of code immediately after the first **case** clause is executed.

If there is no match, the control is directly taken to the next **case** clause. The code block placed between the failed **case** clause and the next are ignored. If all case clauses fail, the code block under the default clause gets executed.

Sometimes, it is possible that the argument in a **switch** statement satisfy two of more conditions. In such situations, the **break** keyword is use to restrict PHP from execution of all **case** clauses having their conditions satisfied. When the control comes across the **break** keyword, the remaining code block held within the condition gets ignored and the control is placed on the next statement.

Iterations In PHP

Iteration (a Loop) in PHP is used to execute the same block of code a pre-specified number of times.

Looping

Often when developing an application, to execute the same block of code a specific number of times a Loop statement is used in the code spec.

In PHP the following Loop statements are available:

- **for** – loops through a block of code a specified number of times
- **while** – loops through a block of code as long as a specified condition is TRUE
- **do...while** – loops through a block of code **at least once** and then repeats the loop as long as a specified condition is TRUE
- **foreach** – loops through the elements of an array. Works only on arrays. Will throw an error when used with a variable of a different data type **or** an un-initialized variable

The for-loop

When the exact number of times a block of code must execute is known, a **for** loop can be used.

Syntax:

```
for (<Initialization>; <Condition>; <Increment>) {
    <Code To Be Executed>; }
```

REMINDER

The **for** statement takes three parameters. The first is for initializing its variables, the second holds the condition to be checked and the third contains the increment required to implement the loop. If more than one variable is included in either the initialization or the increment section, then they should be separated by a comma. The condition must evaluate either to **TRUE** or **FALSE** only.

The for-loop works by first initializing a **control variable**, then performs a conditional test, if the test returns TRUE, the code block is executed. After the code executes the variable initialized, is incremented and then the main code is executed again. This goes on until the **control variable** holds a value equal to the number of times the loop must run, then the loop exits.

Example: Contents of testfor.php

```
<HTML><HEAD><TITLE>Testing The for Iteration</TITLE>
</HEAD><BODY>
<?   for ($ctr = 0; $ctr < 5; $ctr++) {
  echo "The for loop has been executed $ctr times<BR>";   }
?>
</BODY></HTML>
```

Output: (Refer diagram 10.5.1)

The **for** loop first assigns a starting-value **0** to **$i**, then it checks to see if **$i** is smaller than **5** and if it is, the echo-statement is executed. At the end of an iteration (i.e. one pass of the loop) the **$i++** statement is executed.

```
The for loop has been executed 0 times
The for loop has been executed 1 times
The for loop has been executed 2 times
The for loop has been executed 3 times
The for loop has been executed 4 times
```

Diagram 10.5.1: Output for the code block with the **for** iteration.

The statement $i++ is shorthand for $i = $i + 1, which simply increments the variable $i by 1. Similarly, $i-- will decrement the variable $i by 1.

The main code is executed while $i is less than 5, the **for** loop simply prints the numbers from 0 to 4 on a webpage.

Example:
The following code, stored in **testforstr.php**, prints the text message and the counter six times:

```
<HTML><HEAD><TITLE>Using For Loop</TITLE></HEAD>
<BODY>
<?php
  for ($i=1; $i<7; $i++) {   echo "The message is printed $i times!<BR>"; }
?>
</BODY></HTML>
```

Output: (Refer diagram 10.5.2)

```
The message is printed 1 times!
The message is printed 2 times!
The message is printed 3 times!
The message is printed 4 times!
The message is printed 5 times!
The message is printed 6 times!
```

Diagram 10.5.2: Output for testforstr.php.

The while-loop

The while-loop is very similar to the for-loop. The while statement execute a block of code as long the condition is **TRUE**.

Syntax:

```
while (<Some Condition Which Is True>) {<Code To Be Executed>;  }
```

Example:
The following code, stored in **testwhile.php**, demonstrates a loop that will continue to run as long as the variable **$ctr** is less than **11**. The variable **$ctr** will be incremented by **1** each time the loop runs:

```
<HTML><HEAD>
  <TITLE>Using while loop to generate a table for multiple of  2</TITLE>
</HEAD><BODY>
<?php
  $ctr=1;
  echo "The multiples of the number 2 upto 10 times are:<BR>";
  while($ctr < 11) {
      echo "2 x $ctr  = " . ($ctr*2) . "<BR>";
      $ctr++;    }
?>
</BODY></HTML>
```

```
The multiples of the number 2 upto 10 times are:
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Diagram 10.6: Output for testwhile.php.

Output: (Refer diagram 10.6)

Similarity between the **for** and **while** loops:

for LOOP	while LOOP

for (a; b; c) {
  d;
}
	a;
	while (b) {
	  d;
	  c;
	}

Using Iteration To Control An Array

Often a **while** loop is used to run though an array as shown below:

```
while (list($key, $val) = each($array))
    {  echo "<P>$key => $val</P>\n";  }
```

The condition in the **while loop** uses several functions and an assignment. Each time the function **each()** is called, it returns an array with the key and value from $array. When there are no elements left in $array, **each()** returns false. This is what makes the whole condition false and makes the while loop stop.

The **list()** function can be used on the left side of an assignment to extract values from an array. Use list() like this, although it would be much easier to assign the variables one at a time:

```
list($a, $b, $c, $d) = array(1, 2, 3, 4); // $a == 1, $b == 2, $c == 3, $d == 4
```

So the condition in the while-loop prepares $key and $val for the body of the loop. When all key-value pairs have been examined, the loop terminates because **each()** return false, while then becomes the value of the assignment and therefore the value of the condition.

After an array has been run through like this, **each()** will return **false** if used on the same array. Call **reset($array)** to make **each()** start from the beginning again.

Example:
The following code, stored in **arywhile.php**, makes use of the functions **reset()** and **each()**, which will build a table that shows the keys and values of an array:

```
<HTML><HEAD><TITLE>Presenting Data In A Tabular Layout</TITLE>
</HEAD><BODY>
<?php
  $array = array('India' => 'New Delhi', 'Pakistan' => 'Islamabad', 'Sri Lanka' =>
                 'Colombo');
// Start the table
  echo "<TABLE Border='1'>\n";
// Print the first row with headers
  echo "<TR>\n";
  echo "  <TH>Countries</TH>\n";
  echo "  <TH>Capitals</TH>\n";
  echo "</TR>\n";
// Ensure that the each() function starts from the start:
  reset($array);
  while (list($key, $value) = each($array)) {
  // Print a single row
      echo "<TR>\n";
      echo "<TD>$key</TD>\n";
      echo "<TD>$value</TD>\n";
      echo "</TR>\n";
  }
// End the table
  echo "</TABLE>\n";
?>
</BODY></HTML>
```

Countries	Capitals
India	New Delhi
Pakistan	Islamabad
Sri Lanka	Colombo

Diagram 10.7: Output for arywhile.php.

Output: (Refer diagram 10.7)

The important thing to notice in the above code is the way HTML tables can be built using PHP codespec. Tables in HTML begins with <TABLE> opening tag. This is followed by any number of rows, each row consists of a number of cells. Finally the </TABLE> tag closes the HTML table.

The do...while Statement

The **do ... while** statement will execute a block of code at least once. The loop will then repeat as long as a condition is TRUE.

Syntax:

```
do {      <Code To Be Executed>;
} while (<Condition>);
```

Example:
The following code, stored in **testdo.php**, will increment the value of **$ctr** at least once and it will continue incrementing the value held by the variable **$ctr** till it reaches a value less than 10**:**

```
<HTML><HEAD>
  <TITLE>Using do ... while loop to generate a table for multiple of 5</TITLE>
</HEAD><BODY>
<?php
  $ctr=0;
  echo "The multiples of the number 5 upto 10 times are:<BR>";
  do {
      $ctr++;
      echo "5 x $ctr  = " . ($ctr*5) . "<BR>";
  } while ($ctr < 10);
?>
</BODY></HTML>
```

Output: (Refer diagram 10.8)

The multiples of the number 5 upto 10 times are:
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50

Diagram 10.8: Output for testdo.php.

The foreach loop

The **foreach** loop provides an easy way to iterate through arrays. The **foreach** works only on arrays and will throw an error when used on a variable with a different data type **or** an un-initialized variable.

Syntax:

```
foreach (<array_expression> as <$value>) <Statement>
```

This loops over the array given by **array_expression**. On each loop, the value of the current element is assigned to **$value** and the internal array pointer is advanced by one.

foreach (<array_expression> as $key => $value) <Statement>

This does the same thing, except that the current element's key will be assigned to the variable **$key** on each pass of the loop.

REMINDER

When the **foreach** loop first starts executing, the internal array pointer is automatically **reset** to the first element of the array. This means that **reset()** need not be called before a **foreach** loop.

Example:
The following code, stored in **foreachval.php**, is an example that demonstrates using value only:

```
<HTML><HEAD><TITLE>Extracting Values Using The foreach Statement</TITLE>
</HEAD><BODY>
<?php
  $age = array('Hansel', 23, '01/01/1982');
  $ctr = 1;
  foreach ($age as $value) {
      echo "Value held in position $ctr is: $value.<BR>";
      $ctr++;
  }
?>
</BODY></HTML>
```

Output: (Refer diagram 10.9)

Value held in position 1 is: Hansel.
Value held in position 2 is: 23.
Value held in position 3 is: 01/01/1982.

Diagram 10.9: Output for foreachval.php.

REMINDER

The **foreach()** loop operates on a **copy** of a specified array and not the array itself. Hence, the array pointer is **not modified** as with the **each()** loop. Changes to the array element returned are not reflected in the original array. However, the internal pointer of the original array is advanced when the copy of the array is being processed. Assuming the foreach() loop runs to completion, both the array's internal pointers will be at their end.

Example:
The following code, stored in **foreachkey.php**, is an example that demonstrates using key and value:

```
<HTML><HEAD>
  <TITLE>Extracting Name-Value pairs Using The foreach Statement</TITLE>
</HEAD><BODY>
<?php
  $age = array("Name" => 'Hansel', "Age" => 23, "Birthdate" => '01/01/1982');
  $ctr = 1;
```

```
  foreach ($age as $name => $value) {
      echo "The name-value pair on position $ctr is: \$age[$name] =>
            $value.<BR>";
      $ctr++;
  }
?>
</BODY></HTML>
```

```
The name-value pair on position 1 is: $age[Name] => Hansel.
The name-value pair on position 2 is: $age[Age] => 23.
The name-value pair on position 3 is: $age[Birthdate] => 01/01/1982.
```

Output: (Refer diagram 10.10)

Diagram 10.10: Output for foreachkey.php.

The following code, stored in **foreachmlt.php**, is an example that demonstrates multi-dimensional arrays:

```
<HTML><HEAD>
  <TITLE>Extracting Values From A Multi-Dimensonal Array Using foreach
        Statement</TITLE>
</HEAD><BODY>
<?php
  $staff [0][0] = "Sharanam";
  $staff [0][1] = "Hansel";
  $staff [1][0] = "Ivan";
  $staff [1][1] = "Chhaya";
  $ctr1 = 0;
  foreach ($staff as $value1) {
      $ctr2 = 0;
      foreach ($value1 as $value2) {
          echo "Value held in position $ctr1 segment $ctr2 is: $value2<BR>";
          $ctr2++;
      }
      $ctr1++;
  }
?>
  </BODY>
</HTML>
```

```
Value held in position 0 segment 0 is: Sharanam
Value held in position 0 segment 1 is: Hansel
Value held in position 1 segment 0 is: Ivan
Value held in position 1 segment 1 is: Chhaya
```

Output: (Refer diagram 10.11)

Diagram 10.11: Output for foreachmlt.php.

Infinite loops

Surprisingly, infinite loops can sometimes be helpful in scripts. An infinite loop never terminates without outside influence. The most popular way is to break out of the loop and/or exit the script entirely from within the loop whenever a condition is matched. Relying on user input to terminate the loop is also possible.

WARNING

While learning PHP, it is almost certainly best to steer clear of infinite loops, as they can really cause problems.

Using **Ctrl+C** to terminate PHP operation is restricted to command line execution of the script only.

Example:

```
<?php
  while(1) {     print "In loop!\n";  }
?>
```

As "1" evaluates to true, the loop will continue forever.

Usually it is preferred to write the infinite loops as shown below:

```
<?php
  for (;;) {   print "In loop!\n";  }
?>
```

In the above example, the declaration, condition and action parts are missing in the **for** loop meaning that it will always loop.

Special Loop Keywords

There are two special keywords that can be used with loops and they are **break** and **continue**.

REMINDER

The break keyword has been used while working with case switching. It is used there to exit a switch/case block and it has the same effect with loops.

When used inside loops in order to manipulate the loop behavior, **break** causes PHP to exit the loop and **continue** causes PHP to skip the remaining code block in the current loop iteration and go onto the next iteration.

Example: (Contents of spword.php)

```
<HTML><HEAD><TITLE>Using continue and break</TITLE></HEAD><BODY>
<?php
  for ($i = 1; $i < 10; $i++) {
      if ($i == 5) {
          echo("Loop continued at $i<BR>");
          continue;  }
```

```
    if ($i == 8) {
        echo("Loop terminated at $i<BR>");
        break; }
    echo("Number $i<BR>");
  }
?>
</BODY></HTML>
```

```
Number 1
Number 2
Number 3
Number 4
Loop continued at 5
Number 6
Number 7
Loop terminated at 8
```

Diagram 10.12: Output for spword.php.

Output: (Refer diagram 10.12)

In the above example, a for loop iterates for the first four rounds. While in the fifth round the first IF condition becomes **TRUE** (i.e. $i holds 5). At this moment a message stating Loop continued at 5 is rendered and since **continue** is encountered thereafter, the message stating Number 5 is skipped and the loop iterates to the next round. While in the eighth round the loop terminates based on the second IF condition evaluating to **TRUE** which instructs the loop to exit using the **break** keyword.

Loops Within Loops

Loops can be stacked up by having one loop inside the body of another (nested loops).

Example: (Contents of nestlpfrst.php)

```
<HTML><HEAD><TITLE>Using nested loops</TITLE></HEAD>
<BODY>
<?php
  for ($a = 1; $a < 3; $a++) {
      for ($b = 0; $b < 7; $b = $b + 2) {
          for ($c = 10; $c > 8; $c--) {
              echo("A holds: $a, B
                    holds: $b, C holds:
                    $c<BR>");
          }
      }
  }
?>
</BODY></HTML>
```

```
A holds: 1, B holds: 0, C holds: 10
A holds: 1, B holds: 0, C holds: 9
A holds: 1, B holds: 2, C holds: 10
A holds: 1, B holds: 2, C holds: 9
A holds: 1, B holds: 4, C holds: 10
A holds: 1, B holds: 4, C holds: 9
A holds: 1, B holds: 6, C holds: 10
A holds: 1, B holds: 6, C holds: 9
A holds: 2, B holds: 0, C holds: 10
A holds: 2, B holds: 0, C holds: 9
A holds: 2, B holds: 2, C holds: 10
A holds: 2, B holds: 2, C holds: 9
A holds: 2, B holds: 4, C holds: 10
A holds: 2, B holds: 4, C holds: 9
A holds: 2, B holds: 6, C holds: 10
A holds: 2, B holds: 6, C holds: 9
```

Diagram 10.13.1: Output for nestlpfrst.php.

Output: (Refer diagram 10.13.1)

In the above example, the outer loop starts first and completes last. It iterates **two** times.

The middle loop iterates **four** times, but it runs twice, i.e. once for each iteration made by the outer loop.

The inner loop iterates **twice**, but runs eight times, i.e. once for each iteration made by the middle loop.

In this situation, using break is a little more complicated, as it only exits the closest loop.

Example: (Contents of nestlpscnd.php)

```
<HTML><HEAD><TITLE>Using nested loops with a break</TITLE>
</HEAD><BODY>
<?php
  for ($a = 1; $a < 3; $a++) {
      for ($b = 0; $b < 7; $b = $b + 2) {
          for ($c = 10; $c > 8; $c--) {
              echo("A holds: $a, B holds: $b, C holds: $c<BR>");
              break;
          }
      }
  }
?>
</BODY></HTML>
```

Output: (Refer diagram 10.13.2)

As seen, the inner most loop only loops once because of using **break**. However, the other loops execute several times.

```
A holds: 1, B holds: 0, C holds: 10
A holds: 1, B holds: 2, C holds: 10
A holds: 1, B holds: 4, C holds: 10
A holds: 1, B holds: 6, C holds: 10
A holds: 2, B holds: 0, C holds: 10
A holds: 2, B holds: 2, C holds: 10
A holds: 2, B holds: 4, C holds: 10
A holds: 2, B holds: 6, C holds: 10
```

Diagram 10.13.2: Output for nestlpscnd.php.

More control can be exercised by specifying a number after break, such as **break 2** to break out of two loops or switch/case statements.

Example: (Contents of nestlpthrd.php)

```
<HTML><HEAD>
  <TITLE>Using nested loops with break for two loops</TITLE></HEAD><BODY>
<?php
  for ($a = 1; $a < 3; $a++) {
      for ($b = 0; $b < 7; $b = $b + 2) {
          for ($c = 10; $c > 8; $c--) {
              echo("A holds: $a, B holds: $b, C holds: $c<BR>");
              break 2;
          }
      }
  }
?>
</BODY></HTML>
```

```
A holds: 1, B holds: 0, C holds: 10
A holds: 2, B holds: 0, C holds: 10
```

Diagram 10.13.3: Output for nestlpthrd.php.

Output: (Refer diagram 10.13.3)

This time the loop only executes twice, because the inner most loop calls **break 2**, which breaks out of the inner most loop and the one above the inner most, so only the outer most loop will iterate. This could even be **break 3**, meaning break out of all three loops and continue normally.

It is important to remember that "break" works both on loops and switch/case statements.

Example:

```
<HTML><HEAD>
  <TITLE>Using nested loops with break for two loops</TITLE></HEAD><BODY>
<?php
  for ($a = 1; $a < 3; $a++) {
      for ($b = 0; $b < 7; $b = $b + 2) {
          for ($c = 10; $c > 8; $c--) {
              switch($c) {
              case 10:
                  echo("A holds: $a, B holds: $b, C holds: $c<BR>");
                  break 2;
              case 9:
                  echo("A holds: $a, B holds: $b, C holds: $c<BR>");
                  break 3;
              }
          }
      }
  }
?>
</BODY></HTML>
```

```
A holds: 1, B holds: 0, C holds: 10
A holds: 1, B holds: 2, C holds: 10
A holds: 1, B holds: 4, C holds: 10
A holds: 1, B holds: 6, C holds: 10
A holds: 2, B holds: 0, C holds: 10
A holds: 2, B holds: 2, C holds: 10
A holds: 2, B holds: 4, C holds: 10
A holds: 2, B holds: 6, C holds: 10
```

Output: (Refer diagram 10.13.4)

Diagram 10.13.4: Output for nestlpfrth.php.

The **break 2** will break out of the switch/case block and also out of the **$c** loop, whereas the **break 3** will break out of those two and also the **$b** loop. To break out of the loops entirely from within the switch/case statement, **break 4** is required.

Mixed-Mode Processing

A key concept in PHP is that the PHP parsing mode can be toggle whenever and as often as wanted and it can even be done from inside a code block. Here is a basic PHP script:

```
<?php
  if ($temp1 == $temp2) {
     print "Some PHP Process";
     print "Some PHP Process";
     print "Some PHP Process";
      ...[snip]...
```

```
    print "Some PHP Process";
    print "Some PHP Process";
  }
?>
```

As seen, there are a lot of print statements that will only be executed if the variable **$temp1** is the same as variable **$temp2**. All the output is encapsulated into print statements, but PHP allows to exit the PHP code island while still keeping the **if** statement code block open.

Consider the sample shown below:

```
<?php
  if ($temp1 == $temp2) {
?>
  Some PHP Process
  Some PHP Process
  Some PHP Process
      ...[snip]...
  Some PHP Process
  Some PHP Process
<?php
  }
?>
```

The "Some PHP Process" lines are still only sent to output if **$temp1** is equal to **$temp2**, but the PHP mode is exited to print it out. The PHP mode is then reentered to close the **if** statement and continue. This makes the whole script easier to read.

Hands On Exercises

1. Generate a PHP script that will display the multiples of the numbers 1 to 10, upto 10 times. The output should appear within the console window, in a columnar format of 5 sets per row. Refer diagram 10.14.1.

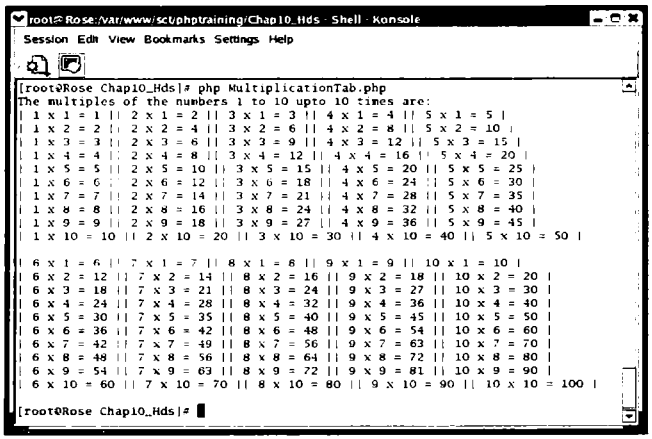


Diagram 10.14.1: Output for MultiplicationTab.php.

2. Generate a PHP script that will display the grade on the bases of marks as follows: (Refer diagram 10.14.2)
   - Distinction – 80 and above
   - First Class – From 60 to below 80
   - Second Class – From 45 to below 60
   - Pass Class – From 35 to below 45
   - Fail – Below 35

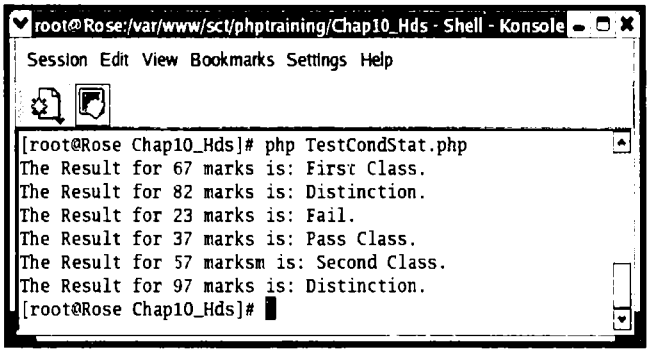


Diagram 10.14.2: Output for TestCondStat.php.

Test the above program using the following set of values stored in an array: 67, 82, 23, 37, 57 and 97

3. Generate a list of possibilities for two digit numbers, consisting of numbers 0 to 4. Using nested **for** loop, obtain an output as shown in diagram 10.14.3.

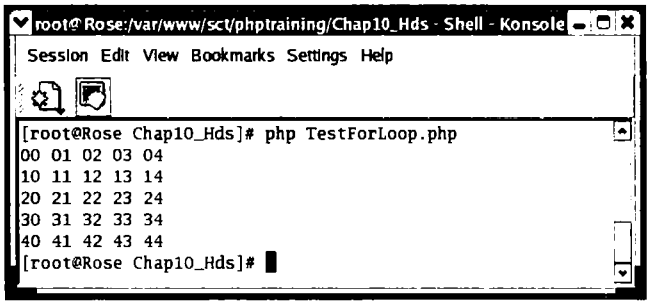


Diagram 10.14.3: Output for TestForLoop.php.

SECTION III: WORKING WITH PHP

Using Functions In PHP

PHP draws its power from a large collection of built-in functions. At last count there were more than 700 such built-in functions available, additionally several PHP developers have extended the PHP core by adding their own library of functions, usually available for free download or very modestly priced.

Functions are an important tool in any programming language. Functions are code snippets that accept values and produce results although there are some functions that do not need to be supplied with values to produce a result. Functions are used in situations where the code snippet is repeated across scripts or even in the same script. The block of code held by the function is not immediately executed, but is called by a script as and when its execution is required.

Functions can either be built-in or user-defined. PHP has several built-in functions, the most common one being the **print()** function. In the examples demonstrated in earlier chapters, the **print()** function was extensively used to output strings to a Web-browser.

Example:

```
print("Sharanam stays in Mumbai");
```

The values passed to a function prior its code executing are called **arguments**. When arguments are passed to functions they reside in the functions pre-created **parameters**. In general, the arguments passed to a function are included within its parentheses. However, there are some exceptions. For some of PHP's built-in functions, the programmer can choose to use **or** ignore the parentheses while passing it an argument. The **print()** function is one such function.

```
print "Sharanam stays in Mumbai";
```

works just as well as:

```
print("Sharanam stays in Mumbai");
```

A primary advantage of user-defined functions is code reusability. If functions were avoided in an application, the same code snippet would have numerous repetitions, irrespective of whether it is being called within a single **or** in multiple PHP file(s).

The Anatomy Of A Function

Writing user-defined functions is as simple as using the function command as:

```
function myFunction() {
// Some PHP Code spec
}
```

While creating a function, a **unique** function name follows the **function** keyword (in this case myFunction). A pair of opening and closing parentheses follows the function name. If arguments were to be accepted, the variables acting as the functions parameter holders must be mentioned within the parentheses. The actual function code, (called the function's **signature**), which follows is enclosed within the curly braces.

The most basic example will look as:

```
function sayHello() {
    print("Hello Sharanam! How are you doing today.<BR>");
}
```

The above function simply prints the text, "Hello Sharanam! How are you doing today.", in a Browser. It can be called just like a regular PHP function from within any HTML page:

```
<HTML><BODY>
<?
// Defining the function sayHello
  function sayHello() {
      print("Hello Sharanam! How are you doing today.<BR>");
  }
// Calling the function defined above
  sayhello();
?>
</BODY></HTML>
```

Save the above code snippet as the file **testFunction.php** under **Chap11_Cds** directory in the **sct.phptraining.com** domain. Run the file using a Web Browser to get the output as shown in diagram 11.1.

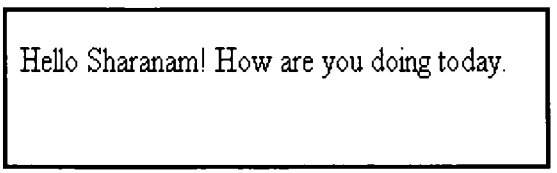


Diagram 11.1: Output for testFunction.php.

Functions With Arguments

The following example illustrates a function that can receive variables as input and return data that can be assigned to another variable. For instance, the function **myTable()** creates a single row of data in a table and the data is passed to the function:

```
function myTable($text) {
    print("<TABLE Border=0 CellPadding=0 CellSpacing=0><TR>");
    print("<TD>$text</TD>");
    print("</TR></TABLE>");
}
```

In this case, the function can be used as:

```
myTable("Sharanam stays in Mumbai");
```

Functions With Multiple Arguments

Different functions would require different arguments. Functions can take multiple arguments. Each argument will be separated from the other by a comma. Functions can also be executed without any arguments, in which case only the parenthesis is required after the function name. It all depends on the function and what it does.

Example 1:
The function **myTable()**, defined earlier, can be extended to allow the user to change the Border, CellPadding and CellSpacing values as:

```
function myTable($text, $border="0", $cellp="0", $cells="0") {
    print("<Table Border=$border CellPadding=$cellp CellSpacing=$cells><TR>");
    print("<Td>$text</Td>");
    print("</TR></TABLE>");
}
```

The **myTable()** function would then be called as:

```
<HTML><BODY>
<?
// Defining the function, which will create a single row single
column table and populate it.
  function myTable($text, $border="0", $cellp="0", $cells="0") {
      print("<Table Border=$border CellPadding=$cellp
            CellSpacing=$cells><TR>");
      print("<Td>$text</Td>");
      print("</TR></TABLE>");
  }
// Calling the myTable() function defined above.
  myTable("Sharanam is an Analyst Designer", 0, 1, 1);
?>
</BODY></HTML>
```

Save the above code snippet as the file **gettable.php** under the **Chap11_Cds** directory in the **sct.phptraining.com** domain. Run the file using a Web Browser to get an output as shown in diagram 11.2.

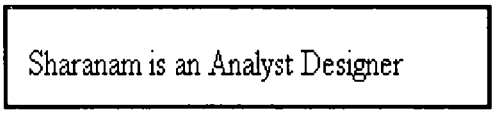


Diagram 11.2: Output for gettable.php.

In the above example, **myTable()** assigns $border, $cellp and $cells a default value of 0 so that if no values are passed to the function the default values will be considered. In the above call to **myTable()**, a string "Sharanam is an Analyst Designer" is passed as the text to be displayed in the table along with a Border of **0** and a CellPadding and CellSpacing of **1**.

Example 2:
Functions can be used to return data as well as to receive it. A function can calculate **car mileage** based on the number of kilometers run **and** the liters of fuel consumed. This will require these two values passed as arguments when the function is called. The result of the calculation can be assigned to a variable as:

```
$mileage = calculate_mileage($kms, $liters);
```

The above code spec calls the function **calculate_mileage()** with two parameters. These parameters must be loaded with appropriate values (as arguments) when the function is called. After performing the calculation **calculate_mileage()** the function will store the result in a variable **$mileage**.

Consider writing the above described function called **calculate_mileage($kms, $liters)**. It must receive the two numbers as input and will then return the result as output. Write the function as:

```
function calculate_mileage($kms, $liters) {
    $result = $kms/$liters;
    return($result);
}
```

Use the function **calculate_mileage()** in the code spec as:

```
$mileage = calculate_mileage(300,30);
```

Only a single **value** can be returned from a function. This is done by using the **return** statement. Any variable can be returned, as long as it is just one variable. The variable returned can be an integer, a string, a database connection, etc. The **return** keyword loads the function's return variable with a value to be returned, then exits the function immediately. Only **return;** can also be used which means exit without sending a value back.

In this case, the value of $mileage will be 10. The function divides the number of kilometers by the liters of fuel consumed received as input, assigns that value to $result. The **return** command returns the value of $result to the caller of the function.

Functions Accepting Values By Reference

Values can also be returned by their reference via a function. Functions can accept parameters by reference and also return values by reference. This is done with the reference operator **&**.

Marking a variable as **passed by reference** is done within the function definition, not in the function call as:

```
function cal(&$num1, &$num2) {
```

is correct, whereas

```
$mynum = cal(&5, &10);
```

is wrong.

This means that if there is a function being used frequently across the project, edit the function definition to make it take variables by reference. Passing by reference is often a good way to make scripts shorter and easier to read. However a script's performance can deteriorate by doing this.

Consider the following code:

```
<?php
  . . .
  function myFunc($number) {
      return $number * $number;   }
  $val = myFunc($val);
  . . .
  function myFuncRef(&$number) {
      $number = $number * $number; }
  myFuncRef($val);
  . . .
?>
```

The first example passes a copy of **$val** while calling the function **myFunc**, multiplies the copy, then returns the result which is then copied back into **$val**. The second example passes **$val** in by reference while calling the function **myFuncRef** and it is modified directly inside the function hence the result is not returned to **$val**. This is because value held by **$val** was worked on (i.e. multiplication was performed on **$val**) directly inside the second function **MyFuncRef**.

One key thing to remember is that by reference is a reference to a variable. If a function is defined as accepting a reference to a variable, it is not possible to pass a constant into it. That is, given our definition of **myFuncRef()**, the function cannot be called using as:

```
myFuncRef(10);,
```

as 10 is not a variable, so it cannot be treated as a reference.

Functions Returning Values By Reference

Another thing to know about user defined functions and by reference, is how to **return** values by reference. To specify a function that returns a value by reference, place the ampersand (i.e. & reference) operator before the function name and specify that the result of the function is to be returned by a reference to it as opposed to using a copy of the value.

```
...
function &myRefFunc($number) {
    return $number * $number;
}
$val =& myRefFunc(10);
...
```

Globally Accessing Variable Within Functions

When a function uses a variable, there are several issues that one should be aware of. It's important to know that if a variable is declared inside a function, that variable will only be available to the rest of that function's code. This is called a **local** variable. If an attempt is made to access this variable in another function, it will not hold any value.

But sometimes it might be a need to access some important information outside the function, without using an argument. One way to do this, is using the **global** statement.

Consider the following example:

```
$kms=300;
$liters=30;

function calculate_mileage() {
  $result = global $kms/global $liters;
  return($result);
}
```

The previous script is expanded here. The above function definition uses **global** to inform the function that the **$kms** and **$liters** variable has been declared outside it. If the global statement was not applied, the function would have returned zero, because **$kms** and **$liters** would have been **NULL**. The function would simply not know the value of these variables if it was not informed that these variables were assigned outside the function.

Be very careful while using a global statement, because if these values (held by the variables used with global statement) are modified within a function, the original values will be modified as PHP uses the variable itself and not a copy when used with global statement.

Globalizing Functions

For small applications, such as a one-page site, functions can be placed at the top of the PHP file, i.e. before the rest of the code spec. For larger applications and multi-page sites, it might be prudent to consider putting all the functions in a special file. These functions can then be made available to the code spec using the **include()** function in each of the Web pages that require those functions. This technique gives the functions portability thereby reducing the size of the web application. It also helps program execution and code reuse by consolidating all the **functions** in one file. A recommended name for this file could be **functions.php**. Consider the following file named functions.php, which holds the following code spec:

```
<?
/*
  Date :- 02/05/05
  Author :- Sharanam Shah
  Filename :- functions.php
  Purpose :- Holds all the common functions used across the
             application.
*/
// Defining the function, which will calculate mileage.
  function calculate_mileage($kms, $liters) {
      $result = $kms/$liters;
      return($result);
  }

// Defining the function, which will create a single row single
column table and populate it.
  function myTable($text, $border="0", $cellp="0", $cells="0") {
      print("<Table Border=$border CellPadding=$cellp CellSpacing=$cells><TR>");
      print("<Td>$text</Td>");
      print("</TR></TABLE>");
  }
?>
```

The following file named **candreg.php** will include the **function.php** file as:

```
<?
/*
  Date:- 05/02/05
  Author:- Sharanam Shah
  Filename:- cand_reg.php
  Purpose:- Allows registering as a candidate. Control is shifted
   .          to this page when the user selects the SIGNUP -->
              Candidate option */
/* A variable named $author is defined to store the name of the
programmer who designed/coded the page. */
  $author = "Sharanam Shah";
```

```
/* A variable named $helppage is defined to store the name of the
HTML page that will serve help on demand. */
  $helppage="cand_reg";
/* The functions.php files is included to make available the
functions defined. */
  include "../includes/functions.php";
...
?>
```

Some PHP applications use filenames such as **functions.inc**, which omits the .php extension. This kind of extension will make the file vulnerable. The .php extension is used to keep the code spec private. If visitors load the **functions.php** file directly, nothing will be rendered on screen however, if they load a file called **functions.inc**, the entire source code is rendered, which could give out sensitive information such as path locations, passwords, database information and so on.

Functions In Files

In PHP it is also possible to use an **entire file** as a function. This makes it possible to save functions in different files and then use them in different scripts. This helps expand the concept of **code-reuse**. This can be done using a pre-defined PHP function **include()**. This function can use a value returned from an external file, instead of calling a function declared in the same file.

The following is a file **getdays.php**, which is to be used within the file **script.php**. The **getdays.php** file includes the following code spec:

```
<?php   return (time() - mktime(0, 0, 0, 1, 21, 1982))/86400;   ?>
```

The **script.php** file uses it as:

```
<?php
  $days = include('getdays.php');
  echo "Value returned by
        getdays.php: $days\n";
?>
```

This will work just like any other code spec, except that code has been re-used from an external file based source.

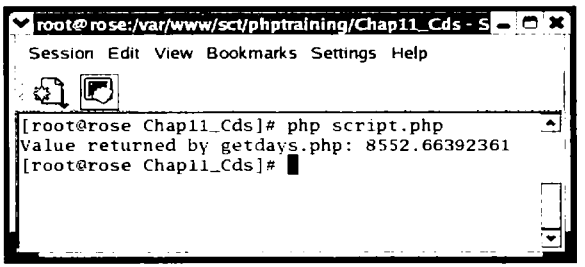


Diagram 11.3: Output for script.php.

A User-Defined Function

Having understood the technique of declaring and calling simple functions, these skills can now be used to generate dynamic results. The PHP illustration described below is a useful user-defined function, which provides dynamism within web based applications.

While working with data entry forms on a HTML page, it is quite common to dynamically populate a Drop Down List Box. A Drop Down List Box is an object that allows the user a choice of option(s) from a large set of possibilities. Apart from being able to display single or multiple option(s), a Drop Down List Box can be instructed to allow selection of single value or multiple values. Another tricky part is assigning a selected / default value to a Drop Down List Box.

The function mentioned below, generates a Drop Down List Box with values/data retrieved from an Oracle database table (Oracle OCI8 functions and concepts explained in Chapter 14). The Drop Down List Box will have the following properties:

- It accepts an SQL query and populates the Drop Down List Box based using the field name passed as a parameter
- It allows setting a default value for the Drop Down List Box
- It allows setting a display prompt as an entry in the Drop Down List Box
- It allows setting additional attributes like defining action for Drop Down List Box's event, setting display styles and so on

```
function display_ddlb($p_value_field, $p_display_field, $p_query, $p_cntrl_name,
                      p_sel_value, $p_initial, $p_function, $p_rcq) {
  $parsed = ociparse($p_rcq, $p_query);
  ociexecute($parsed);
  echo "<SELECT Name='" . $p_cntrl_name . "' " . $p_function . ">";
  if ($p_initial <> "") {  echo "<OPTION Value=''>$p_initial</OPTION>";  }
  while ($fn_select_arr = oci_fetch_array($parsed)) {
      if ($p_sel_value == $fn_select_arr["$p_value_field"]) {
          echo "<O`PTION Value='" . $fn_select_arr["$p_value_field"] . "'
                Selected>" . $fn_select_arr["$p_display_field"] . "</OPTION>";
      }
      else {
          echo "<OPTION Value='" . $fn_select_arr["$p_value_field"] . "'>" .
                $fn_select_arr["$p_display_field"] . "</OPTION>";
      }
  }
  echo "</SELECT> ";
}
```

Understanding The Above Function

The parameters in the above function are:

p_cntrl_name	Accepts a name for the Drop Down List Box
p_query	Accepts an SQL query accountable to retrieve desired data
p_value_field	Accepts a column name from the result set (formed after execution of the SQL query) which forms the value part of the Drop Down List Box when retrieved

p_display_field	Accepts a column name from the result set (formed after execution of the SQL query) which forms the display part of the Drop Down List Box when selected/displayed on a page
p_sel_value	Accepts the current value displayed by the Drop Down List Box. This is useful in update mode for setting the Drop Down List Box value returned from the database based on the current record
p_initial	Accepts the initial value that the Drop Down List Box will display (Default Value)
p_function	Accepts any extra HTML attributes to be attached to the Drop Down List Box
p_rcq	Accepts the Database handle

The function processes as follows:

The data for populating the name-value pair for each option in the Drop Down List Box is retrieved by an SQL query passed via the **p_query** parameter. This query is then parsed and executed to generate the STATEMENT object.

The rendering of the HTML tag for creating the Drop Down List Box follows. The value passed via the **p_cntrl_name** parameter is used to assign a name to the Drop Down List Box object, while the value passed via the **p_function** parameter defines addition HTML/JavaScript attributes (if any).

If a value is passed via the **p_initial** parameter, the first item, for the Drop Down List Box, is created having set the display text as held in **p_initial** parameter. This is followed by iteration through the Statement object created earlier.

Each row in the statement object consists of two columns, display text and a value. A check is made to compare the value with the value passed via the **p_sel_value** parameter. If a match is found, then that value is set to be SELECTED. Finally, the closing HTML tag for the Drop Down List Box is rendered.

PHP Functions

One of things that make PHP so easy to use is the wealth of built-in functions. These functions act as tools to simplify and speed up many of the everyday tasks required during the creation of a full-fledged application.

There are built in PHP functions for working with just about everything an application requires such as strings, arrays, databases, dates and times, email, numbers, even functions for working with other functions and so on. Its impossible to cover all of built in PHP functions in this material hence a few of the important functions will be introduced and elaborated on.

print()

The function **print()** basically accepts a string as an argument and outputs this to a Browser. It works with or without parentheses as shown below:

```
...
print "Testing the print function without parenthesis\n";
print ("Testing the print function with parenthesis\n");
...
```

include()

The function **include()** allows embedding one file inside another. To understand how it works, first create a file called **textInclude.txt** and write some text in it as:

```
This is testing the use of Include() function.
```

Next, create a new PHP file called **testInclude.php** and place the following code spec in it:

```
<?php
  include "textInclude.txt";
?>
```

REMINDER

Just as in the **print()** function, the parentheses for the **include()** function are optional.

WARNING

Make sure both files are saved to the same directory on the web server.

When **testInclude.php** is run, the text placed in **textInclude.txt** will be sent to the browser that invoked it. Refer diagram 11.4. In the same manner html **or** php files can be included.

This is testing the use of Include() function.

Diagram 11.4: Output for testInclude.php.

header()

This function allows sending a raw HTTP header to the client browser. Most of its applications are technical, but one of its most common usages is relatively simple, redirecting a user elsewhere (i.e. to another URL). This has the same effect as the META redirection tag in HTML.

One of the most useful feature of this function is to redirect. This will tell the browser to load a different page.

Consider the following example:

```
<?php
  header( "Location:http://rediff.com/");
  exit;
?>
```

REMINDER

If the **header()** function is used, it must be called **before** sending any information to the browser, either from PHP or HTML. If not, then an error such as "headers already sent" will be displayed.

This function also takes a second parameter, which is optional and can hold either TRUE or FALSE to determine if the current header should replace the previous header. By default, it is **TRUE**. However, if **FALSE** is passed as the second argument, multiple headers of the same type can be FORCED. Consider the following example:

```
<HTML><BODY>
<?php
  header("WWW-Authenticate: Negotiate");
  header("WWW-Authenticate: NTLM", FALSE);
?>
</BODY></HTML>
```

The above example sends header information multiple times. The first header function does not use second argument while the second header function uses FALSE as the second argument to send or force the header information the second time.

phpinfo()

The **phpinfo()** function is used to describe the version (and a host of other attributes) of PHP installed at and bound to the Web server. The information displayed in a Browser as the output of this function is useful for troubleshooting PHP.

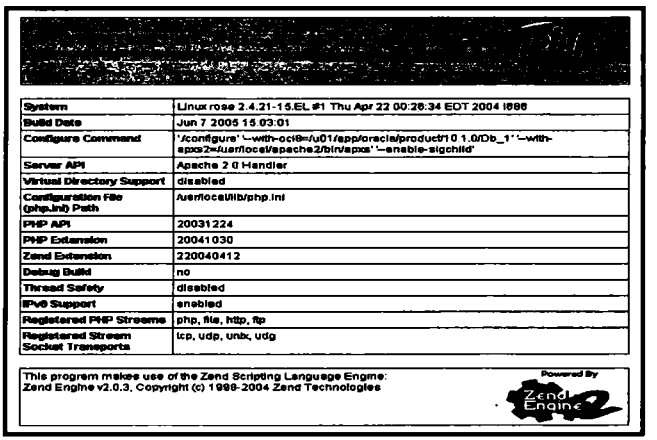
The **phpinfo()** function options:

Name	Description
INFO_GENERAL	The configuration line, php.ini location, build date, Web Server, System and more
INFO_CREDITS	PHP credits
INFO_CONFIGURATION	Local and master values for php directives
INFO_MODULES	Loaded modules
INFO_ENVIRONMENT	Environment variable information
INFO_VARIABLES	All predefined variables from EGPCS (Environment, GET, POST, Cookie, Server)

Name	Description
INFO_LICENSE	PHP license information
INFO_ALL	Shows all of the above. This is the default value

Example: phpinfoSample.php

```
<HTML><BODY>
<?php
  // Shows all PHP
  information
     phpinfo();
  // Shows only the
  general information
     phpinfo(INFO_GENERAL);
?>
</BODY></HTML>
```



System	Linux rose 2.4.21-15.EL #1 Thu Apr 22 00:26:34 EDT 2004 i686
Build Date	Jun 7 2005 15:03:01
Configure Command	'/configure' '--with-oci8=/u01/app/oracle/product/10.1.0/Db_1' '--with-apxs2=/usr/local/apache2/bin/apxs' '--enable-sigchild'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/lib/php.ini
PHP API	20031224
PHP Extension	20041030
Zend Extension	220040412
Debug Build	no
Thread Safety	disabled
IPv6 Support	enabled
Registered PHP Streams	php, file, http, ftp
Registered Stream Socket Transports	tcp, udp, unix, udg

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.0.3, Copyright (c) 1998-2004 Zend Technologies

Powered By Zend Engine 2

Diagram 11.5: Page generated after executing the function **phpinfo(**INFO_GENERAL**)**.

PHP Server Variables

All Web servers hold information such as the URL from which the user request came from, which browser is being used and so on. This information is stored in several **server variables**.

In PHP, **$_SERVER** is a reserved variable that contains all such Web server information. $_SERVER is a **global** variable, which means that it's available for access by the codespec of all PHP scripts.

The following example will dispatch to the user Browser the URL the server respond came from, the user's browser name and the user's IP address:

```
<HTML><BODY>
<?php
  echo "Referer: " . $_SERVER["HTTP_HOST"] . "<BR>";
  echo "Browser: " . $_SERVER["HTTP_USER_AGENT"] . "<BR>";
  echo "User's IP address: " . $_SERVER["REMOTE_ADDR"];
?>
</BODY></HTML>
```

Save the above code snippet as a file **testServerVar.php** under the **Chap11_Cds** directory in the **sct.phptraining.com** domain. Run the file by invoking it via the Web Browser to get an output as shown in diagram 11.6.

Referer: sct.phptraining.com
Browser: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
User's IP address: 192.168.0.101

Diagram 11.6: Output for testServerVar.php.

To check the type of browser a client is using scan the Web server variable HTTP_USER_AGENT for the **user agent string** each browser sends as part of its HTTP request. The variable of interest right now is:

```
$_SERVER['HTTP_USER_AGENT']
```

REMINDER

The **$_SERVER** is a special reserved PHP variable that contains all Web server information. It is known as autoglobal (**or** superglobal). These special variables were introduced in PHP 4.1.0. Prior this older $HTTP_*_VARS arrays were used instead, such as **$**HTTP_SERVER_VARS. Although deprecated, the older variables still exist.

To display the contents of this variable, use:

```
<?php echo $_SERVER['HTTP_USER_AGENT']; ?>
```

A sample output of this script may be:

```
Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
```

OR

```
Mozilla/4.0 (compatible; Konqueror/3.0; Linux)
```

There are many types of variables available in PHP. In the above example an **array element was printed**.

The variable $_SERVER is just one variable that PHP automatically makes available.

A list can be seen in the Reserved Variables section of the PHP manual **or** a complete list can be obtained by running a PHP script like this: **(Refer diagram 11.7)**

```
<?php phpinfo(); ?>
```

When this PHP code spec executes on the Web server a well-formatted HTML page, full of information about PHP, will be displayed along with a list of all the Web server variables available.

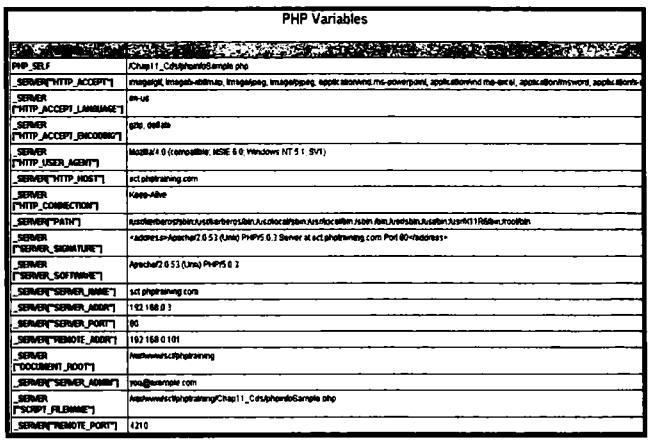


Diagram 11.7: The PHP variable section in the output for the function **phpinfo()**.

Multiple PHP statements can be put inside PHP tags creating little blocks of code that do more than just echo text. For example, **to check for Internet Explorer** and print a line of text based on a condition do this:

```
<?php
  if (strpos($_SERVER['HTTP_USER_AGENT'], 'Konqueror') !== false) {
      echo 'The Web Browser is Konqueror'; }
?>
```

A sample output of this script may be:

```
The Web Browser is Konqueror
```

The above code spec uses the **strpos()** function, which searches a character, word or string from within another string.

In this case **Konqueror** is being searched for inside $_SERVER['HTTP_USER_AGENT']. If found, the function returns the position of **Konqueror** relative to the start of the string. Otherwise, it returns FALSE.

If it **does not** return FALSE, (**i.e.** if expression evaluates to TRUE) the code within {braces} is executed. Otherwise, the code does not run.

To take this a step further, jump in and out of PHP mode in the middle of a PHP block. Store the following script in **testBrowser.php:**

```
<HTML>
  <HEAD><TITLE>Identifying The Web Browser</TITLE></HEAD>
  <BODY>
<?php
  if (strpos($_SERVER['HTTP_USER_AGENT'], 'Konqueror') !== false) {
?>
  <H3>The Web Browser is <B>Konqueror</B></H3>
  <CENTER>The function <B>strpos()</B> has returned <U>TRUE<U> for
          <B>Konqueror</B></CENTER>
<?php
  }
  elseif (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== false) {
?>
  <H3>The Web Browser is <U>Internet Explorer</U></H3>
  <CENTER>The function <B>strpos()</B> has returned <U>TRUE</U> for
          <B>MSIE</B></CENTER>
<?php
  }
  else {
?>
  <H3>The Web Browser is neither Konqueror nor Internet Explorer</H3>
```

```
  <CENTER>The function <B>strpos()</B> has returned <U>FALSE</U> for BOTH
          conditions</CENTER><BR>
  <B>Browser Type Is - </B>
<?php
     echo $_SERVER['HTTP_USER_AGENT'];
  }
?>
  </BODY>
</HTML>
```

Output: (Refer diagram 11.8.1, 11.8.2 and 11.8.3)

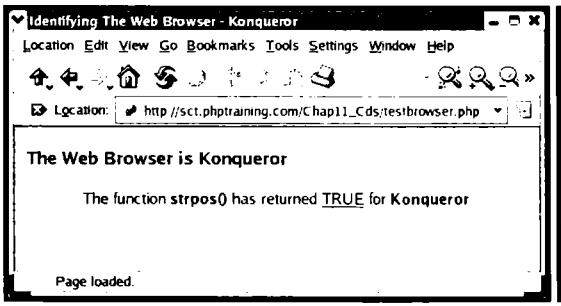


Diagram 11.8.1: Output in Konqueror.

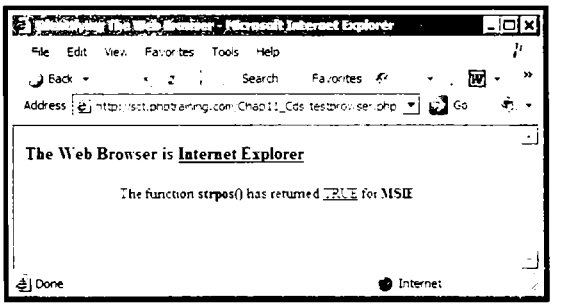


Diagram 11.8.2: Output in Internet Explorer.

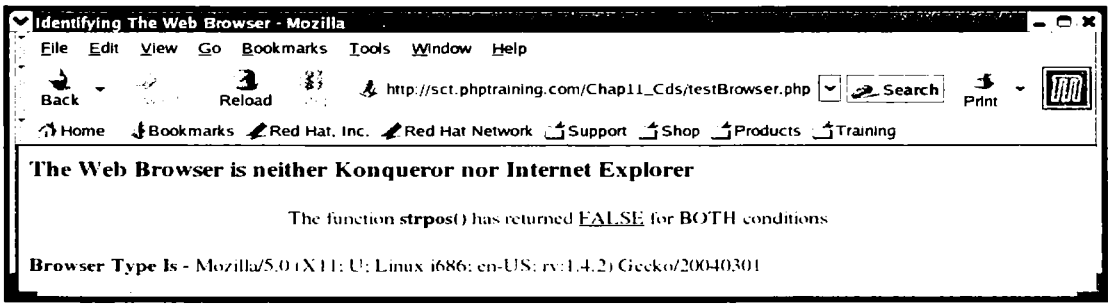


Diagram 11.8.3: Output in Web Browsers Other than Konqueror or Internet Explorer.

Instead of using the PHP **echo** statement to output something, the PHP code block was exited and straight HTML was sent to the browser. The important and powerful point to note here is that the logical flow of the PHP script continues to remain intact.

Functions For Variables

There are three basic functions that are useful while using variables. These are:

- **isset()** – Returns TRUE if a variable that is passed as its *only* parameter, has been initialized. If not then returns FALSE. Can take multiple arguments.

- If a variable has been unset with **unset()**, it will no longer be set in such a case isset() will return FALSE if testing a variable that has been set to NULL.
- Also note that a NULL byte ("\0") is not equivalent to the PHP NULL constant.
- isset() only works with variables, passing anything else will result in a parse error

Syntax:

```
bool isset (mixed <String> [, mixed <String> [, mixed ...]])
```

- **empty()** – Returns true if the variable, passed as its *only* parameter, has a false or empty values.
  - The following things are considered empty:
    - "" (an empty string)
    - 0 (0 as an integer)
    - "0" (0 as a string)
    - NULL
    - FALSE
    - array() (an empty array)
    - var $var; (a variable declared, but without a value in a class)

Syntax:

```
bool empty (mixed <String>)
```

- **unset()** – Removes or unsets an existing variable. It is used when a variable needs to be deleted. unset() destroys the specified variables.
  - The behavior of unset() inside of a function can vary depending on what type of variable you are attempting to destroy.
  - If a global variable is **unset()** inside of a function, only the local variable is destroyed. The variable in the calling environment will retain the same value as before unset() was called. If it is desired to unset() a global variable inside of a function, use the $GLOBALS statement while referring the variable.
  - If a variable that is PASSED BY REFERENCE is unset() inside of a function, only the local variable is destroyed. The variable in the calling environment will retain the same value as before unset() was called.
  - If a static variable is unset() inside of a function, unset() destroys the variable and all its references.

Syntax:

```
void unset (mixed <String> [, mixed <String> [, mixed ...]])
```

Example: varFunction.php

```
<?
  if(isset($test)) {   echo('The Variable Holds Value.'."\n"); }
  else {
      echo('The Variable Does Not Hold A Value.'."\n");
      $test = false;
      if(isset($test)) {
          echo('The Variable Now Holds A Value.'."\n");
          if(empty($test)) {
              echo('The Variable Now Holds FALSE.'."\n");
          }
          unset($test);
          if(!isset($test)) {
              echo('The   Variable
Is Being Deleted.'."\n");
          }
      }
  }
?>
```

Output: (Refer diagram 11.9)

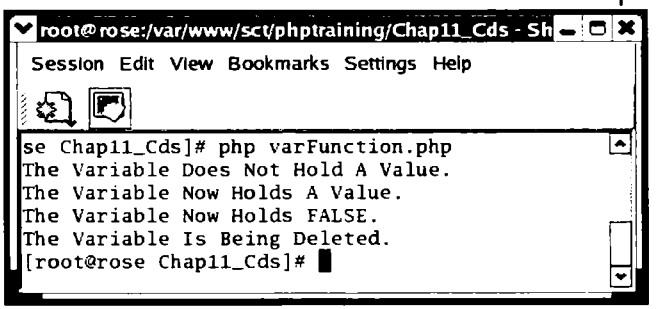


Diagram 11.9: Output for varFunction.php.

Controlling Script Execution

There are basically three functions, which can control the execution of a script:

- **exit()** – Used for immediately terminating the script execution. It accepts only one optional parameter, which acts as the script exit code. If it is a string, then it is printed out.

Syntax:

```
void exit ([mixed <Status>])
```

- **eval()** – Takes one string parameter, but it executes that string as if it were PHP code

Syntax:

```
mixed eval ([string <String>])
```

- **die()** – Used for immediately terminating the execution of the script. The function **die()** is an alias of the **exit()** function and works the same way.

Syntax:

```
void die ([mixed <Status>])
```

Example:

```
<?php
  $str = "exit()";
  eval($str);
?>
```

The script assigns **exit()** to **$str**, then passes $str into **eval()**. The **eval()** function will execute the string it gets as if it were PHP code, so the end result is that PHP runs the **exit()** function through **eval()** and terminates the script.

The function **exit()** is very common and is used wherever a PHP script must end with nothing further done.

The function **eval()** allows passing in any string as PHP code and have it executed. This allows storing PHP code in a database or runtime calculation, which gives lot more flexibility to the PHP code snippet.

REMINDER

Since the **exit()** function takes only one parameter, (either a program return value or a string) it is preferred to return a number and link this number with subsequent processing of the PHP script. In this case, **0** (zero) would usually mean, "Everything went OK" and everything else means, "Something went wrong".

Using **exit()** with a string causes PHP to output the string then terminate the script. This is commonly used by programmers with an exit() alias, such as:

```
perform_action() OR die("Action failed!");
```

In that situation, **perform_action()** will be called and, if it returns **false**, **die()** will be called to terminate the script. This is a fast and easy way to make sure that certain functions have run successfully before getting into the core of the PHP script.

REMINDER

The reason that code works is because the **OR** operator means that PHP will only execute the second function if the first function returned false. This is a common use of the OR operator.

WARNING

Remember that the || operator is higher than the **OR** operator in terms of precedence and, more importantly, || also higher than =. Consider the processing of the following code:

```
$processfile = fopen("filename", "r") || die("Could not open file!");
```

As || has a higher precedence than =, it is calculated first and PHP attempts to load *filename*. If it succeeds, **fopen()** will return a resource that evaluates to **true**. Naturally all calls to **die()** succeed, so **die()** will also always be evaluated as true.

However, because PHP will short-circuit the || operator, when **fopen()** succeeds **die()** will not be called and the || operator will return **1** (true) because the operand on its left side (the call to **fopen()**) returned true. Thus, PHP reads the line of code like this:

```
processfile = (fopen("somefile", "r") || die("Could not open file!"));
```

Finally, the value **1** will be assigned to **$processfile**. As this is not intended, the **OR** operator comes in handy. As **OR** has a lower priority than =, PHP reads the line of code like this:

```
$processfile = fopen("somefile", "r")) or die("Could not open file!");
```

That is, **fopen()** is called and its value assigned to **$processfile**, which is then put through the comparison operator.

Working With Date And Time

Being able to work easily with date and time information is a basic skill every PHP programmer should acquire as soon as possible.

REMINDER

PHP represents time as the number of seconds that have passed since January 1st 1970 00:00:00 GMT, a date known as the start of the Unix epoch, hence this date format is known as "epoch time" or a "Unix timestamp". This might seem like a peculiar way to store dates, but it is actually remarkably good as internally any date since 1970 can be stored as an integer and converted to a human-readable string wherever necessary. The advantage of working with Unix time is that it is not tied down to any specific formatting.

The Date() Function

The function **date()** is used to format either time or a date.

Syntax:

```
string date(string <Format> [, int <TimeStamp>])
```

This function returns a string formatted according to the specification. The table below shows the characters that can be used as format specifications:

Character	Description
a	"am" or "pm"
A	"AM" or "PM"
B	Swatch Internet time (000-999)
d	Day of the month with a leading zero (01-31)
D	Three characters that represents the day of the week (Mon-Sun)
F	The full name of the month (January-December)
g	The hour in 12-hour format without a leading zero (1-12)
G	The hour in 24-hour format without a leading zero (0-23)
h	The hour in 12-hour format with a leading zero (01-12)
H	The hour in 24-hour format with a leading zero (00-23)
i	The minutes with a leading zero (00-59)
I	"1" if the date is in daylights savings time, otherwise "0"
j	Day of the month without a leading zero (1-31)
l	The full name of the day (Monday-Sunday)
L	"1" if the year is a leap year, otherwise "0"
m	The month as a number with a leading zero (01-12)
M	Three letters that represents the name of the month (Jan-Dec)
n	The month as a number without a leading zero (1-12)
0	The difference to Greenwich time (GMT) in hours
R	An RFC 822 formatted date (e.g. "Tue, 10 Apr 2005 18:34:07 +0300")
s	The seconds with a leading zero (00-59)
S	The English ordinal suffix for the day of the month (st, nd, rd or th)
t	The number of days in the given month (28-31)
T	The local time zone (e.g. "GMT")
U	The number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)
w	The day of the week as a number (0-6, 0=Sunday)
W	ISO-8601 week number of year, weeks starting on Monday
y	The year as a 2-digit number (e.g. 03)
Y	The year as a 4-digit number (e.g. 2003)
z	The day of the year as a number (0-366)

Example: dateFunction.php

```
<?php
// Prints something like:
Saturday
  echo date("l")."\n";
// Prints something like:
Saturday 18th of June 2005
05:07:23 PM
  echo date("l dS of F Y h:i:s A")."\n";
```

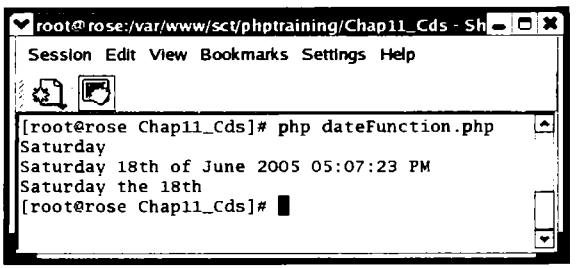


Diagram 11.10: Output for dateFunction.php.

```
// Prints something like: Saturday the 18th
  echo date("l \\t\h\e jS")."\n";
?>
```

Converting From A String

PHP has a built in function to help convert strings to a timestamp. The function **strtotime()** takes two parameters i.e. the string time to convert and a second optional parameter that can be a relative timestamp.

Syntax:

```
int strtotime(string <time> [, int now])
```

In this function the first parameter is important. Consider this script saved as **getUnixTime.php:**

```
<?php
  print strtotime("22nd June 2005")."\n";
  print strtotime("22 June 2005 17:30")."\n";
  print strtotime("2005/06/22")."\n";
?>
```

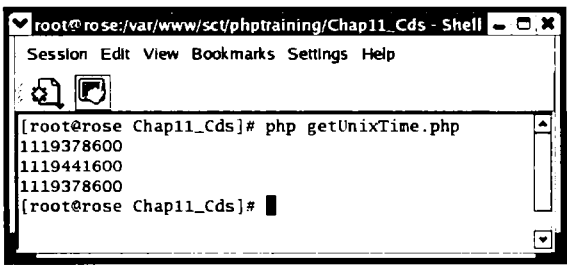


Diagram 11.11: Output for getUnixTime.php.

Here there are three ways of representing the same date, with the second option also including a time. If the script is executed, PHP outputs an integer representation for each one. Refer diagram 11.11. In the output, the first and third integers are the same, while the second one is slightly higher. These numbers are the Unix timestamps for the dates passed into **strtotime()**, which it has successfully managed to convert.

WARNING

The function **strtotime()** uses American-style dates, which means that if it finds a date like 10/11/2003, it will consider it to be October the 11th as opposed to November the 10th.

If PHP is unable to convert the string into a timestamp, it will return **-1**.

Example: (testDateConvert.php)

```
<?php
  $UnixDate1 = "Year End 2004";
  $UnixDate2 = "31st Dec. 2004";
```

```
  if (strtotime($UnixDate1) == -1) {
      print "Date conversion failed for $UnixDate1!\n";     }
  else {
      print "Date conversion succeeded for $UnixDate1!\n";       }
  if (strtotime($UnixDate2) == -1) {
      print "Date conversion failed for $UnixDate2!\n";     }
  else {
      print "Date conversion succeeded for $UnixDate2!\n";       }
?>
```

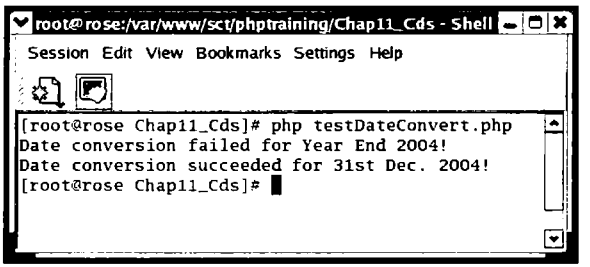


Diagram 11.12: Output for

Output: (Refer diagram 11.12)

In the above example, the **strtotime()** function is unable to resolve the string "Year End".

An optional second parameter for the **strtotime()** function is a timestamp used to provide relative dates. This is because the first parameter to **strtotime()**, the date string, can include relative dates such as "Next Sunday", "2 days", or "1 year ago". In this situation, PHP needs to know what to base these relative times on and this is where the second parameter comes in, any timestamp can be provided and PHP will calculate "Next Sunday" from that timestamp. If the second parameter is not provided and the first parameter is a relative time, PHP **assumes** that the reference is to the current time.

Example:(testTimeRef.php)

```
<?php
  print "Next Sunday is: " . date("l dS of F Y h:i:s A", strtotime("Next Sunday")) .
        "\n";
  print "Today is: " . date("l dS of F Y h:i:s A", strtotime("Today")) . "\n";
  print "Two days after the Day Before Yesterday is: " . date("l dS of F Y h:i:s A",
        strtotime("2 days", time() - (86400 * 2))) . "\n";
  print "A year ago is: " . date("l dS of F Y h:i:s A", strtotime("1 year ago",
        time())) . "\n";
?>
```

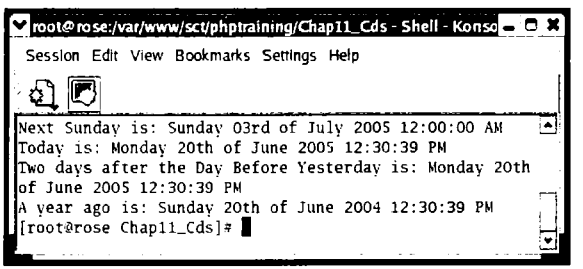


Diagram 11.13: Output for testTimeRef.php.

Output: (Refer diagram 11.13)

The first line will print the datetime for the next Sunday (not the upcoming Sunday, but the one after). The second line will print the current datetime. The third line actually uses **time()** minus two days as its second parameter and "2 days" for its first parameter, which means it returns the current datetime (two days time from now minus two days is now). The final example simply subtracts a year from the current datetime.

Converting From Components

The **mktime()** function is a special function for creating a Unix timestamp from a date component. It can be used to store year, month and day in separate variables.

Syntax:

```
int mktime ([int <hour> [, int <minute> [, int <second> [, int <month> [, int <day>
            [, int <year> [, int <is_dst>] ] ] ] ] ] ])
```

The order of the parameters in this function is as: hour (24Hr), minute, second, month, day, year, Is_Daylight_Savings_Time.

Example:
To use the **mktime()** function for passing 2:30 pm on the 30th of June 2005, the command is used like this:

```
$newTime = mktime(14, 30, 0, 6, 30, 2005, -1);
```

The last parameter (**i.e. -1**) tells PHP whether daylight savings time (DST) should be in effect. The options are:

- **1** to have DST switched ON
- **0** to have DST switched OFF
- **-1** to let PHP decide

Using the **mktime()** function provides performing date arithmetic, as it will correct crazy dates quite well.

If 13 months are to be added to the function call above without having to figure out the new settings, just add 13 to the month parameter (currently 6), as shown below:

```
$newTime = mktime(14, 30, 0, 19, 30, 2005, -1);
```

As a year consists of 12 months, PHP will add one to the year value, subtract 12 from the month's value, and calculate the date from there.

Example: (usemktime.php)

```
<?php
  $newTime1 = date("l dS of F Y h:i:s A", mktime(14, 30, 0, 6, 30, 2005, -1));
  $newTime2 = date("l dS of F Y h:i:s A", mktime(14, 30, 0, 19, 30, 2005, -1));

  print "Original Value: " . $newTime1 . "\n";
  print "Modified Value: " . $newTime2 . "\n";
?>
```

Output: (Refer diagram 11.14)

Performing Mathematical Operations

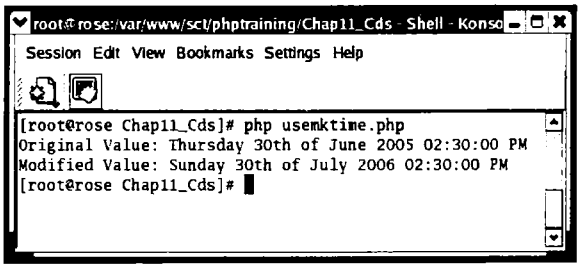


Diagram 11.14: Output for usemktime.php.

PHP has a great number of mathematical functions ready for use, as well as several constants for popular numbers to avoid re-calculations. In PHP, nearly all mathematical functions can be grouped into three areas: rounding, randomization and conversion.

Rounding Up A Number

There are three basic functions, used for rounding up numbers in PHP. These are:

- **ceil()** – It accepts one parameter, a number and returns the round to the nearest integer greater than the number.

Syntax:

```
float ceil (float <value>)
```

- **floor()** – It accepts one parameter, a number and returns the round to the nearest integer smaller than the number.

Syntax:

```
float floor (float <value>)
```

- **round()** – It accepts two parameters, a number and returns the round to the nearest integer smaller than the number.

Syntax:

```
float round (float <value> [, int <precision>])
```

REMINDER

The **floor()** function converts a floating-point number to an integer in the same way as typecasting, except typecasting is faster.

Example: testRounding.php

```
<?php
  $num1 = 4.9;
  print "Original value: ".$num1."\n";
```

```
    print "Ceiled value: ".ceil($num1)."\n";
    print "Floored value: ".floor($num1)."\n";
    print "Rounded value: ".round($num1)."\n";
?>
```

Output: (Refer diagram 11.15.1)

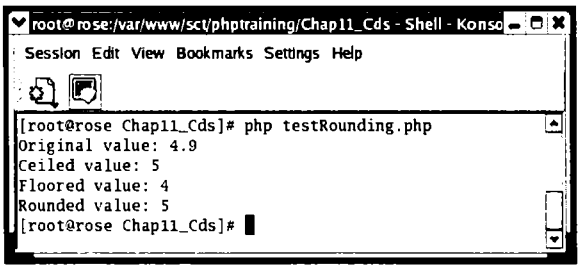


Diagram 11.15.1: Output for

Example: useRound.php

```
<?php
    $num1 = 4.45;
    $num2 = 4.5;
    $num3 = 4.95457;

    print "Round of $num1: ".round($num1)."\n";
    print "Round of $num2: ".round($num2)."\n";
    print "Round of $num3: ".round($num3)."\n";
    print "Round of $num3 with precision as 3: " .round($num3, 3)."\n";
    print "Round of $num3 with precision as 4: " .round($num3, 4)."\n";
?>
```

Output: (Refer diagram 11.15.2)

As can be seen, 4.45 is rounded up to 4, 4.5 to 5, while 4.95457 to 5. The fourth operation has the second parameter being used, which shows that it is very easy to round to a given number of decimal places.

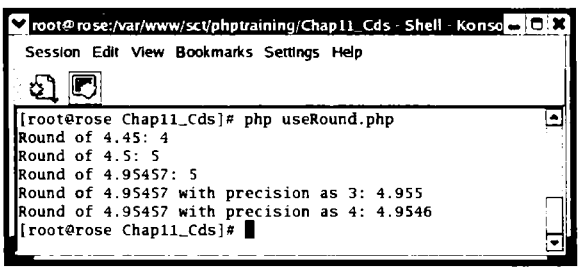


Diagram 11.15.2: Output for useRound.php.

Other Mathematical Conversion Functions

PHP provides several key functions that manipulate numbers in non-trigonometrical ways, of which the most important are:

- **abs()** – It is the most basic function and returns the absolute value of the parameter passed to it, (**i.e.** the function leaves positive values untouched, while it converts negative values into positive values).

Syntax:

```
number abs (number <value>)
```

- **sqrt()** – It is short for square root and takes just one parameter - the value whose square root needs to be calculated.

Syntax:

```
float sqrt (float <value>)
```

- **pow()** – It takes two parameters - a base and a power to raise it by. The value in the first parameter is multiplied with it self, one time less than the value entered as the second parameter. The pow() function is capable of handling negative powers for the second parameter, in which case it first multiplies the value in the first parameter with it self (as many times as one less the absolute value of the second parameter) and then use the result to divide the value in the first parameter.

Syntax:

```
float pow (number <base value> , number <exponent>])
```

- **hypot()** – It is designed to calculate the length of a vector (or the hypotenuse of a triangle). It takes two parameter: X and Y, and returns the value **sqrt((X * X) + (Y * Y))**.

Syntax:

```
float hypot (float <first side> , float <second side>])
```

Example: (testMaths.php)

```
<?php
   print "Absolute of 25.7: ".abs(25.7)."\n";
   print "Absolute of -4.1: ".abs(-4.1)."\n";
   print "Square Root of 25: ".sqrt(25)."\n";
   print "Square Root of 50: ".sqrt(50)."\n";
   print "10 to power of 3: ".pow(10, 3)."\n";
   print "10 to power of 5: ".pow(10, 5)."\n";
   print "10 to power of -2: ".pow(10, -2)."\n";
   print "10 to power of -4: ".pow(10, -4)."\n";
   print "100 to power of 0.5: ".pow(100, 0.5)."\n";
   print "Vector of 3 & 4: ".hypot(3, 4)."\n";
   print "Vector of 7 & 9: ".hypot(7, 9)."\n";
?>
```

WARNING

It is important to note that supplying 0.5 as the second parameter to **pow()** is the equivalent of calculating the square root of the first parameter, which means that using any negative number as parameter one and 0.5 as the second parameter will generate an error, because negative numbers have no square root.

Output: (Refer diagram 11.16)

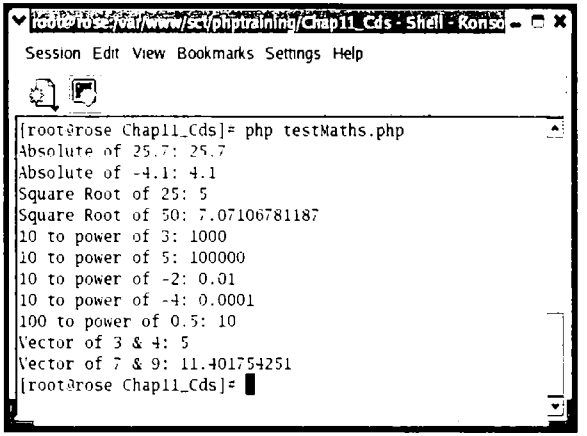


Diagram 11.16: Output for testMaths.php.

Working With String Functions

Learning how to manipulate strings is one of the more rewarding things to learn in PHP. There are many helpful functions in the language that can be used quickly and easily to allow an array of operations on user inputs.

As with variable names PHP considers strings to be case sensitive, which means matching and replacing strings would be quite tricky if it were not for the fact that PHP provides case-insensitive versions of each function where it would be helpful.

Extracting Part Of A String

The **substr()** function allows reading just part of a string and takes a minimum of two parameters, the string to work with and where to start reading from. There is an optional third parameter to allow specifying how many characters are to be read.

Syntax:

```
string substr ( string <source>, int <start position> [, int <length>])
```

REMINDER

In PHP, strings and arrays start at index **0** rather than **1**.

If the start position is larger than the actual length of the string, PHP will not return an error. It will just return an empty string.

A negative number can be specified as the third parameter. PHP will consider that number as the amount of characters that have to be **omitted from the end** of the string.

Similarly, a negative number can be specified as the second parameter. PHP will start reading from the string end. A negative length can be used with a negative start index.

Example: (testsubstr.php)
Here are some examples of basic usage of the **substr()** function:

```
<?php
  $str = "String Extraction";
```

```
   print "Start at 1: ".substr($str, 1)."\n";
   print "Start at 7: ".substr($str, 7)."\n";
   print "Start at 17: ".substr($str, 17)."\n";
   print "From 7 get 5: ".substr($str, 7, 5)."\n";
   print "From 2 get -11: ".substr($str, 2, -11)."\n";
   print "Start at -6: ".substr($str, -6)."\n";
?>
```

Output: (Refer diagram 11.17)

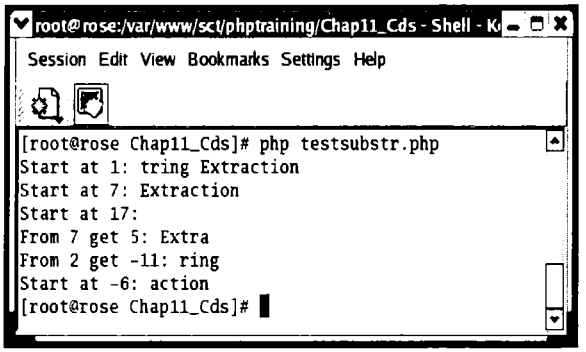


Diagram 11.17: Output for testsubstr.php.

Finding A String Within A String

PHP provides two functions to perform string search. The **strops()** function and its case-insensitive sibling the **stripos()** functions, returns the index of the first occurrence of a substring within a string.

Syntax:

```
int strpos ( string <source>, string <search text> [, int <start position>])
int stripos (string <source>, string <search text> [, int <start position>])
```

If a whole words is specified as the second parameter, the **strpos()** function returns the first position of that word within the string. If the substring specified in the second parameter is not found in first parameter, **strpos()** will return **false**.

Example: (teststrpos.php)

```
<?php
   $str = "String Extraction";
   print "Position of E: ".strpos($str, "E")."\n";
   print "Position of ing: ".strpos($str, "ing")."\n";
   print "Position of s: ".strpos($str, "s")."\n";
   print "Position of s: ".stripos($str, "s")."\n";
?>
```

Output: (Refer diagram 11.18.1)

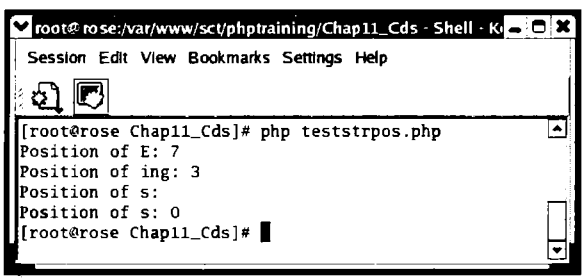


Diagram 11.18.1: Output for teststrpos.php.

WARNING

While using **strops()** in a conditional statement, to determine whether the substring exists or not, it is possible that the result fails even though the substring is found at **0** index. This is because, PHP considers **0** to be the same value as **false**, which means that the conditional statement cannot tell the difference between *Substring not found* and *Substring found at index 0*.

The problem encountered while using conditional statement can be resolved by using the **===** (is identical to) operator. In such situation, the comparison will be made between **0** of the type **false** (Boolean) *for substring not found*, rather than between **0** of the type integer *for substring found at index **0***.

Example: (strposCndt.php)

```
<?php
   $str = "Using strops() For String Extraction";
   $idx = strpos($str, "Using");
/* Using the 'comparison' (==) operator. */
   if ($idx == false) {
       print "The equation condition results a SUBSTRING NOT FOUND\n";   }
   else {
       print "The equation condition results a SUBSTRING FOUND at $idx\n"; }
/* Using the 'is identical to' (===) operator. */
   if ($idx === false) {
       print "The equation condition results a SUBSTRING NOT FOUND\n";   }
   else {
       print "The equation condition results a SUBSTRING FOUND at $idx\n"; }
?>
```

Output: (Refer diagram 11.18.2)

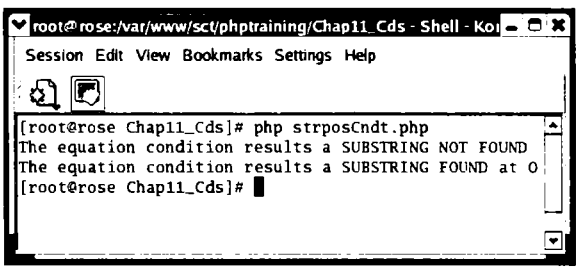


Diagram 11.18.2: Output for strposCndt.php.

Finally, the third parameter to the **strpos()** function allows to specify where to start the search from. Consider the next script, (saved as **strposStartAt.php**,) which tries to match the "ing" in "String":

```
<?php
   $str = "Using strops() For String Extraction";
   $idx1 = strpos($str, "ing");
   $idx2 = strpos($str, "ing", 15);
/* Using the strops() without the three parameter. */
   if ($idx1 === false) {
       print "First identical condition results a SUBSTRING NOT FOUND\n";   }
   else {
       print "First identical condition results a SUBSTRING FOUND at $idx1\n";
   }
```

```
/* Using the strops() with the three parameter. */
  if ($idx2 === false) {
      print "Second identical condition results a SUBSTRING NOT FOUND\n";
  }
  else {
      print "Second identical condition results a SUBSTRING FOUND at $idx2\n";
  }
?>
```

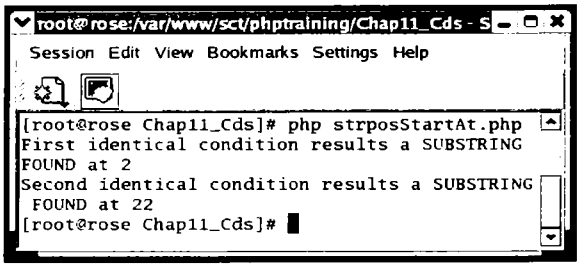


Diagram 11.18.3: Output for strposStartAt.php.

Output: (Refer diagram 11.18.3)

In the above example, the **strpos()** function matches the first "ing" it comes across, which will be in "Using". When the third parameter is specified, strpos() will ignore substring found before the specified starting point.

Returning The First Occurrence Of A String

This function extracts the substring or returns the first occurrence of a string within a string.

Syntax:

```
string strstr ( string <source>, string <search text>)
string stristr ( string <source>, string <search text>)
```

The **strstr()** function is case-insensitive subset of **stristr()** and finds the first occurrence of a substring (specified as the second parameter) inside another string (specified as the first parameter) and returns all characters from the first occurrence to the end of the string.

Example: (testStrSearch.php)
The following code snippet uses both **strstr()** and **stristr()** to scan a string for the word **WWW**. When found, it returns the text from the **WWW** until the end of the string:

```
<?php
  $str = "Solutions at -> http://www.sctindia.com/Chap_Cds/answers.php";
  $strURL1 = strstr($str, "WWW");
  $strURL2 = stristr($str, "WWW");
  echo "Original string: $str\n";
  echo "strstr() returns: $strURL1 \n";
  echo "stristr() returns: $strURL2 \n";
?>
```

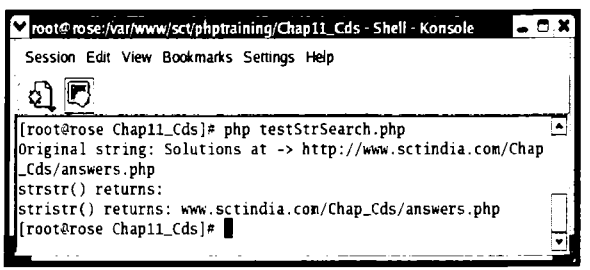


Diagram 11.19: Output for testStrSearch.php.

Output: (Refer diagram 11.19)

Replacing Parts Of A String

The **str_replace()** and the **str_ireplace()** functions replace one part of a string with new parts. The **str_ireplace()** function is case-insensitive.

Syntax:

```
mixed str_replace (mixed <search text>, mixed <replace text>, mixed
                   <source> [, int &<counter>] )
mixed str_ireplace (mixed <search text>, mixed <replace text>, mixed
                   <source> [, int &<counter>])
```

The function accepts a minimum of three parameters i.e. what to look for, what to replace it with and the string to work with. It also has an optional fourth parameter, which, if passed, will hold the number of replacements made.

Example: (strReplace.php)

```
<?php
  $str = "Smita will become Mrs. Smita after Smita's marriage.\n";
  $newStr = str_replace("Smita", "Chhaya", $str , $count);
  print "Original: ".$str;
  print "Modified: ".$newStr;
  print "$count changes were made.\n";
?>
```

Output: (Refer diagram 11.20)

In the above example, the new string is generated after three changes, as PHP will replace "Smita" with "Chhaya" three times.

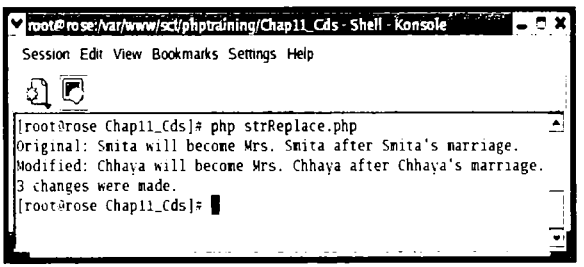


Diagram 11.20: Output for strReplace.php.

Converting To And From ASCII

The American Standard Code for Information Interchange is a special set of 255 numbers that evaluate to letters, symbols and actions used in most computers. For example, 74 is J, 106 is j, 123 is {, and 32 is a space.

Syntax:

```
string chr ( int <ASCII value>)
int ord ( string <text>)
```

The **chr()** function is used to convert ASCII to textual characters. This function accepts an ASCII value as its single parameter and returns the text equivalent, if exists.

The **ord()** function is used to convert textual characters to ASCII. This function accepts a string and returns the equivalent ASCII value.

Example: (strConvert.php)

```
<?php
   $str = "SHARANAM is the Project Analyst";
   if (ord($str{2}) == 65) {print "The third letter in the string is A\n";      }
   else {  print "The third letter in the string is not A\n";       }
   $ltr = chr(109);
   print "ASCII number 109 is equivalent to $ltr\n";
?>
```

Output: (Refer diagram 11.21)

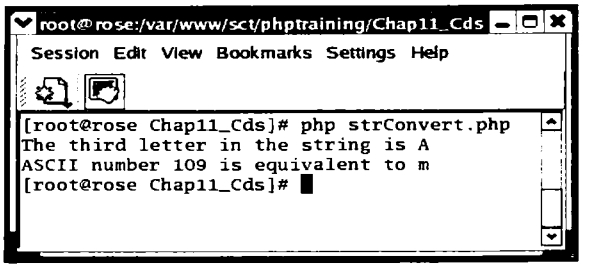


Diagram 11.21: Output for strConvert.php.

Trimming Whitespace

Trimming functions are used to strip whitespaces from left, right or either side of a string variable. Whitespaces include spaces, new lines and tabs.

There are three trimming functions:

- **trim()** – It removes whitespaces from either sides of the string passed as the first parameter.

Syntax:

```
string trim (string <source> [, string <charlist>])
```

- **ltrim()** – It removes whitespaces from the beginning of the string passed as the first parameter.

Syntax:

```
string ltrim (string <source> [, string <charlist>])
```

- **rtrim()** – It removes whitespaces from the end of the string passed as the first parameter.

Syntax:

```
string rtrim (string <source> [, string <charlist>])
```

The second optional parameter specifies other character(s), which need to be removed from the string.

An optional second parameter can also be passed to **trim()**, which should be a string specifying the characters that require to be trimmed.

Example: (strTrim.php)

```
<?php
  $a = " trimming ";
  echo "Original text: ".$a."\n";
  echo "LTrim returns: ".ltrim($a)."\n";
  echo "RTrim returns: ".rtrim($a)."\n";
  echo "Trim returns: ".trim($a)."\n";
  echo "Trim with 2 params returns: ".trim($a, " tng")."\n";
?>
```

Output: (Refer diagram 11.22)

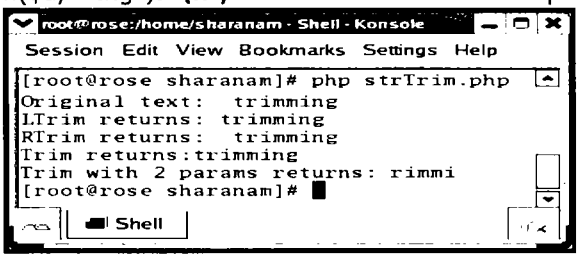


Diagram 11.22: Output for strTrim.php.

Wrapping Lines In Text Messages

When printing to a console as opposed to a web page, text does not wrap automatically. In such situations, it is best to wrap text.

Syntax:

```
string wordwrap (string <source> [, int <width> [, string <new line marker> [,
                 boolean <cut>]]])
```

When a sentence of text is passed to **wordwrap()** with no additional parameters, it will return that same string wrapped at the 75-character mark using "\n" for new lines. To override the defaults, specify the line size and new line marker as parameters two and three respectively.

Example: (strWordwrap.php)

```
<HTML><BODY>
<?php
  $text = "Sharanam, Hansel and
           Manisha are working on book
           called Application
           Development Using Oracle
           And PHP On Linux.";
  $text = wordwrap($text, 20,
           "<BR/>");
  print $text;
?>
</BODY></HTML>
```

Sharanam, Hansel and
Manisha are working
on book called
Application
Development Using
Oracle And PHP On
Linux.

Diagram 11.23: Output for strWordwrap.php.

When the script runs within a Web Browser, a page as shown in diagram 11.23 is displayed. While generating the output **wordwrap()** uses
, a HTML new line marker and split up words at the 20-character mark.

REMINDER

By default, the **wordwrap()** function wraps words pessimistically. If the second parameter is set to 20, **wordwrap()** will always wrap when it hits 20 characters or under. The only exception to this is if words are individually longer than 20 characters. In such situation the width of the line increases.

To enforce a specific line width, the value 1 is passed as the fourth parameter, which enables **cut** mode, (i.e. words over the limit will be broken up if this is enabled).

Changing String Case

The following functions affect the case of characters in strings:

- **strtoupper()** – It takes one string parameter and returns that string entirely in uppercase.

Syntax:

```
string strtoupper (string <source>)
```

- **strtolower()** – It takes one string parameter and returns that string entirely in lowercase.

Syntax:

```
string strtolower (string <source>)
```

- **ucfirst()** – It converts the first letter of every string to uppercase.

Syntax:

```
string ucfirst (string <source>)
```

- **ucwords()** – It converts the first letter of every word in the string to uppercase.

Syntax:

```
string ucwords (string <source>)
```

Example: (strChngCase.php)

```
<?php
  $str = "application development using ORACLE and PHP on LINUX\n";
  echo "Original: ".$str;
  echo "strtoupper(): ".strtoupper($str);
  echo "strtolower(): ".strtolower($str);
  echo "ucfirst(): ".ucfirst($str);
  echo "ucwords(): ".ucwords($str);
?>
```

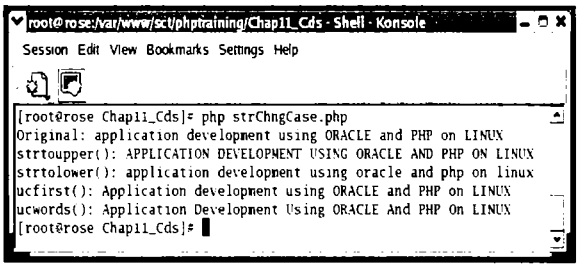


Diagram 11.24: Output for strChngCase.php.

Complex String Printing

The **printf()** function is the C and C++ way to format text and it is also available in PHP. It is not easy to use, but is useful for producing shorter code during code formatting.

Syntax:

```
void printf ( string <format> [, mixed <args> [, mixed ...])
```

The function takes a variable number of parameters - a format string is always the first parameter, followed by optional other parameters of various types.

Example: (useprintf.php)

```
<?php
  $name="Manisha";
  printf("Now %s is working in SCT.\n", $name);
?>
```

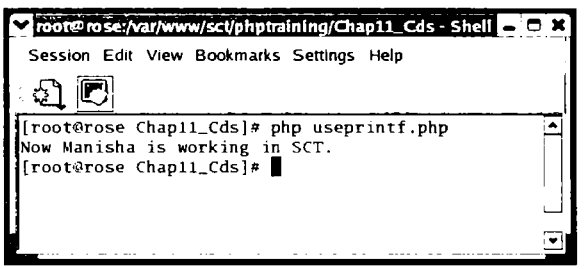


Diagram 11.25.1: Output for useprintf.php.

Output: (Refer diagram 11.25.1)

The **%s** is a special format string that means "string parameter to follow", which means that $name will be treated as text inside the string that **printf()** creates.

The **printf()** function takes a variety of other format strings other than **%s**. The complete list is as mentioned below:

Format	Meaning
%%	A literal percent character; no matching parameter is required
%b	Parameter is an integer; present it as binary
%c	Parameter is an integer; present it as a character with that ASCII value

Format	Meaning
%d	Parameter is an integer; present it as a signed number
%f	Parameter is a float; present it as a float
%o	Parameter is an integer; present it as octal
%s	Parameter is a string; present it as a string
%x	Parameter is an integer; present it as hexadecimal with lowercase letters
%X	Parameter is an integer; present it is hexadecimal with uppercase letters

Example: (printfSymbols.php)

```
<?php
   $number = 123;
   printf("123 in binary is: %b.\n", $number);
   printf("123 in float is: %f.\n", $number);
   printf("123 in hex is: %x.\n", $number);
   printf("123 as a string is: %s.\n", $number);
   printf("%% allows to print percent
characters.\n");
?>
```

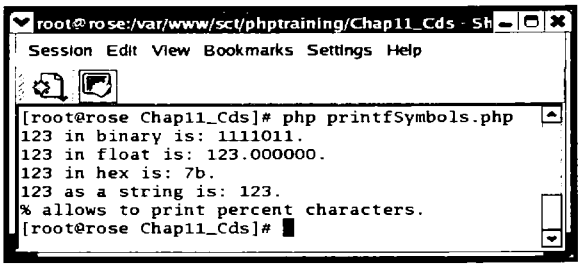


Diagram 11.25.2: Output for printfSymbols.php.

Output: (Refer diagram 11.25.2)

Pausing Script Execution

There are two ways to pause execution in a script.

Syntax:

```
   void sleep (int <seconds>)
   void usleep (int <micro_seconds>)
```

Example: (sleepFunction.php)

The most basic way of pausing execution is as follows:

```
<?php
   $now = time();
   print date("h:i:s A")."\n";
   echo "Pause.\n";
   while ($now + 4 > time()) {
      // No Action
   }
   echo "Resume.\n";
   print date("h:i:s A")."\n";
?>
```

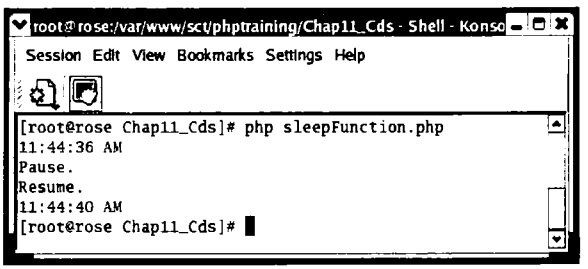


Diagram 11.26.1: Output for sleepFunction.php.

While it does work, there are two problems with it. Firstly, **time()** has a very low precision, only returning the number of whole seconds that have passed, which makes the whole thing quite vague. Secondly, PHP has to sit there looping thousands of times while it waits, essentially doing nothing. A much better solution is to use the one of the two script sleep functions, **sleep()** and **usleep()**, which take the amount of time to pause execution as their only parameter.

The difference between **sleep()** and **usleep()** is that **sleep()** takes a number of seconds as its parameter, whereas **usleep()** takes a number of microseconds i.e. millionths of a second as its parameter. Using either of them is far more accurate than the previous **time()** loop and they both have their advantages. The **sleep()** function is better if accuracy is not desired and **usleep()** is better if accuracy is desired.

The above script could be rewritten like this (**sleep.php**):

```
<?php
   $now = time();
   print date("h:i:s A")."\n";
   echo "Pause.\n";
   sleep(4);
   echo "Resume.\n";
   print date("h:i:s A")."\n";
?>
```

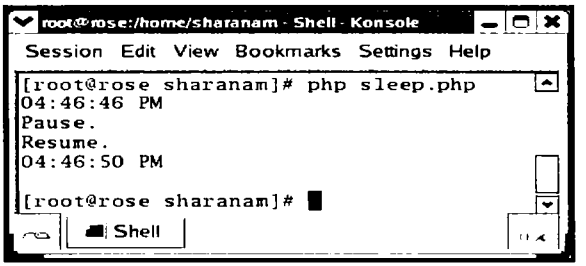


Diagram 11.26.2: Output for sleep.php.

On similar ground, the objective of the above script can be achieved with the use of the **usleep()** function. The script (**usleep.php**) mentioned below demonstrates this:

```
<?php
   $now = time();
   print date("h:i:s A")."\n";
   echo "Pause.\n";
   usleep(4000000);
   echo "Resume.\n";
   print date("h:i:s A")."\n";
?>
```

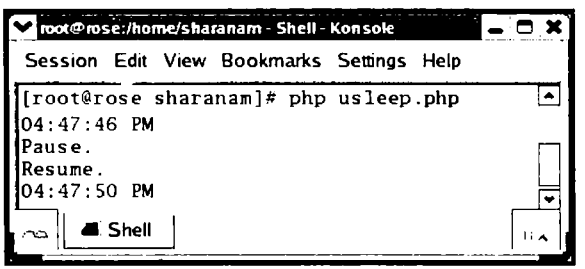


Diagram 11.26.3: Output for usleep.php.

REMINDER

The default maximum script execution time is 30 seconds, but **sleep()** and **usleep()** can be used to make the scripts go on for longer than that because technically PHP does not have control during the sleep operation.

WARNING

The use of **usleep()** is a big no-no if backwards compatibility is desired, because it wasn't available on Windows prior to PHP 5.

SECTION III: WORKING WITH PHP

Working With Web Pages

When building an interactive web site, sooner or later a form will be needed. There are a number of different forms that can be used to capture (or display) information. Some forms require information to be entered using the keyboard others require selecting one or more choices from a drop down list of values by clicking with a mouse. Yet others may simply involve a hidden form value that is embedded in the form itself, which is **not intended** to be modified by the user.

Forms are an elegant way of gathering data from visitors. Forms can be as small as a login form or as large as a membership subscription form, which asks for lots of information. Obviously, the bigger the form, the more information there is to process and thus greater the challenge.

It is possible to have **multiple** forms on the same HTML page, so there must be some way to distinguish one form from the other. Furthermore, there must be a way to tell the client's browser what to do once the user initiates some form action (usually by clicking a button). Both of these needs are taken care of by enclosing each form entities as follows:

```
<FORM Action="some_action" Method="post">
    ... form entities ...
</FORM>
```

Two important elements make up this enclosure:

- **Action** - The value passed to the keyword **Action** specifies what script should process the form on the Web server.
- **Method** - The method specifies how the Browser will send the data captured by the form to the script on the Web server. There are two possible methods:
  - The **get** method sends all of the form information encoded in the URL to the Web server. This method is rarely used due to various length and language restrictions.
  - The **post** method sends all of the form information in the request body. This method is **usually preferred** over the **get** method for dispatching form information back to the Web server.

A powerful feature of PHP is the way it handles HTML forms. The basic concept is that any element in an HTML form will automatically be available to run PHP scripts at the Web server.

Here is an example of a basic HTML form:

```
<FORM Action="action.php" Method="post">
    Your name: <INPUT Type="text" Name="name">
    Your age: <INPUT Type="text" Name="age">
    <INPUT Type="submit" Value="Submit">
</FORM>
```

This is a simple HTML form with no special tags of any kind. When a user fills the form and hits the submit button, the data captured by the form elements is dispatched to the action.php by the client's browser. action.php is located and runs at the Web server which served the HTML form to the client browser in the first place.

Note down the following points while designing a form:

- Always include the MAXLENGTH attribute in text fields to limit the size of a field. This could save a script from throwing up unexpected errors
- If a field's value is to be displayed without allowing the user to change it, set its READONLY boolean attribute to 1. Additionally, set it's TABINDEX attribute's value to –1

Example:

```
<INPUT Type="text" Name="price" READONLY Tabindex="-1">
```

The code spec in action.php could be:

```
Hi <?php echo $_POST['name']; ?>.
You are <?php echo $_POST['age']; ?> years old.
```

A sample output of this script may be:

```
Hi Sharanam. You are 24 years old.
```

The $_POST['name'] and $_POST['age'] variables are automatically set by PHP. In the above codespec, the **$_POST** autoglobal variable contains **all** POST data that has been introduced. **Notice** the method in the HTML form example is POST, if the method in the HTML form was GET then the form information would be contained in the **$_GET** autoglobal instead.

The **$_REQUEST** autoglobal variable can also be used, if the source of the requested data is unknown. The $_REQUEST autoglobal variable contains the merged information of GET, POST and COOKIE data.

REMINDER

In the above code spec, it is possible to use $name as the variable name instead of $_POST['name']. Having said that, this method of accessing variables has a problem if the web site hosting company has turned off **register_globals**. With this setting to OFF, PHP returns an **empty value**.

So, using **$_POST['variable']** is the best option. However, nothing works if the web site, hosting company puts off the **track_vars** option in the php.ini file.

HINT

According to the HTTP specifications, the **POST** method should be used when form data is being used to **change the state** of a Web server's autoglobal variables.

For example, if a page has a form to allow users to pass their comments to the web master, the form should use POST. If **Reload** or **Refresh** is clicked on a page that is reached through a POST, it almost always is an error. This means that same comment will not be posted twice, which is why these pages are not book marked or cached.

Use the GET method with an HTML form when the form provides something to the Web server, but does not actually change anything at the Web server. One disadvantage of the POST method is that the URL **cannot** be book marked.

WARNING

Apache server logs the complete URL including form contents in a file that anyone can read, so be sure, **never to include** a password in a form that uses GET to return data back to the Web server.

Designing A Login Module In HTML And PHP

The function of a Login module is to authenticate visitors to the Web Site. In PHP, the simplest way to build a Login module is to collect the Login information via an HTML form and then pass the information to a PHP page for authentication.

Building An HTML Form

In general, Login information consists of two parts, namely username and password. This information is demanded by a Login Page containing the HTML form object. The purpose of the HTML Form is to capture login information with the help of these fields (in this case textboxes).

Layout For The Login Page

As the form in the Login page contains two fields, the layout can be either horizontal (i.e. side-by-side) or vertical (i.e. one below the other). The horizontal layout is preferred when the height of the form is restricted, but it has sufficient width. The vertical layout is preferred when the form has to cover a large area of the page.

Apart from two fields, (used for capturing Login and its associated Password information) the form holds a button, which is clicked to allow the page to Submit captured information back to the Web server.

Optionally, the HTML page displays a small description about the function of the page (i.e. general Login help for a visitor). Sometimes the page contains a link (labeled "Forgot Password"), which allows a registered visitor to get a new password from the Web site's administrator.

Based on the above discussion, the Login page should be designed as an HTML page. The file containing the HTML code, used for rendering the Login page, will be named **login.html**.

Use any ASCII editor to generate the **login.html** file. The contents of file will be as follows:

```
<HTML>
<!-- Text displayed in the browser's Title bar. -->
<HEAD><TITLE>Visitor's Login Page</TITLE></HEAD>
<BODY>
<!-- The table containing the Top bar for the page. -->
  <TABLE Border="0" BgColor="#FFFFFF" CellPadding="0" CellSpacing="0"
         Width="600"><TR>
  <!-- A empty column to make the page presentable. -->
     <TD Width="356"> </TD>
  <!-- The column containing a Logo, to make the page
  presentable. -->
     <TD Align="right" VAlign="top">
        <IMG Height="50" Src="/images/Sctonly2.gif" Width="202">
        <IMG Height="1" Src="/images/pixel.gif" Width="25">
     </TD>
  </TR><TR>
  <!-- The column containing a Header for the Login Form. -->
     <TD ColSpan="2">
        <TABLE Width="100%" Border="0" CellSpacing="0"
               CellPadding="0"><TR>
           <TD BgColor="#FFFFFF" Width="14"> </TD>
           <TD BgColor="#FFFFFF" Width="150"><H2>Members
               Login</H2></TD>
           <TD><IMG Height="1" Src="/images/line.gif" Width="410"></TD>
        </TR></TABLE>
     </TD>
  </TR><TR>
     <TD ColSpan="2">
        <TABLE Border="0" CellPadding="0" CellSpacing="0" Width="593"><TR>
        <!-- An empty column to make the page presentable. -->
           <TD Width="14"> </TD>
           <TD>
```

```
  <!-- Designing the layout of the form, which will capture the
  user name and password. -->
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Height="300"
       Width="100%"><TR>
  <TD Width="8%"><BR></TD>
  <TD VAlign="top" Width="75%"><BR>
<!-- Form codespec begins. -->
  <FORM Name="frmAuthLogin">
  <!-- The Login message displayed as a guideline to
  visitors. -->
     <CENTER>Please enter the <B>LOGIN</B> information to get access to the
             resources available<BR></CENTER>
     <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"><TR>
     <!-- An empty row to make the page presentable. -->
        <TD ColSpan="2"><IMG Height="10" Src="/images/pixel.gif"
             Width="1"></TD>
     </TR><TR>
     <!-- A row which display the form title. -->
        <TD ColSpan=2><B>Login Information</B></TD>
     </TR><TR>
     <!-- A row for capturing Username. -->
        <TD Align="right">Username:</TD>
        <TD Align="left"><INPUT maxLength="15" Name="txtUserName"
             Type="text"></TD>
     </TR><TR>
     <!-- A row for capturing password. -->
        <TD Align="right">Password:</TD>
        <TD Align="left">
           <INPUT maxLength="15" Name="txtPassword"
                Type="password"></TD>
     </TR><TR>
     <!-- An empty row to make the page presentable. -->
        <TD Colspan="2"><IMG Height="10" Src="/images/pixel.gif"
             Width="1"></TD>
     </TR><TR>
     <!-- A row for submit button. -->
        <TD Align="right" ColSpan="2">
           <INPUT Name="cmdSubmit" Type="submit" Value="Login"></TD>
     </TR><TR>
     <!-- An empty row to make the page presentable. -->
        <TD Colspan="2"><IMG Height="10" Src="/images/pixel.gif"
             Width="1"></TD>
     </TR><TR>
  <!-- The row containing the link to contact the site
  administrator. -->
        <TD Align="center"><A HRef="mailto:admin@sctindia.com">Forgot
             Password</A></TD>
     </TR></TABLE>
  </FORM>
```

```
      </TD><TD VAlign="top" Width="15%"><BR></TD>
  </TR>
</TABLE>
<!-- Final code corresponding to the template begins. -->
            </TD>
        <!-- A empty row to make the page presentable. -->
            <TD Width="14"> </TD>
        </TR></TABLE>
      </TD>
  </TR></TABLE>
</BODY>
</HTML>
```

Alternatively, this file can be found in the accompanying CD-ROM. It is located at:

BookCodes/Chap12_Cds/login.html

When the file (login.html) runs in a browser, the Login page is displayed as shown in diagram 12.1.

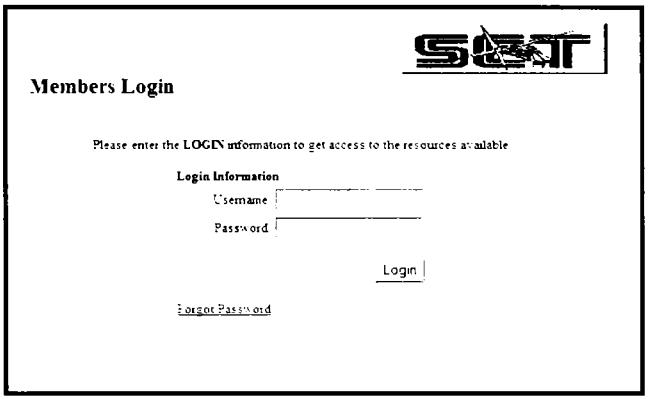


Diagram 12.1: Output for login.html.

The above code places the following objects on the Login page:

- A form object, named **frmAuthLogin**. This acts as a container for holding the data fields and the button
- A field for capturing user name, named **txtUserName**. It has a label **Username** placed before it
- A field for capturing password, named **txtPassword**. It has a label **Password** placed before it
- A Submit button, named **cmdSubmit**. It is labeled **Login**
- A link labeled **Forgot Password**. This provides an option to request a new password from the site's administrator

Validating The Information Captured

The Login form created above is incomplete, as it addresses only to the aesthetic needs of the page. The page may capture login information but it does nothing with it. The real purpose of the page is to capture, validate and submit information.

The function of capturing information has been handled, now its time to validate the information captured. Since the Login page is an HTML form and HTML being static, a different Client-side scripting language is used to validate information captured by this page.

There are many such client-side scripting languages like JavaScript, VBScript and so on. In this material the client-side scripting language used is JavaScript.

The **<SCRIPT>** tag is used to embed client-side scripting into the HTML page. The JavaScript (sandwiched between the **<SCRIPT>** and **</SCRIPT>** tags) can be placed anywhere within the HEAD section **or** the BODY section of the HTML page. *This illustration places the script in the BODY section.*

To understand the technique of adding client-side scripting to the Login page, open the file **login.html** in an ASCII editor and paste the following script (placed between horizontal lines) immediately after the <BODY> tag:

```
<HTML>
<!-- Text displayed in the browser's Title bar. -->
<HEAD><TITLE>Visitor's Login Page</TITLE></HEAD>
<BODY>
  <!-- Java script code-spec follows to validate the username and
  the password captured by this page.  -->
  <SCRIPT Language="JavaScript">
  /* Generating the function used for trimming string values. */
     function strtrim() {
     /* Returns the string passed as a parameter after removing
     blank spaces, if any, at both the beginning and the end of
     the parameter. */
        return this.replace(/^\s+/,'').replace(/\s+$/,'');
     }

  /* Generating a global alias name for the function used for
  trimming string values. */
     String.prototype.trim = strtrim;

  /* Generating the function used to validate the username and
  the password captured by this page. */
     function frmValidate() {
     /* A variable holding a reference to the form object
     initialized in this page. */
        var frm = document.frmAuthLogin;
     /* A variable holding the username (after trimmed for
     spaces) captured by this page. */
        var username = frm.txtUserName.value.trim();
     /* A variable holding the password (after trimmed for
     spaces) captured by this page. */
        var password = frm.txtPassword.value.trim();
     /* A check made for empty username or the size of the
     username is less that a 5 letter word. */
        if (username == "" || username.length < 5) {
        /* If either one of the above conditions return TRUE, the
        username is considered invalid. A message is displayed,
        which indicates the error. */
           alert("Please enter a username of minimum 5 characters.");
        /* Placing the form cursor on the username text box. */
           frm.txtUserName.focus();
```

```
        /* Setting an indication that the information captured
        cannot be submitted. */
            return false;
        }
    /* When the condition for username is satisfactory, a check
    is made for empty password or the size of the password is
    less that a 6 letter word. */
        else if (password == "" || password.length < 6) {
        /* If either one of the above conditions return TRUE, the
        password is considered invalid. A message is displayed,
        which indicates the error. */
            alert("Please enter a password of minimum 6 characters.");
        /* Placing the form cursor on the password text box. */
            frm.txtPassword.focus();
        /* Setting an indication that the information captured
        cannot be submitted. */
            return false;
        }
    /* When the conditions for username and password are
    satisfactory, setting an indication that the information
    captured can be submitted. */
        else { return true; }
    }
</SCRIPT>
<!-- The table containing the Top bar for the page. -->
  <TABLE Border="0" BgColor="#FFFFFF" CellPadding="0" CellSpacing="0" Width="600"><TR>
  <!-- A empty column to make the page presentable. -->
      <TD Width="356"> </TD>
```

Even though the above script exists in the file **login.html**, the HTML page needs to be informed as to when the script should be executed. In this case, the best event to execute the above script is when Login is clicked. To incorporate this, add the **onSubmit** attribute to the <FORM> tag. The <FORM> tag will now read as follows:

```
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Height="300" Width="100%"><TR>
  <TD Width="8%"><BR></TD>
  <TD VAlign="top" Width="75%"><BR>
<!-- Form codespec begins. -->
  <FORM Name="frmAuthLogin" onSubmit="return frmValidate();">
  <!-- The Login message displayed as a guideline to visitors. -->
      <CENTER>Please enter the <B>LOGIN</B> information to get access to the resources
available<BR></CENTER>
```

Running the **login.html** file in a web browser will display a page, which looks similar to the previous page rendered. The difference is observed when Login is clicked. Now, if the username or password fields capture invalid information, (i.e. length of value entered in the username field is less than five and length of value entered in the password field is less than six) an appropriate error message is displayed. Refer diagrams 12.2.1 and 12.2.2.

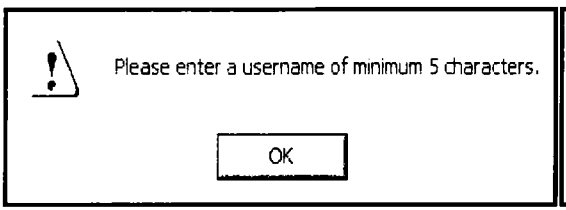


Diagram 12.2.1: Message for invalid username.

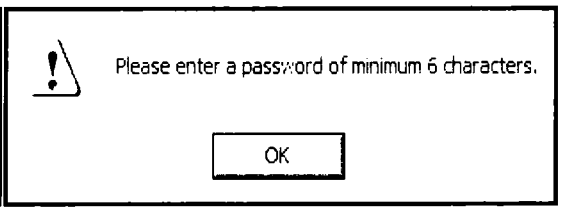


Diagram 12.2.2: Message for invalid password.

Modularizing The HTML Page

Before building the PHP page that will respond to the call made by the **Action** attribute, the HTML page can be enhanced to make it more sophisticated.

HTML allows isolation of client side-scripts, the script can be external to the HTML file. Bearing this in mind the next attempt would be to isolate the JavaScript functions from the HTML file.

To understand the technique of separating client-side scripting from the Login page, open the file **login.html** in an ASCII editor. Extract the JavaScript code block mentioned between the <SCRIPT> and </SCRIPT> tags. Create a file named **loginscript.js** and paste into that file.

Finally, modify the <SCRIPT> tag (placed between horizontal lines) in the login.html file as shown below:

```
<HTML>
<!-- Text displayed in the browser's Title bar. -->
<HEAD><TITLE>Visitor's Login Page</TITLE></HEAD>
<BODY>
```

```
  <!-- Java script code-spec follows to validate the username and
  the password captured by this page.  -->
  <SCRIPT Language="JavaScript" Src="loginscript.js" Type="text/javascript">
```

```
  </SCRIPT>
<!-- The table containing the Top bar for the page. -->
  <TABLE Border="0" BgColor="#FFFFFF" CellPadding="0" CellSpacing="0" Width="600"><TR>
  <!-- A empty column to make the page presentable. -->
      <TD Width="356"> </TD>
```

REMINDER

In the above **<SCRIPT>** tag, the value of the **Src** attribute holds the site's relative path to locate the JavaScript file. For the purpose of this illustration, the JavaScript file is placed under the **Chap12_Cds** directory present under the **sct.phptraining.com** domain.

The contents of **loginscript.js** will be as:

```
/* Generating the function used for trimming string values. */
    function strtrim() {
```

```
    /* Returns the string passed as a parameter after removing blank spaces, if
    any, at both the beginning and the end of the parameter. */
        return this.replace(/^\s+/,'').replace(/\s+$/,'');
    }
/* Generating a global alias name for the function used for trimming string
values. */
    String.prototype.trim = strtrim;
/* Generating the function used to validate the username and the password
captured by this page. */
    function frmValidate() {
    /* A variable holding a reference to the form object initialized in this
    page. */
        var frm = document.frmAuthLogin;
    /* A variable holding the username (after trimmed for spaces) captured by
    this page. */
        var username = frm.txtUserName.value.trim();
    /* A variable holding the password (after trimmed for spaces) captured by
    this page. */
        var password = frm.txtPassword.value.trim();
    /* A check made for empty username or the size of the username is less that
    a 5 letter word. */
        if (username == "" || username.length < 5) {
        /* If either one of the above conditions return TRUE, the username is
        considered invalid. A message is displayed, which indicates the error.
        */
            alert("Please enter a username of minimum 5 characters.");
        /* Placing the form cursor on the username text box. */
            frm.txtUserName.focus();
        /* Setting an indication that the information captured cannot be
        submitted. */
            return false;
        }
    /* When the condition for username is satisfactory, a check is made for
    empty password or the size of the password is less that a 6 letter word. */
        else if (password == "" || password.length < 6) {
        /* If either one of the above conditions return TRUE, the password is
        considered invalid. A message is displayed, which indicates the error.
        */
            alert("Please enter a password of minimum 6 characters.");
        /* Placing the form cursor on the password text box. */
            frm.txtPassword.focus();
    /* Setting an indication that the information captured cannot be submitted.
    */
            return false;
        }
    /* When the conditions for username and password are satisfactory, setting
    an indication that the information captured can be submitted. */
        else {    return true;  }
    }
```

Passing Values To A PHP Page

The final step in this illustration is to identify the PHP page that will authenticate the information captured by the HTML form. Adding the **Action** attribute to the <FORM> tag does this identification. Additionally, the **Method** attribute of the <FORM> tag is also specified. This specifies the method used to submit (i.e. GET or POST) the form's information back to the Web server.

Edit the **login.html** file and ensure that the <FORM> tag appears as shown below:

```
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Height="300" Width="100%"><TR>
   <TD Width="8%"><BR></TD>
   <TD VAlign="top" Width="75%"><BR>
<!-- Form codespec begins. -->
```

```
   <FORM Action="logincheck.php" Method="post" Name="frmAuthLogin"
         onSubmit="return frmValidate();">
```

```
   <!-- The Login message displayed as a guideline to visitors. -->
       <CENTER>Please enter the <B>LOGIN</B> information to get access to the resources
available<BR></CENTER>
```

After incorporating all the changes the contents of the **login.html** page will appear as show below:

```
<HTML>
<HEAD><TITLE>Visitor's Login Page</TITLE></HEAD>
<BODY>
<!-- Java script code-spec follows to validate the username and
the password captured by this page.  -->
  <SCRIPT Language="JavaScript" Src="loginscript.js" Type="text/javascript">
  </SCRIPT>
<!-- The table containing the Top bar for the page. -->
<TABLE Border="0" BgColor="#FFFFFF" CellPadding="0" CellSpacing="0"
       Width="600"><TR>
     <TD Width="356"> </TD>
  <!--  The column containing a Logo, to make the page
  presentable. -->
     <TD Align="right" VAlign="top">
        <IMG Height="50" Src="/images/Sctonly2.gif" Width="202">
        <IMG Height="1" Src="/images/pixel.gif" Width="25">
     </TD>
  </TR><TR>
  <!-- The column containing a Header for the Login Form. -->
     <TD ColSpan="2">
        <TABLE Width="100%" Border="0" CellSpacing="0"
               CellPadding="0"><TR>
           <TD BgColor="#FFFFFF" Width="14"> </TD>
           <TD BgColor="#FFFFFF" Width="150"><H2>Members
               Login</H2></TD>
           <TD><IMG Height="1" Src="/images/line.gif" Width="410"></TD>
        </TR></TABLE>
     </TD>
  </TR><TR>
     <TD ColSpan="2">
        <TABLE Border="0" CellPadding="0" CellSpacing="0" Width="593"><TR>
           <TD Width="14"> </TD>
           <TD>
```

```
<!-- Designing the layout of the form, which will capture the
user name and password. The data captured will be submitted to
the logincheck.php page. A JavaScript based validation is
performed on the captured information before it is submitted. -->
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Height="300"
        Width="100%"><TR>
  <TD Width="8%"><BR></TD>
  <TD VAlign="top" Width="75%"><BR>
<!-- Form codepsec begins. -->
  <FORM Action="logincheck.php" Method="post" Name="frmAuthLogin"
        onSubmit="return frmValidate();">
  <!-- The Login message displayed as a guideline to visitors.-->
     <CENTER>Please enter the <B>LOGIN</B> information to get access to the
              resources available<BR></CENTER>
     <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"><TR>
        <TD ColSpan="2"><IMG Height="10" Src="/images/pixel.gif"
             Width="1"></TD>
  </TR><TR>
     <!-- A row which display the form title. -->
        <TD ColSpan=2><B>Login Information</B></TD>
     </TR><TR>
     <!-- A row for capturing Username. -->
        <TD Align="right">Username:</TD>
        <TD Align="left"><INPUT maxLength="15" Name="txtUserName"
             Type="text"></TD>
     </TR><TR>
     <!-- A row for capturing password. -->
        <TD Align="right">Password:</TD>
        <TD Align="left">
           <INPUT maxLength="15" Name="txtPassword"
                Type="password"></TD>
     </TR><TR>
        <TD Colspan="2"><IMG Height="10" Src="/images/pixel.gif"
             Width="1"></TD>
     </TR><TR>
     <!-- A row for submit button. -->
        <TD Align="right" ColSpan="2">
           <INPUT Name="cmdSubmit" Type="submit" Value="Login"></TD>
     </TR><TR>
        <TD Colspan="2"><IMG Height="10" Src="/images/pixel.gif"
             Width="1"></TD>
     </TR><TR>
     <!-- The row containing the link to contact the site
     administrator. -->
        <TD Align="center"><A HRef="mailto:admin@sctindia.com">Forgot
             Password</A></TD>
     </TR></TABLE>
</FORM>
```

```
    </TD><TD VAlign="top" Width="15%"><BR></TD>
  </TR>
</TABLE>
<!-- Final code corresponding to the template begins. -->
          </TD>
       <!-- A empty row to make the page presentable. -->
          <TD Width="14"> </TD>
       </TR></TABLE>
    </TD>
  </TR></TABLE>
</BODY>
</HTML>
```

Finally, test the **login.html** page by calling it in a Web browser

Building A PHP Page

Before building this PHP page, edit the php.ini file and switch on the **register_globals** attribute. This is because the code spec used in this material, uses or accesses values held by form variables, without the use of the global variable $_POST[]. This means that a form field named txtUserName will be used or accessed as $txtUserName instead of **$_POST['txtUserName']**.

Changes in php.ini

Edit the **php.ini** file available under **/usr/local/lib/** to allow the registration of global variables. Global variables are extensively used in this material. The Registering Globals feature in Apache Web server is usually disabled and hence needs to be activated by switching the **register_globals** to **ON**. Refer diagram 12.3.0

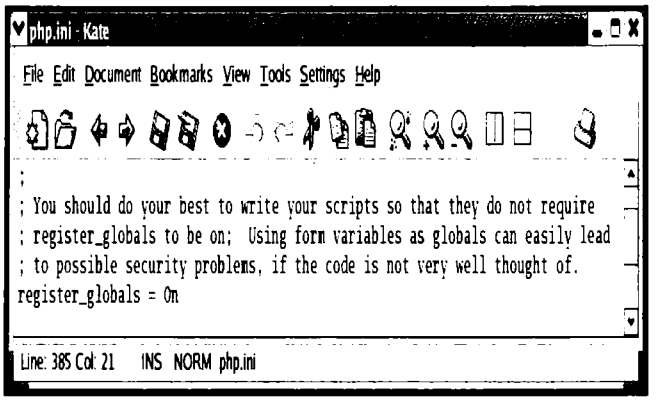


Diagram 12.3.0

The PHP page, which responds to the information passed by **login.html**, should be name **logincheck.php**.

For the Login module to be functional, the PHP module should be capable of accessing information returned to it from the Clients browser. The following illustration demonstrates the technique used by the PHP module to access information returned to it.

Consider the following code, created and saved in the file **logincheck.php:**

```
<?
/*
  Date :- 11/05/05
  Author :- Hansel Colaco.
  Filename :- logincheck.php
  Purpose :- Validates a visitor's login information and displays
  the values passed to it from the calling form.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco';

/* Storing the system date in the format 1st Jan, 2000. */
  $sysdt = date("jS M, Y");
/* Storing the system time in the format 00:00. */
  $systm = date("H:i");

/* Checking if the variable, either username or password, is
empty. This means that the visitor has managed to submit form
information with empty fields. */
  if(empty($txtUserName) || empty($txtPassword)) {
?>
<!-- Generating a page to indicate a flaw in the login attempt.
-->
<HTML><HEAD><TITLE>Login Failure</TITLE></HEAD>
<BODY><H2>An unsuccessful login was attempted!!</H2><HR>
<?
  }  // End of code block executed for empty username or
        password.
/* When both variables (i.e. username or password) hold a value,
the visitor has managed to submit valid information. */
  else {
?>
<!-- Generating a page to indicate a genuine login attempt. -->
<HTML><HEAD><TITLE>Login Succeed</TITLE></HEAD>
<BODY><H2>A successful login was attempted!!</H2><HR>
<?   }  // End of code block executed for valid username &
           password.
?>
<!-- Presenting the login inform in a tabular layout. -->
  <H4>Details of the attempt are:<H4>
  <TABLE Border="1" CellSpacing="1" Width="200"><TR>
      <TD Align="right" Width="75">Username :</TD>
      <TD Align="left" Width="100"><?="$txtUserName";?></TD>
  </TR><TR>
      <TD Align="right" Width="75">Password :</TD>
      <TD Align="left" Width="100"><?="$txtPassword";?></TD>
  </TR><TR>
```

```
        <TD Align="right" Width="75">Server Date :</TD>
        <TD Align="left" Width="100"><?="$sysdt";?></TD>
    </TR><TR>
        <TD Align="right" Width="75">Server Time :</TD>
        <TD Align="left" Width="100"><?="$systm";?></TD>
    </TR></TABLE>
</BODY></HTML>
```

The output of the **logincheck.php** file is:

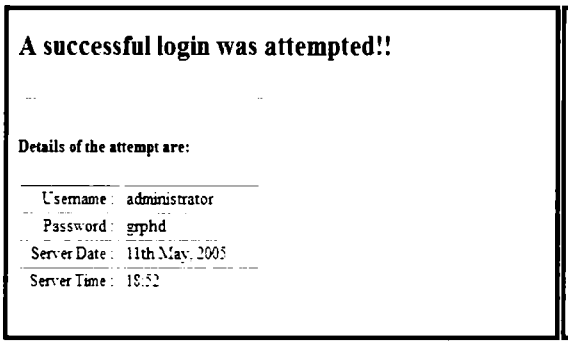


Diagram 12.3.1: Page for valid information.

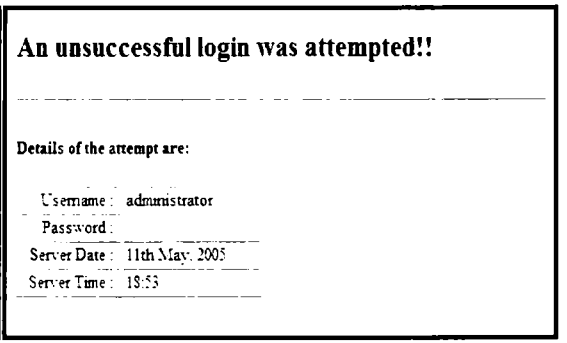


Diagram 12.3.2: Page for invalid information.

Understanding the **logincheck.php** file is fairly simple. The page begins with a group of commented information, which covers the following:

- The date when the file was created
- The author of the file
- The file name for identification
- A brief explanation of the functionality of the file

The processing of the **logincheck.php** file is as follows:

- A variable named **$author** is defined to store the name of the programmer who designed/coded the page
- A variable named **$sysdt** is defined to store the system date of the server serving the PHP page. The date is stored in the format 1st Jan, 2000
- A variable named **$systm** is defined to store the system time of the server serving the PHP page. The time is stored in the format 24:00
- A check is performed to find if the parameters (variables transferred by the POST method) passed to the PHP page have a value. The parameters are accessed by preceding the parameter name with a dollar sign ($), i.e. the parameters **txtUsername** and **txtPassword** will be accessed by the variables **$txtUsername** and **$txtPassword** respectively (This requires **register_globals ON** in the php.ini file)

- If either one of the two variable is found to be empty, a HTML page is designed which indicate an unsuccessful login attempt
- If variables **$txtUsername** and **$txtPassword** have a value, a HTML page is designed which indicate a successful login attempt

- Finally, the details of variables available to the page are displayed in a tabular layout. The variables are listed as:
  - The Username entered by the visitor in the login.html page (i.e. value of $txtUsername)
  - The Password entered by the visitor in the login.html page (i.e. value of $txtPassword)
  - The System date when the PHP file was processed (i.e. value of $sysdt)
  - The System time when the PHP file was processed (i.e. value of $systm)

The page generated by the PHP file is then sent to the visitor providing the login information and rendered within the Web browser.

REMINDER

The above code spec assumes that the web server hosting PHP pages has its **register_globals** attribute turned **ON**. This attribute is found in the **php.ini** file.

The setting **register_globals** = **ON** is required to allow the transfer of HTML form data values from **login.html** to the target PHP module (i.e. logincheck.php). In other words, the values captured in the HTML form field(s) are made available to the PHP module, which is indicated as the **Value** to the **Action** attribute in the <FORM> tag of the HTML page.

PHP automatically creates a collection of variables, *having the same names as the HTML form fields* and assigns them corresponding values retrieved from the form fields. For example, the PHP module creates a variable named **$txtUsername** and assigns it the value returned by the form field **txtUsername**.

If the register_globals attribute is turned **OFF**, the PHP module **will not** be able to access the form fields (variables) returned by the Clients browser directly. In such situations, the **$_POST[]** technique will be required to access the values passed by the HTML form.

In the above situation, the following lines of code should be inserted before the **if** condition in the **logincheck.php** file:

```
/* Using $_POST[] to extract values held in the URL string. */
  $txtUserName = $_POST['$txtUserName'];
  $txtPassword = $_POST['txtPassword'];
```

This completes the Login module, which is based on communication between a HTML form and a PHP module.

Designing A Data Entry fOrm Using PHP

The real advantage of a PHP module when compared to a simple HTML page is the level of dynamism that can be attained. Unlike HTML pages, PHP allows the clubbing of code for data capture **and** data processing into a single file.

The following example will help illustrate this and will cover the following:

- Designing an HTML form to capture information, (i.e. Job categories consisting of Category name and description fields)
- Developing JavaScript code blocks to perform the following:
  - Remove extra spaces on either side of the value entered in a field
  - Change the contents in a field to title case (i.e. first alphabet of every word, capitalized)
  - Check for empty values when the data is submitted
  - Populate the form fields when specific data is selected for editing
  - Clear variables and form fields when data is deleted
- Developing a PHP module to perform the following:
  - Access information captured via the form fields
  - Display captured information by generating a tabular layout below the HTML form
- Enhancement to the page by extracting generic code to an external file

Building An HTML Based Data Manipulation Form

This example is based on the Category master form. A d/e form with two text boxes that captures a category name and its description and a grid, is created. The data captured using the two textboxes is displayed in the grid. The grid is simply an HTML table, which displays the data captured. This grid will allow modifications to the data captured. When a link is clicked, the record will appear in the text boxes for modification. The grid will allow deleting the records using the check box on the right hand side of each record displayed in the grid.

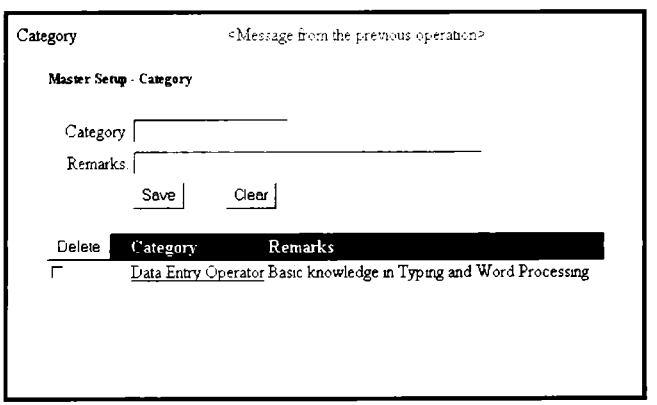


Diagram 12.3.3

Skeleton Of The Code Spec

To understand the development of the Category master form, glance through its pseudo structure as shown:

```
<?
```

1	Commented statements providing a description and purpose of the file.

```
/*
  Date :- <Creation Date >
  Author :- <Developer's Name>
  Filename :- <Filename>
  Purpose :- <Brief Description About File Functionality>
*/
```

2	Common PHP variables.

```
/* A variable holding the developer's name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Category Master Form";
```

3	PHP code block to perform the DELETE operation.

```
  /* Checking if the form is in the delete mode. */
     if ($hidMode == 'D') {
```

3.1	PHP code block to generated the message which indicates the data lost during deletion.

4	PHP variables re-initialization bound to the page processing/layout.

```
/* A variable named $hidMode is initialised to 'I'. This variable
holds the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
?>
```

5	HTML code block rendering the actual page begins.

```
<HTML>
<HEAD>
  <TITLE><?=$title;?></TITLE>
```

6	META tags bound to the HTML page.

7	SCRIPT tag holding JavaScript under the HEAD section.

```
</HEAD>
```

8	BODY section.

```
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
```

9	SCRIPT tag holding JavaScript unique to the current page under the BODY section.

9.1	JavaScript function to check contents of form field before submitting data contained in them.

```
/* The chkBlanks() function verifies that the form field of
category name is not left empty. If the field is empty or
contains blank spaces, a message indicates to enter a valid
value. If the field is not empty, the data is submitted. */
    function chkBlanks() {
```

9.2	JavaScript function to validate contents of form field before submitting data contained in them.

```
/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
    function vldtFrmFlds() {
```

9.3	JavaScript function to prepare the form to accept a new set of data.

```
/* The function setNewMode() is called to prepare the form to
except a new set for data. This function is called directly when
the Clear button is clicked. */
    function setNewMode() {
```

9.4	JavaScript function to prepare the form to accept changes in the selected set of data.

```
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
    function setEditMode(id, name, rmrk) {
```

9.5 JavaScript function to prepare the form to delete selected set(s) of data.

```
/* The function setDelMode() is called to prepare the form to
remove data from the tabular layout. This function is called
directly when the Delete button is clicked. */
   function setDelMode() {
```

9.6 SCRIPT tag holding JavaScript under the BODY section ends.

```
</SCRIPT>
```

10 Parent Table begins.

```
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="<?=$gl_mstr_frst_bar_color;?>" Border="0"
        CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="90%">
```

10.1 Parent Table → First Row: Columns to hold page title and a message header.

Category	<Message from the previous operation>

Diagram 12.4.1

11 Form Section holding Data and Control objects begins.

```
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="ctgry_mstr.php" Method="post" Name="frmMstrCat"
        onSubmit="return chkBlanks();">
```

11.1 Form Hidden Variable to identify the current page.

```
  <!-- Declaring a hidden form field used to identify the current
  (i.e. category master) page. -->
     <INPUT Name="hidPage" Type="hidden" Value="<Page Name>">
```

11.2 Form Hidden variables to maintain a copy of the data held by the form fields just before changes attempted. This is useful to confirm change(s) made to the form fields before an update is fired. (To avoid database or flat files concern as the case may be)

```
  <!-- Declaring hidden form fields for data validation. -->
     <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
     <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
     <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
```

11.3	**Form Hidden Variable to determine the form operation. Default, being Insert operation.**

```
<!-- Declaring a hidden form field used for determining the
form mode. It holds 'I' for Insert, 'U' for Update and 'D' for
Delete. It will hold 'I' when the page is rendered for the 1st
time. -->
    <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
```

11.4	**Form Hidden Variable holding entries (checkbox values) selected for deletion.**

```
<!-- Declaring a hidden form field used for identifying records
which have been selected for deletion. -->
    <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
```

12	**Parent Table → Second Row: Columns to hold the actual d/e form.**

```
<TR Height="300" VAlign="top">
    <TD Align="center" Border="1" ColSpan="10"><BR>
```

13	**First Child Table holding d/e Form objects, to Capture / Control data begins.**

```
    <!-- Form Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
               Name="TlbInner" Width="90%">
```

13.1	**First Child Table → First and Second Rows: Columns to hold form title.**

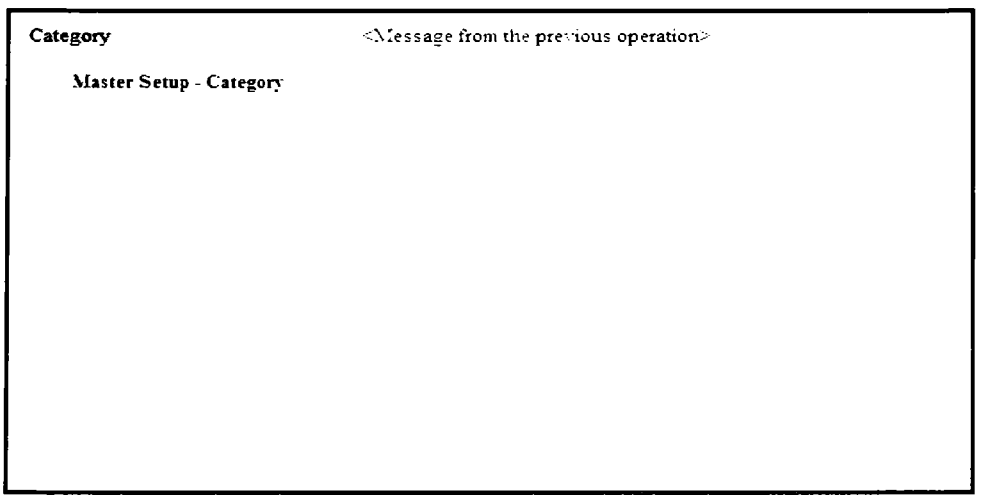


Diagram 12.4.2

13.2	First Child Table → Third & Forth Rows: Columns holding label and data capture objects to capture data.

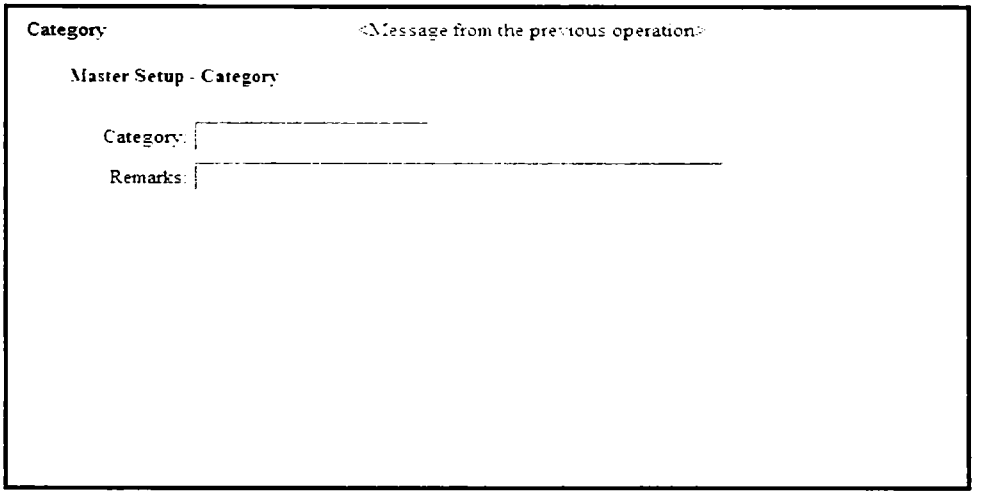


Diagram 12.4.3

13.3	First Child Table → Last Row: Columns holding data control objects.

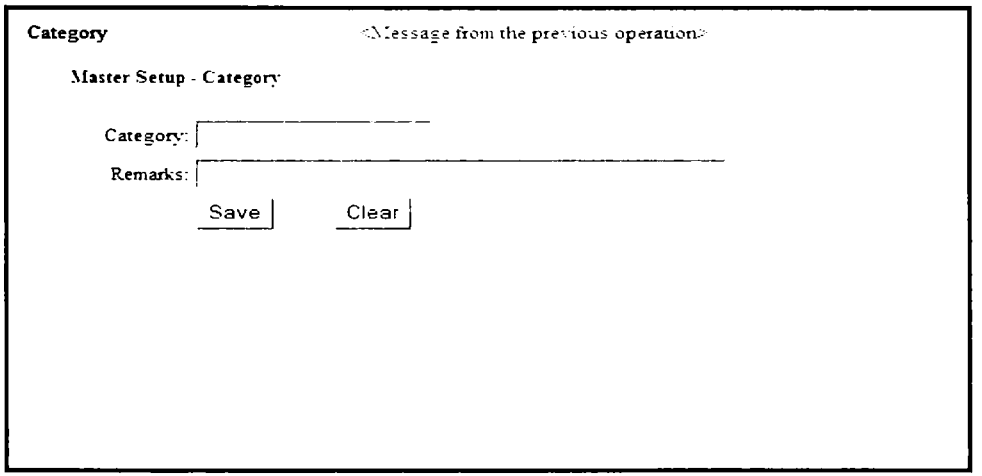


Diagram 12.4.4

13.4	First Child Table ends.

```
<?
```

14	PHP code block retrieving data to create the tabular layout. The tabular layout is generated only after data gets retrieved.

```
?>
```

15	Second Child Table holding the tabular layout, to display / control data captured begins.

```
<!-- Data Table Code Begins. -->
    <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
           Width="90%">
```

16	Second Child Table → First Row: Columns to header row.

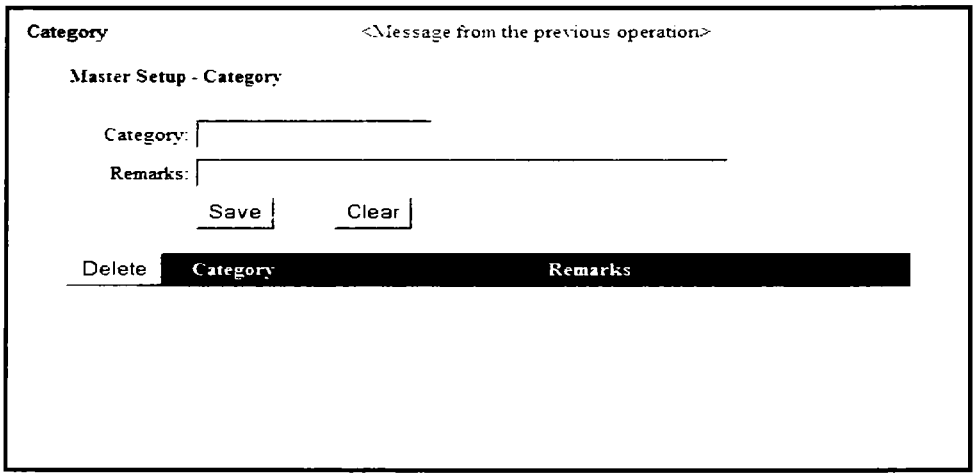


Diagram 12.4.5

17.1	HTML code block to populate data in the tabular layout.

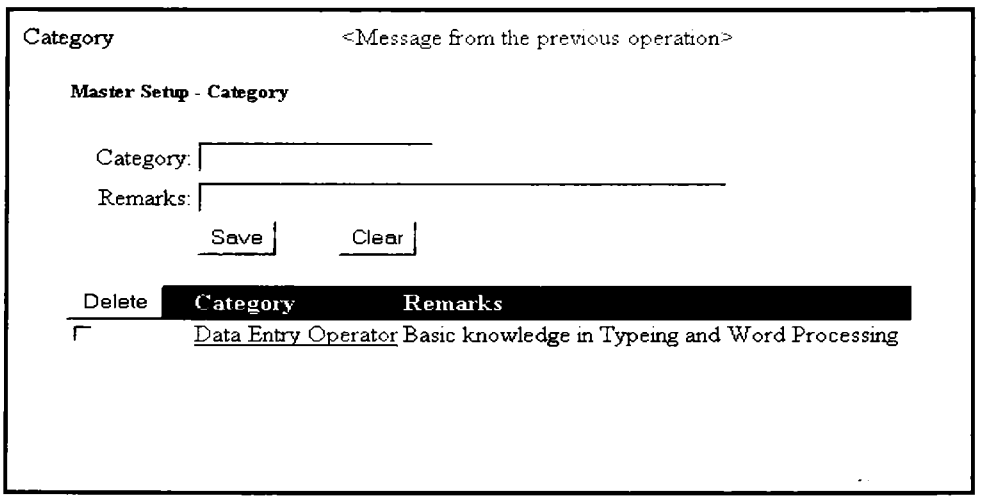


Diagram 12.4.6

17.2	The Second Child table ends.

18	Form holding Data and Control objects ends.

19	Parent Table ends.

```
</TABLE>
<!-- Outer Table Code Ends. -->
```

20	HTML code block along with BODY section rendering the actual page ends.

```
</BODY>
</HTML>
```

Form Layout For Capturing Information

The data capture form is created using HTML's <FORM> and <TABLE> tags. (Refer points **1, 2, 5, 6, 8, 10, 10.1, 11, 11.1, 12, 13, 13.1, 13.2, 13.3, 13.4, 18, 19 and 20** under the sub-title **Skeleton Of The Code Spec.**) Create a PHP file named **ctgry_mstr.php** and enter the code spec as shown below:

```
<?
/*
  Date :- 13/05/05
  Author :- Hansel Colaco.
  Filename :- ctgry_mstr.php
  Purpose :- Allows display, insert, updates and delete of
  category master table.
*/
/* A variable holding the developer's name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Category Master Form";
?>
<HTML><HEAD><TITLE><?=$title;?></TITLE>
  <META Content="<?=$author;?>" Name="Author">
  <META Content="" Name="Keywords">
  <META Content="" Name="Description">
  <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
  <BR><BR>
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0"
        CellSpacing="0" Name="TlbOuter" Width="90%"><TR>
    <TD Align="center" Border="1" BgColor="#C0C0C0"
        Width="10%">Category</TD>
    <TD Align="center" BgColor="#FFFFFF" ColSpan="9"
        Width="90%"> </TD>
  </TR>
```

```
<!-- Initializing a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="ctrgy_mstr.php" Method="post" Name="frmMstrCat">
  <!-- Declaring a hidden form field to identify/store the
  current page name. -->
     <INPUT Name="hidPage" Type="hidden" Value="ctgry_mstr">
  <TR Height="300" VAlign="top">
     <TD Align="center" Border="1" ColSpan="10"><BR>
     <!-- Form Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
              Name="TlbInner" Width="90%"><TR>
           <TD Align="left" ColSpan="2"><FONT Size="2"
                Color="#0000CC"><B>Master Setup -
                Category</B></FONT></TD>
        </TR><TR>
           <TD Align="center" ColSpan="2"> </TD>
        </TR><TR>
           <TD Align="right" Width="15%">Category:</TD>
           <TD Align="left"><INPUT maxLength="35" Name="txtCatName"
              Type="text"></TD>
        </TR><TR>
           <TD Align="right">Remarks:</TD>
           <TD Align="left"><INPUT maxLength="255" Name="txtCatRmrk"
              Type="text" Size="50"></TD>
        </TR><TR>
           <TD> </TD>
           <TD Align="left">
              <INPUT class="button" Name="cmdSubmit" Type="submit"
                    Value="Save">
              <IMG Height="1" Src="../Images/pixel.gif" Width="30">
              <INPUT class="button" Name="cmdReset" Type="button"
                    Value="Clear">
           </TD>
        </TR></TABLE><BR>
     <!-- Form Table Code
     End. -->
     </TD>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code
  Ends. -->
</BODY></HTML>
```

Run the PHP module by calling it from a Web Browser. The layout appears as shown in diagram 12.5.

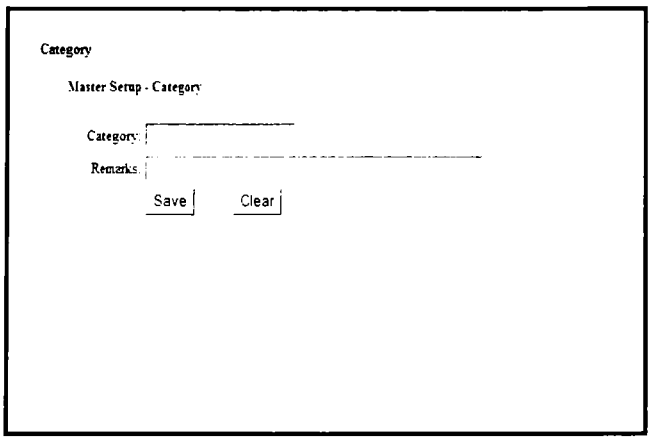


Diagram 12.5: Output for **ctgry_mstr.php**.

The page consists of two fields and two buttons. It also contains a hidden form field (i.e. hidPage), which is used to identify the current page.

The form fields accept a Category name and a remark. When Save is clicked, a call is returned to the same page (**ctgry_mstr.php**) and the data captured data is submitted.

Adding Form Field Validations

The purpose of the above page is to capture user login information. Adding data validation code enhances the functionality of the page.

Since this is client-side validation, JavaScript code is used. This illustration uses two sets of JavaScript code blocks, which are distinguished as:

- Generic code – Can be reused by other pages
- Form bound code – Can be used only within the current page

Adding Generic Code Block

Usually, generic code blocks are placed in an external file. The functionality of the code, contained in the external file, is accessed via a reference to the file.

Create a file named **mstrscript.js** and copy the following contents into it:

```
/*
  Date :- 13/05/05
  Author :- Hansel Colaco
  Filename :- mstrscript.js
  Purpose :- Defines Generic JavaScript functions for trimming
  strings and changing string case.
*/
/* Generating the function used for trimming string values. */
  function strtrim() {
  /* Returns the parameter string after removing blank spaces at
  the beginning or the end of the parameter. */
     return this.replace(/^\s+/,'').replace(/\s+$/,'');
  }
/* Generating a global alias name for the function used for
trimming string values. */
  String.prototype.trim = strtrim;
/* The chngCase() function changes the first alphabet to an upper
case character for the value held in the field passed as the
first parameter. The second parameter is a reference to the form
holding the field in the first parameter. This function is called
when a form cursor moves away from a field. */
  function chngCase(pFld, pFrm) {
```

```
/* Using the eval() function to extract the value held in the
field passed as a parameter. */
   var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
/* Removing extra spaces for the value extracted above and
restoring into the same variable. */
   mFldVal = mFldVal.trim();
/* If the variable containing the extracted value is not empty.
*/
   if (mFldVal != "") {
   /* Changing the string Proper case. */
      mFldVal = mFldVal.charAt(0).toUpperCase() + mFldVal.substr(1,
                mFldVal.length);
   }
/* Using the eval() function to display the new value in the
field passed as a parameter. */
   eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
}
```

To make the above functionality available in the Category master form, modify the **ctgry_mstr.php** file. (Refer point **7** under the sub-title **Skeleton Of The Code Spec.**) Insert the following line of code before the </HEAD> tag:

```
<META Content="" Name="Description">
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<SCRIPT Language="JavaScript" Src="mstrscript.js" Type="text/javascript">
</SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0"><BR><BR>
```

The JavaScript functionality in the external file can be used by extending the behavior of the form fields.

Modify the <INPUT> tags, in the **ctgry_mstr.php** file, as shown below (Refer point **13.2** under the sub-title **Skeleton Of The Code Spec.**):

```
...
      <!-- Form Table Code Begins. -->
         <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0" Name="TblInner"
                Width="90%"><TR>
         ...
         </TR><TR>
            <TD Allgn="right" Width="15%">Category:</TD>
            <TD Align="left"><INPUT class="text" maxLength="35"
               Name="txtCatName" onBlur="chngCase('txtCatName',
               'frmMstrCat');" Type="text"></TD>
         </TR><TR>
            <TD Align="right">Remarks:</TD>
            <TD Align="left"><INPUT class="text" maxLength="255"
               Name="txtCatRmrk" onBlur="chngCase('txtCatRmrk',
               'frmMstrCat');" Type="text" Size="50"></TD>
```

```
            </TR><TR>
                <TD> </TD>
                ...
            </TR></TABLE><BR>
        <!-- Form Table Code End. -->
...
```

The attribute **onBlur** defines the action carried out when the current object loses form focus. In this case, the name of the current form and the object losing focus is passed to function **chngCase()**. The **chngCase()** function will change case of the string held in the object losing focus.

Adding The Form Bound Code Block

The additional JavaScript features are unique to this form and are held within the **ctgry_mstr.php** file itself. These features include the OnClick functionality of the Clear button and verifying information captured by this form.

The JavaScript code block unique to the category master form will be placed in the BODY section. Alternatively, if desired, this code block can be placed within the HEAD section.

Add these additional JavaScript features by inserting the following code before the first <TABLE> tag. (Refer points **9, 9.1 and 9.3** under the sub-title **Skeleton Of The Code Spec.**):

```
    <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0"><BR><BR>
    <SCRIPT Language="JavaScript">
    /* The chkBlanks() function verifies that the form field of
    category name is not left empty. If the field is empty or
    contains blank spaces, a message indicates to enter a valid
    value. If the field is not empty, the data is submitted. */
        function chkBlanks() {
        /* A variable holding a reference to the form object is
        initialized. */
            frm = document.frmMstrCat;
        /* Extracting the contents of the category name field into a
        variable named str. */
            var str = frm.txtCatName.value;
        /* If the category name field is empty or contains blank
        spaces. */
            if(str.trim() == "") {
            /* A message indicates to enter a valid value. */
                alert('Please enter category name.');
            /* Placing the form cursor on the category name field. */
                frm.txtCatName.focus();
```

```
        /* Preventing the data in the form field from being
        submitted. */
            return false; }
    /* If the category name field holds a value. */
        else {
        /* Allowing the data in the form field to be submitted.
        */
            return true; }
    }
  /* The function setNewMode() is called to prepare the form to
  except a new set for data. This function is called directly
  when the Clear button is clicked. */
    function setNewMode() {
    /* A variable holding a reference to the form object
    initialized in this page. */
        frm = document.frmMstrCat;
    /* Clearing form fields. */
        frm.txtCatName.value = "";
        frm.txtCatRmrk.value = "";
    }
</SCRIPT>
```

```
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0" CellSpacing="0"
Name="TlbOuter" Width="90%"><TR>
      <TD Align="center" Border="1" BgColor="#C0C0C0" Width="10%">Category</TD>
      <TD Align="center" BgColor="#FFFFFF" ColSpan="9" Width="90%"> </TD>
  </TR>
```

In addition to the above script, the following changes are made in the<FORM> tag and the <INPUT> tag of the Clear button. (Refer points **11 and 13.3** under the sub-title **Skeleton Of The Code Spec.**):

```
...
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0" CellSpacing="0"
Name="TlbOuter" Width="90%"><TR>
      <TD Align="center" Border="1" BgColor="#C0C0C0" Width="10%">Category</TD>
      <TD Align="center" BgColor="#FFFFFF" ColSpan="9" Width="90%"> </TD>
  </TR>
```

```
<!-- Initializing a form object, which will submit data captured
by the form to the processing file. The JavaScript function
chkBlanks() is called to validate data before submitting it. -->
  <FORM Action="ctgry_mstr.php" Method="post" Name="frmMstrCat"
      onSubmit="return chkBlanks();">
```

```
  <!-- Declaring a hidden form field to identify/store the current page name. -->
      <INPUT Name="hidPage" Type="hidden" Value="ctgry_mstr">
  <TR Height="300" VAlign="top">
      <TD Align="center" Border="1" ColSpan="10"><BR>
      <!-- Form Table Code Begins. -->
          <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0" Name="TlbInner"
              Width="90%"><TR>
          ...
```

```
            </TR><TR>
                <TD> </TD>
                <TD Align="left">
                    <INPUT class="button" Name="cmdSubmit" Type="submit" Value="Save">
                    <IMG Height="1" Src="../Images/pixel.gif" Width="30">
<INPUT      class="button"      Name="cmdReset"      onClick="setNewMode();"
Type="button" Value="Clear">
                </TD>
            </TR></TABLE><BR>
        <!-- Form Table Code End. -->
        </TD>
    </FORM>
    </TR></TABLE>
    <!-- Outer Table Code Ends. -->
...
```

Save the changes made to the **ctgry_mstr.php** file and run the page in a web browser. When the page is rendered, click Save without entering any data. This will cause the form to call the **chkBlanks()** function before submitting information.

The **chkBlanks()** function accesses and retrieves information from the form field for category name. The retrieved information is stored into a variable. The **trim()** function is called to remove spaces from either side of the value held in the variable. If the result is an empty variable, a message is displayed which prompts the entry of a category name, (refer diagram 12.6). The form cursor is returned to the form field for category name and the process of sending form data is terminated.

Diagram 12.6: Message displayed when category name not entered in the **ctgry_mstr.php** file.

Similarly, to test the functionality of Clear, enter text into the form fields and then click Clear. The click event will call the JavaScript function **setNewMode()**. This function will reset the contents of the form fields to empty values.

Accessing Data Submitted By The Form

When the form data is submitted to the PHP module on the Web server, the values in the form fields are transferred to PHP variables having the same name as the form fields (This Requires **register_globals** set **ON** in the php.ini file). In this example, the PHP module submits form data to itself.

The form data will be passed to the PHP module via variables with names that match the form fields, (i.e. variable **$txtCatName** and **$txtCatRmrk** will hold the values of the form fields **txtCatName** and **txtCatRmrk** respectively). Besides these form fields, any hidden form fields are also passed to the PHP module called, (i.e. variable **$hidPage** will hold the value of the hidden form field **hidPage**).

For the purpose of verifying whether data entered in the previous form instance is accessible, this illustration will create a tabular layout in which the accessed data will be displayed. Modify the code in the **ctgry_mstr.php** file by inserting the following code block: *(The insert is made between the
 tag after the </TABLE> tab for the table containing the form objects and the </TD> for the parent column containing the form objects.)* (Refer points **13.4, 14, 15, 16, 17.1 and 17.2** under the sub-title **Skeleton Of The Code Spec.**)

```
...
                    <INPUT class="button" Name="cmdReset" Type="button" Value="Clear">
                </TD>
            </TR></TABLE><BR>
<!-- Form Table Code End. -->
```

```
<?
/* Checking if data have been retrieved from the previous
instance of the Category form. */
  if ($txtCatName != '') {
?>
     <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
Width="90%">
        <TR BgColor="#400040">
            <TD Width="15%"> </TD>
            <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
        </TR>
        <TR BgColor="#E4E4E4">
            <TD>1</TD>
            <TD><?echo($txtCatName);?></TD>
            <TD><?echo($txtCatRmrk);?></TD>
        </TR>
        </TABLE><BR>
<?   }  ?>
```

```
      </TD>
   </FORM>
   </TR></TABLE>
   <!-- Outer Table Code Ends. -->
</BODY></HTML>
```

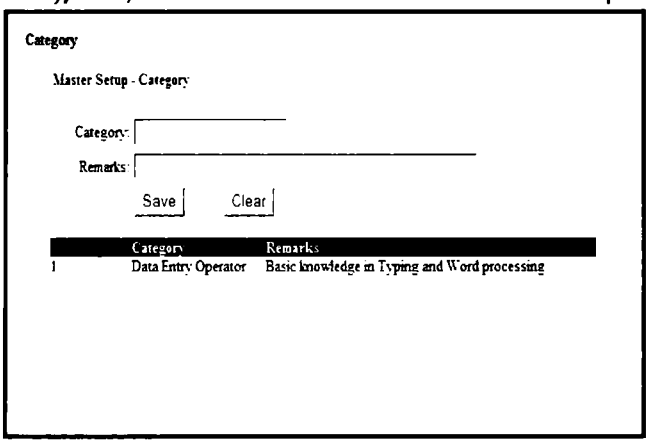


Diagram 12.7: Output for the **ctgry_mstr.php** file.

Save the changes and then run the page in a Web Browser. When the page appears enter some information in the form fields and click Save. This will submit the data captured to the same PHP module for processing. Data is validated and returned via the same HTML page in a tabular layout. The recent addition to the code block will perform the following action:

- A check is made whether the category name field had been populated in the previous instance. If yes, a table with three columns is created using the TABLE tag

- The first row is the header bar (i.e. the second and third column is labeled Category and Remarks respectively)
- The second row contains data (i.e. the first column displays the entry number, the second column displays the previously entered category name, while the third column will display a description entered in the previous instance of the page)

When the page is re-rendered, with the new table, it appears as shown in diagram 12.7.

Updating Data From The Previous Form Instance

Apart from simply displaying values entered in the previous instance, PHP (in combination with JavaScript) can be used to modify this data. To demonstrate this technique, modify the code in the **ctgry_mstr.php** file as mentioned below.

The form fields are empty when the page is rendered. But it is necessary that the fields contain values for the update operation. Additionally, the modified data needs to be validated (which comprises of a comparison between the old and new values).

For the purpose of comparison, hidden form fields are defined to act as temporary variables. One more hidden form fields is defined to act as a temporary variable, which specifies the form mode. (**i.e. I** for **I**nsert, **U** for **U**pdate and **D** for **D**elete).

Insert the code for new hidden form fields immediately after the declaration of the hidden field for identifying the current page. (Refer point **11.2** under the sub-title **Skeleton Of The Code Spec.**) The code inserted is as follows:

```
...
  <!-- Declaring a hidden form field to identify/store the current page name. -->
      <INPUT Name="hidPage" Type="hidden" Value="ctgry_mstr">
  <!-- Declaring hidden form fields required for data validation.
  -->
      <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
      <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
      <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
  <!-- Declaring a hidden form field used for determining the
  form mode. It will holds 'I' for Insert, 'U' for Update and 'D'
  for Delete. It will be left empty when the page is rendered for
  the 1st time. -->
      <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
  <TR Height="300" VAlign="top">
      <TD Align="center" Border="1" ColSpan="10"><BR>
...
```

Initialize the variable **$hidMode** in the declaration block at the beginning of the PHP file (immediately after the initialization of variable **$title**) as, (Refer point **4** under the sub-title **Skeleton Of The Code Spec.**):

```
...
/* A variable holding the developer's name. */
  $author = 'Hansel Colaco.';
```

```
/* A variable holding the title for the page. */
  $title = "Category Master Form";
/* A variable named $hidMode is initialized to 'I'. This variable
holds the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
?>
<HTML><HEAD>
  <TITLE><?=$title;?></TITLE>
...
```

Further, the value displayed in the tabular layout (**i.e.** category name) will be change into a hyperlink. When this hyperlink is clicked, the form fields will be populated with the data displayed in the row containing the hyperlink. Change the second column of the data row as shown below, (Refer point **17.1** under the sub-title **Skeleton Of The Code Spec.**):

```
...
      <!-- Data Table Code Begins. -->
          <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%">
          <TR BgColor="#400040">
          ...
          </TR>
          <TR BgColor="#E4E4E4">
              <TD>1</TD>
<TD><A HRef="JavaScript:setEditMode('1', '<?=$txtCatName;?>',
  '<?=$txtCatRmrk;?>')"><?echo($txtCatName);?></A></TD>
              <TD><?echo($txtCatRmrk);?></TD>
          </TR>
          </TABLE><BR>
...
```

As specified in the above code, when the hyperlink is click a JavaScript function named **setEditMode()** is processed. This function will populate the form fields with value passed as parameters. It will also set the form status to update mode.

The code for **setEditMode()** is inserted immediately after the JavaScript function **setNewMode()**. (Refer point **9.4** under the sub-title **Skeleton Of The Code Spec.**)

The contents of the function **setEditMode()** is as shown below:

```
...
  /* The function setNewMode() is called to prepare the form to except a new set for data.
  This function is called directly when the Clear button is clicked. */
      function setNewMode() {
      ...
      }
  /* The function setEditMode() is called to prepare the form to
  update data held in the tabular layout. This function is called
  directly when a link in the tabular layout is clicked. */
      function setEditMode(id, name, rmrk) {
```

```
    /* A variable holding a reference to the form object
    initialized in this page. */
       frm = document.frmMstrCat;
    /* If the form is not in the update mode. */
       if(frm.hidMode.value != 'U') {
       /* Assigning values passed as parameters to hidden
       variables. These are used for comparison during the
       update operation. */
          frm.hidCatID.value = id;
          frm.hidCatName.value = name;
          frm.hidCatRmrk.value = rmrk;
       /* Populating the form fields. */
          frm.txtCatName.value = name;
          frm.txtCatRmrk.value = rmrk;
       /* Setting the form mode to update. */
          frm.hidMode.value = 'U';
       }
    /* If the form is in the update mode. */
       else {
          alert('Update in progress.\n An entry has already been selected for
                modification.');  }
    }
  </SCRIPT>
...
```

The above JavaScript function accepts three parameters and is used to populate form fields and commence the processing involved in the update mode. The processing involved with this function is:

- A check is made whether the current form mode is update or not. If the mode is **not** update:
  - The values passed as parameters are assigned to the hidden form fields
  - The second and third parameters are assigned to the category name and remark fields respectively
  - The hidden field maintaining the form mode is set to **U**pdate mode
- If the form mode is already set to **U**pdate, a message indicates that the current mode is update

With the new additions in form fields the JavaScript function **setNewMode()** will expand. (Refer point **9.3** under the sub-title **Skeleton Of The Code Spec.**) It will now contain the following code in extension to its previous contents:

```
...
  /* The function setNewMode() is called to prepare the form to except a new set for data.
  This function is called directly when the Clear button is clicked. */
     function setNewMode() {
     /* A variable holding a reference to the form object initialized in this page. */
        frm = document.frmMstrCat;
```

```
        /* Clearing form fields. */
            frm.txtCatName.value = "";
            frm.txtCatRmrk.value = "";
        /* Clearing hidden variables used for comparison during the
        update operation. */
            frm.hidCatID.value = "";
            frm.hidCatName.value = "";
            frm.hidCatRmrk.value = "";
        /* Setting the form mode to insert. */
            frm.hidMode.value = "I";
        }
...
```

In addition to clearing form fields, the JavaScript function **setNewMode()** will now clear the new hidden fields and set the hidden field for form status to **I**nsert mode.

When the form is in the update mode an extra validation is required to ensure that data has been changed. This check is brought in to avoid submission of the same information.

The JavaScript function **vldtFrmFlds()** checks for changes during the update mode. This code is inserted between the functions **chkBlanks** and **setNewMode**. (Refer point **9.2** under the sub-title **Skeleton Of The Code Spec.**) The contents of this function is as shown below:

```
...
  <SCRIPT Language="JavaScript">
  /* The chkBlanks() function verifies that the form field of category name is not left
  empty. If the field is empty or contains blank spaces, a message indicates to enter a valid
  value. If the field is not empty, the data is submitted. */
      function chkBlanks() {
          ...
      }
```

```
  /* The function vldtFrmFlds() is called when the form data is
  about to be submitted. This function verifies the presence of
  changes in the form fields during the update mode. If changes
  are not encountered, then a message indicating the same is
  displayed and any further processing gets terminated. */
      function vldtFrmFlds() {
      /* Variable holding reference to the form object
      initialized. */
          frm = document.frmMstrCat;
      /* If the form is in the update mode. */
          if (frm.hidMode.value == 'U') {
          /* Initializing a variable named mNoChng to hold TRUE.
          This value determines whether data in the form field has
          been modified or not. */
              var mNoChng = true;
          /* Checking the form objects individually for modified
          information. If any changes have been made, the value of
          the variable named mNoChng is set to hold FALSE. */
```

```
            if (frm.hidCatName.value != frm.txtCatName.value) {
                mNoChng = false; }
            if (frm.hidCatRmrk.value != frm.txtCatRmrk.value) {
                mNoChng = false; }
        /* If data in the form fields have remained unchanged. */
            if (mNoChng) {
                alert('Update failed.\n No changes have been made during
                      modification.');
            /* Calling the JavaScript function to reset values in
            the form fields. */
                setNewMode();
            /* Placing the form cursor on the category name field.
            */
                frm.txtCatName.focus();
            /* Preventing the data in the form field from being
            submitted. */
                return false;
            }
        /* If data in the form fields have been changed. */
            else {
            /* Allowing the data in the form field to be
            submitted. */
                return true; }
        }
    /* If the form is not in the update mode. */
        else {
    /* Allowing the data in the form field to be submitted. */
        return true; }
    }
/* The function setNewMode() is called to prepare the form to except a new set for data.
This function is called directly when the Clear button is clicked. */
    function setNewMode() {
        ...
    }
...
```

The above function checks whether the page is in the update mode. If yes, the following actions are performed:

- A variable named **mNoChng** is declared to hold **TRUE**
- A check is made whether the value held in the hidden field is different from the value in its corresponding form field. If a change is found, the value of the variable **mNoChng** is changed to **FALSE**
- Finally, a check is made to determine the value held by the variable **mNoChng**. If it holds **TRUE**, it indicates that data has not been changed. The update operation is terminated as:
  - A call to the function **setNewMode()** is made to clear the form fields

- The form cursor is placed on the first field on the form
- The function returns **false**, indicating that the data should **not** be submitted

- If the variable **mNoChng** holds **FALSE**, it indicates that data has been changed. The function returns **TRUE**, indicating that the data should be submitted

*If the form is in a mode other than update, the function returns **true** and terminates itself.*

The function **vldtFrmFlds()** is inserted immediately after the JavaScript function **chkBlanks()**. (Refer point **9.1** under the sub-title **Skeleton Of The Code Spec.**) Since the processing of the function **vldtFrmFlds()** is bound to submission of data, it is called from within the **chkBlanks()** function. Modify the **else** clause in the **chkBlanks()** function to appear as shown below:

```
...
   <SCRIPT Language="JavaScript">
   /* The chkBlanks() function verifies that the form field of category name is not left
   empty. If the field is empty or contains blank spaces, a message indicates to enter a valid
   value. If the field is not empty, the data is submitted. */
       function chkBlanks() {
       /* A variable holding a reference to the form object is initialized. */
           frm = document.frmMstrCat;
       /* Extracting the contents of the category name field into a variable named str. */
           var str = frm.txtCatName.value;
       /* If the category name field is empty or contains blank spaces. */
           if(str.trim() == "") {
           /* Displaying a message indicates to enter a valid value. */
               alert('Please enter category name.');
           /* Placing the form cursor on the category name field. */
               frm.txtCatName.focus();
           /* Preventing the data in the form field from being submitted. */
               return false;
           }
       /* If the category name field holds a value. */
           else {
           /* Calling the function vldtFrmFlds() to validate data
           captured by the form. If the function returns TRUE. */
               if (vldtFrmFlds() == true) {
               /* Allowing the data in the form field to be
               submitted. */
                   return true; }
           /* If the function vldtFrmFlds() returns FALSE during
           validation of data captured by the form. */
               else {
               /* Preventing the data in the form field from being
               submitted. */
                   return false; }
           }
       }
...
```

This completes the changes to be made in the **ctgry_mstr.php** to facilitate modifying data inserted via the previous instance of the page. Test the changes by running the page in a Web Browser.

When the page appears, only a form is displayed. Enter data into the field and click Save. The page reappears with data just captured in the previous instance. Refer diagram 12.8.1.

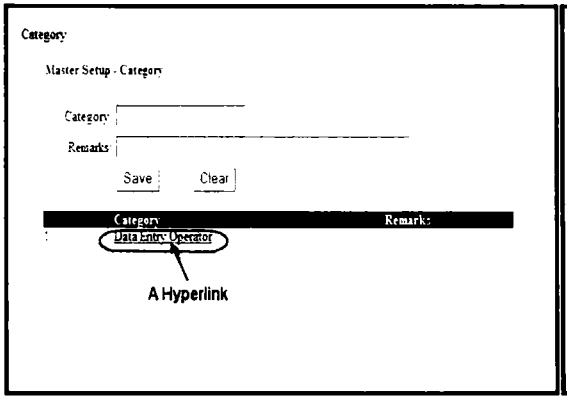


Diagram 12.8.1: Output for the **ctgry_mstr.php** file.

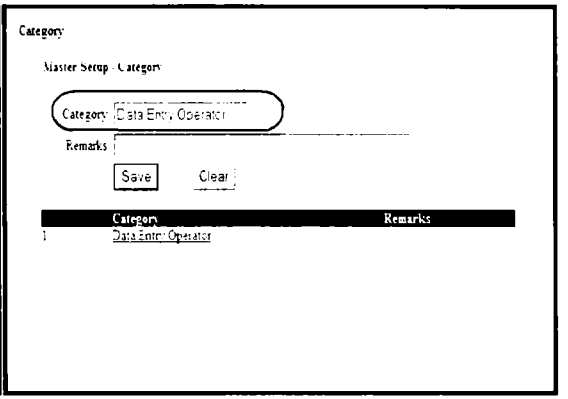


Diagram 12.8.2: Form field populated with data for modification.

The second column in the tabular layout is a hyperlink. When clicked, the form is populated with data displayed in the tabular layout. Refer diagram 12.8.2. This means that the form is in the UPDATE mode.

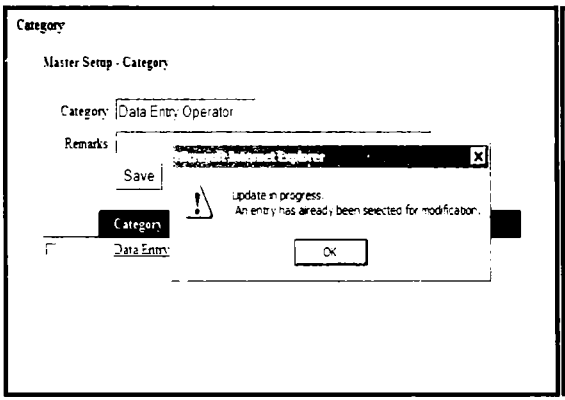


Diagram 12.8.3: Error message displayed when the hyperlink is clicked twice before clicking Save.

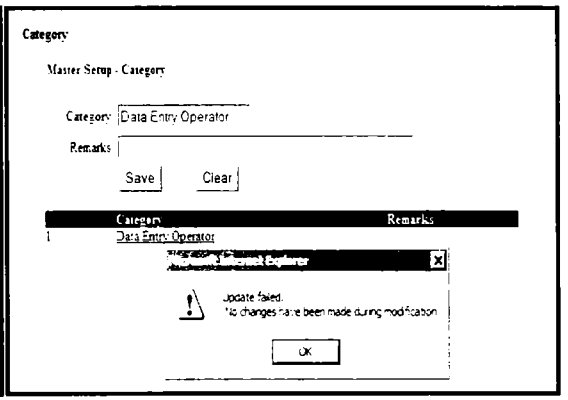


Diagram 12.8.4: Error message displayed when Save is clicked without changing field data.

If the hyperlink is clicked when the form is in the update mode, an error message is prompted which indicates that the previous update operation is not completed. Refer diagram 12.8.3.

If Save is clicked without making any changes, a message (as shown in diagram 12.8.4) prompts for changes. In such situation, the form fields are cleared and the form mode is reverted to the insert mode.

To modify data displayed in the tabular layout, click the hyperlink. Change the contents of the form fields (as shown in diagram 12.8.5) before clicking Save. When the page is re-rendered, the modified data is placed in the tabular layout as shown in diagram 12.8.6.

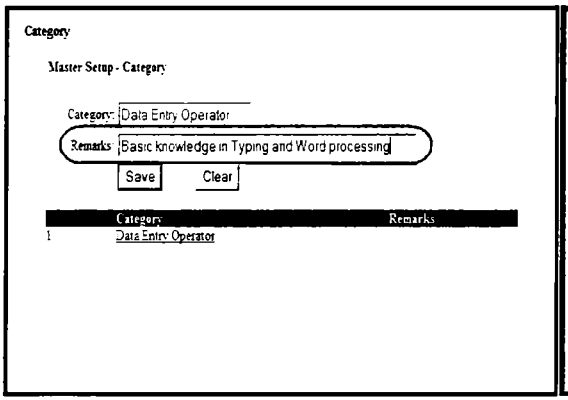


Diagram 12.8.5: Changing field data during the update mode.

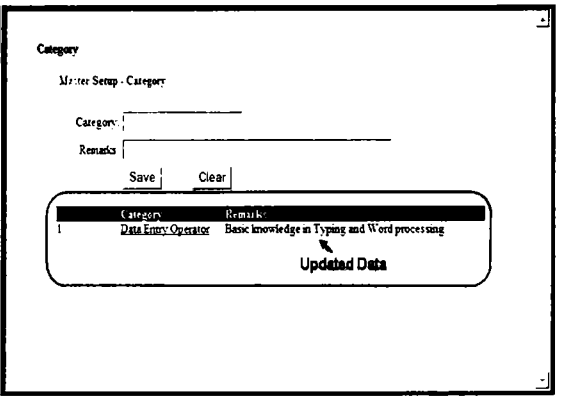


Diagram 12.8.6: The appearance of the tabular layout after modifications.

Deleting Data Previously Captured By The Form

Having completed the insert and update operation of form data, the delete operation is fairly simple. Modify the contents of the **ctgry_mstr.php** file as mentioned below which will add the delete operation to the Category master page.

The delete operation will need a hidden variable to capture the record that needs to be deleted. To support this, insert the following code after the declaration of the hidden field named **hidMode**. (Refer point **11.4** under the sub-title **Skeleton Of The Code Spec.**):

```
...
  <!-- Declaring a hidden form field used for determining the form mode. It will holds 'I'
  for Insert, 'U' for Update and 'D' for Delete. It will be left empty when the page is
  rendered for the 1st time. -->
      <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
```

```
  <!-- Declaring a hidden form field used for identifying records
  which have been selected for deletion. -->
      <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
```

```
  <TR Height="300" VAlign="top">
...
```

The new hidden field is initialized empty and is assigned a value only when delete operation completes.

Modify the appearance of tabular layout so as to display a checkbox for the row of data record and a Delete button. The Delete button will call a JavaScript function to perform the delete operation.

The above changes are made possible by replacing the code of the first column for both rows (header and data) with the following code. (Refer points **16 and 17.1** under the sub-title **Skeleton Of The Code Spec.**):

```
...
      <!-- Data Table Code Begins. -->
          <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%"><TR
                BgColor="#400040">
```

```
              <TD  Width="15%"><INPUT  class="button"  Name="cmdDelete"
onClick="setDelMode();" Type="button" Value="Delete"></TD>
```

```
              <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
              <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
          </TR>
          <TR BgColor="#E4E4E4">
```

```
              <TD><INPUT Name="chk1" Type="checkbox" Value="1"></TD>
```

```
              <TD><A HRef="JavaScript:setEditMode('1', '<?=$txtCatName;?>',
                  '<?=$txtCatRmrk;?>')"><?echo($txtCatName);?></A></TD>
              <TD><?echo($txtCatRmrk);?></TD>
          </TR>
          </TABLE><BR>
...
```

View the modified page in the web browser. Refer diagram 12.9.1. As seen in the diagram the header row of the tabular layout will display Delete, while the data row will have a checkbox in the first column.

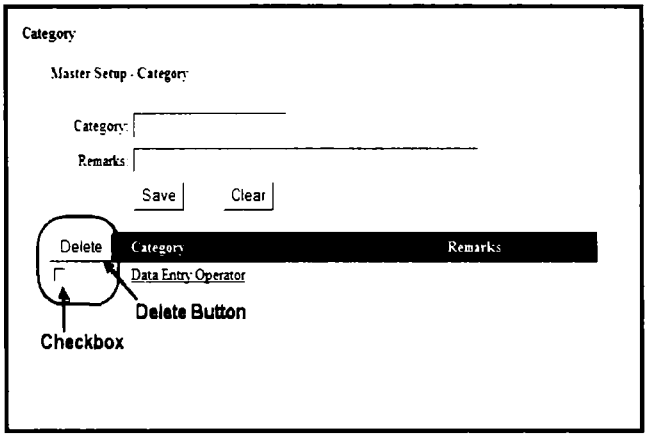


Diagram 12.9.1: Output for the **ctgry_mstr.php** file.

When Delete is clicked, a call to the JavaScript function **setDelMode()** is made. This code block is placed immediately after the function **setEditMode()**.

The following code is inserted in the JavaScript code block of the **ctgry_mstr.php** file. (Refer point **11.5** under the sub-title **Skeleton Of The Code Spec.**) It specifies the processing for the **setDelMode()** function:

```
...
  /* The function setEditMode() is called to prepare the form to update data held in the
  tabular layout. This function is called directly when a link in the tabular layout is
  clicked. */
      function setEditMode(id, name, rmrk) {  ...  }
```

```
  /* The function setDelMode() is called to prepare the form to
remove data from the tabular layout. This function is called
directly when the Delete button is clicked. */
      function setDelMode() {
```

```
    /* Calling the JavaScript function to reset values in the
    form fields. */
       setNewMode();
    /* Setting the form mode for deleting record(s). */
       document.frmMstrCat.hidMode.value = 'D';
    /* Calling the JavaScript function to populate the variable
    holding a list of deleted records. The function
    formDeleteValues() has been defined in the mstrscript.js
    file. */
       formDeleteValues('hidDelRcrdLst', 'frmMstrCat');
    }
  </SCRIPT>
...
```

The above function begins with a call to the function **setNewMode()** to clear all form fields. Then the hidden field **hidMode** is set to indicate that the delete mode is in progress. Finally, a call to the generic function **formDeleteValues()** is made by passing two parameters. The parameters passed includes the name of the hidden field used to identify entries to be deleted and the name of the current form.

Changes In setNewMode() And setEditMode() Functions

The delete process adds new objects to the page form causing the functionality of the **setNewMode()** and **setEditMode()** functions to extend.

Add the following code block toward the end of the **setNewMode()** function, (Refer point **11.3** under the sub-title **Skeleton Of The Code Spec.**):

```
...
  /* The function setNewMode() is called to prepare the form to except a new set for data.
  This function is called directly when the Clear button is clicked. */
     function setNewMode() {
     ...
     /* Setting the form mode to insert. */
        frm.hidMode.value = "I";
```

```
     /* If the form contains a Delete button. */
        if (frm.cmdDelete) {
        /* Enabling the Delete button. */
           frm.cmdDelete.disabled = false;       }
```

```
     }
...
```

The earlier code block checks if the page contains the object Delete, if found it is enabled.

The additional code for the **setEditMode()** function is placed after the hidden field (indicating the form status) is set to update. (Refer point **11.4** under the sub-title **Skeleton Of The Code Spec.**) The code disables Delete as shown below:

```
...
   /* The function setEditMode() is called to prepare the form to update data held in the
   tabular layout. This function is called directly when a link in the tabular layout is
   clicked. The function also sets the hidden field form mode to update and populates the text
   fields */
       function setEditMode(id, name, rmrk) {
       /* A variable holding a reference to the form object initialized in this page. */
           frm = document.frmMstrCat;
       /* If the form is not in the update mode. */
           if(frm.hidMode.value != 'U') {
               ...
           /* Setting the form mode to update. */
               frm.hidMode.value = 'U';
           /* Disabling the Delete button. */
               frm.cmdDelete.disabled = true;
           }
       /* If the form is in the update mode. */
           else {
               alert('Update in progress.\n An entry has already been select for modification.');}
       }
...
```

Additions In The External JavaScript File

Since the **setDelMode()** function calls the **formDeleteValues()**, the file **mstrscript.js** file (containing generic JavaScript functions) needs to be updated. Add the following code block towards the end of this file:

```
...
/* The formDeleteValues() function generates a string of comma-
separated identities of records selected for deletion. It is
called when the DELETE button is clicked. */
  function formDeleteValues(pHid, pFrm) {
  /* Initializing variables for later use. */
      var selValues = "";
      var firstSelBox = "";
  /* Using the eval() function to create a variable to hold a
  reference to the form object. */
      eval("frm = document." + pFrm);
  /* Iterating through every object on the form. */
      for (i=0; i<frm.elements.length; i++) {
      /* If the current object is a Checkbox. */
          if (frm.elements[i].type == "checkbox") {
          /* If the variable firstSelBox is empty. */
              if (firstSelBox == "") {
              /* Assigning the element number, of the current form
              object, to the variable firstSelBox. */
                  firstSelBox = i; }
              /* If the current Checkbox has been selected, (i.e.
              checked). */
              if (frm.elements[i].checked == true) {
```

```
            /* Assigning the element number, of the current form
            object, to the variable selValues. */
               selValues = selValues + frm.elements[i].value + ",";      }
         }
      }
   /* If the variable selValues does not hold a value. */
      if (selValues.length < 1) {
      /* Displaying a message indicating to select a checkbox. */
         alert('Please choose records you wish to delete.');
      /* Using the eval() function to place the form cursor on the
      first checkbox. */
         eval("document." + pFrm + ".elements[" + firstSelBox + "].focus();");
      }
   /* If the variable selValues hold a value. */
      else {
      /* Removing the last character (i.e. a comma) for the value
      held in the variable selValues. */
         selValues = selValues.substring(0, selValues.length-1);
      /* Using eval() function to assign a value to the form's
      hidden variable. */
         eval("document." + pFrm + "." + pHid + ".value = '" + selValues + "';");
         frm.submit();
      }
   }
```

The **formDeleteValues()** function generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same. In this function, the first parameter is a reference to a hidden variable (that will be used to store a string, generated for listing the records marked for deletion), while the second parameter is a reference to the form object. The function is called when the DELETE button is clicked.

In this example only one record can be entered and deleted as there exists no storage mechanism for form data, (such as a base table) this will be covered in the next chapter.

Having saved the **mstrscript.js** file, return to **ctgry_mstr.php** file to perform the final changes for the delete operation. When processed, the generic JavaScript function **formDeleteValues()** stores a list of deleted entries into the hidden field **hidDelRcrdLst**.

The value in this field is assessed and a message is generated to display the list of entries deleted in the previous instance of the page.

Add the following code block before declaring the PHP variable named **$hidMode**, (Refer point **3.1** under the sub-title **Skeleton Of The Code Spec.**):

```
...
/* A variable holding the developers name. */
   $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
   $title = "Category Master Form";
```

```
/* If the form mode is set to Delete. */
  if ($hidMode == 'D') {
     /* A variable $Message is initialized to hold a message,
     which has occurred during the previous (delete) operation.
     */
        $Message = '<BR><FONT Color="RED">* Details for '.$hidDelRcrdLst.' has
                   been deleted.</FONT>';      }
/* A variable named $hidMode is initialized to 'I'. This variable hold the form status in
terms of I - Insert, U - Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

The above code checks the value of the variable **$hidMode**, returned by any previous instance of the page. If it holds D, (**i.e.** the delete operation was performed in the previous instance of the page,) a new PHP variable **$Message** is created to hold a message indicating the entries deleted.

Finally, the message created above is displayed by adding the following code, within the second column of the first row of the outer TABLE object on the page, (Refer point **13.1** under the sub-title **Skeleton Of The Code Spec.**):

```
...
  <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0" CellSpacing="0"
Name="TlbOuter" Width="90%"><TR>
      <TD Align="center" Border="1" BgColor="#C0C0C0" Width="10%">Category</TD>
      <TD Align="center" BgColor="#FFFFFF" ColSpan="9" Width="90%">
        <?=$Message;?></TD>
  </TR>
...
```

Save the changes made to the file **ctgry_mstr.php** and run it. Enter data into the fields and click Save. The inserted data is displayed in a tabular layout with a checkbox in the first column. **Note** the placement of Delete. Refer diagram 12.9.1.

Now, attempt to update data by clicking on the hyperlink (placed within the tabular layout). When the form fields are populated Delete is automatically disabled. This is because the form is in the update mode. Refer diagram 12.9.2.

Click Clear to return to the insert mode and notice Delete is enabled.

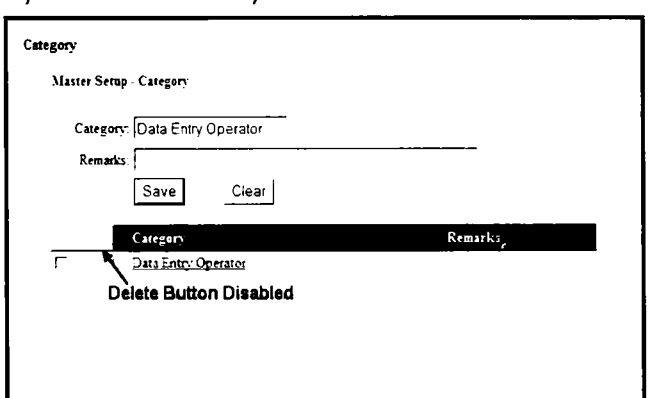


Diagram 12.9.2: Delete disabled during the Update mode.

Test the delete feature by clicking Delete without selecting the checkbox. A message as shown in diagram 12.9.3 will be displayed.

Diagram 12.9.3: Message displayed when the delete option is clicked without selecting a checkbox.

Now, select the checkbox and click Delete, the delete operation proceeds and the page is re-rendered with a message as shown in diagram 12.9.4.

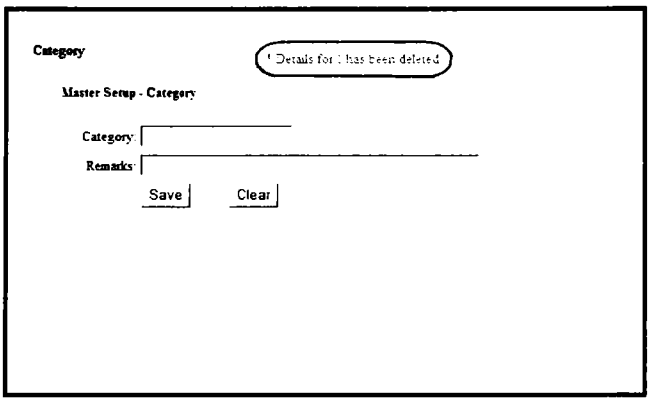


Diagram 12.9.4: Message displayed after the delete operation completes.

The final contents of the file **ctgry_mstr.php** appears as follows:

```
<?
/*
  Date :- 13/05/05
  Author :- Hansel Colaco.
  Filename                :-
  ctgry_mstr.php
  Purpose     :-     Allows
  display, insert, updates
  and delete of category
  master table.
*/
/* A variable holding the
developers name. */
  $author = 'Hansel Colaco.';
/* A variable holding the
title for the page. */
  $title = "Category Master Form";
/* If the form mode is set to Delete. */
  if ($hidMode == 'D') {
     /* A variable $Message is initialized to hold a message,
     which has occurred during the previous (delete) operation.
     */
        $Message = '<BR><FONT Color="RED">* Details for '.$hidDelRcrdLst.' has
                   been deleted.</FONT>';     }

/* A variable named $hidMode is initialized to 'I'. This variable
hold the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
?>
<HTML><HEAD><TITLE><?=$title;?></TITLE>
  <META Content="<?=$author;?>" Name="Author">
  <META Content="" Name="Keywords">
  <META Content="" Name="Description">
  <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```
<!-- Generic Java script code-spec to trim a string and change
string case.  -->
  <SCRIPT Language="JavaScript" Src="mstrscript.js"
          Type="text/javascript"></SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0"
       TopMargin="0"><BR><BR>
<!-- JavaScript code-spec unique to the category master form. -->
  <SCRIPT Language="JavaScript">
  /* The chkBlanks() function verifies that the form field of
  category name is not left empty. If the field is empty or
  contains blank spaces, a message indicates to enter a valid
  value. If the field is not empty, the data is submitted. */
     function chkBlanks() {
     /* A variable holding a reference to the form object
     initialized in this page. */
        frm = document.frmMstrCat;
     /* Extracting the contents of the category name field into a
     variable named str. */
        var str = frm.txtCatName.value;
     /* If the category name field is empty or contains blank
     spaces. */
        if(str.trim() == "") {
        /* Displaying a message indicating to enter a valid
        value. */
           alert('Please enter category name.');
        /* Placing the form cursor on the category name field. */
           frm.txtCatName.focus();
        /* Preventing the data in the form field from being
        submitted. */
           return false;
        }
     /* If the category name field holds a value. */
        else {
        /* Calling the function vldtFrmFlds() to validate data
        captured by the form. If the function returns TRUE. */
           if (vldtFrmFlds() == true) {
           /* Allowing the data in the form field to be
           submitted. */
              return true;  }
        /* If the function vldtFrmFlds() returns FALSE during
        validation of data captured by the form. */
           else {
           /* Preventing the data in the form field from being
           submitted. */
              return false;
           }
        }
     }
```

```
/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a message indicating the same is
displayed and any further processing gets terminated. */
   function vldtFrmFlds() {
   /* A variable holding a reference to the form object
   initialized in this page. */
      frm = document.frmMstrCat;
   /* If the form is in the update mode. */
      if (frm.hidMode.value == 'U') {
      /* Initializing a variable named mNoChng to hold TRUE.
      This value of this variable determines whether data in
      the form field has been modified or not. */
         var mNoChng = true;
      /* Checking the form objects individually for modified
      information. If any changes have been made, the value of
      the variable named mNoChng is set to hold FALSE. */
         if (frm.hidCatName.value != frm.txtCatName.value) {
            mNoChng = false; }
         if (frm.hidCatRmrk.value != frm.txtCatRmrk.value) {
            mNoChng = false; }
      /* If data in the form fields have remained unchanged. */
         if (mNoChng) {
            alert('Update failed.\n No changes have been made during
                  modification.');
      /* Calling the JavaScript function to reset values in the
      form fields. */
            setNewMode();
         /* Placing the form cursor on the category name field.
         */
            frm.txtCatName.focus();
         /* Preventing the data in the form field from being
         submitted. */
            return false;
         }
      /* If data in the form fields have been changed. */
         else {
         /* Allowing the data in the form field to be
         submitted. */
            return true; }
      }
   /* If the form is not in the update mode. */
      else {
   /* Allowing the data in the form field to be submitted. */
      return true; }
   }
```

```
/* The function setNewMode() is called to prepare the form to
accept a new set of data. This function is called directly when
the Clear button is clicked. */
   function setNewMode() {
   /* A variable holding a reference to the form object
   initialized in this page. */
      frm = document.frmMstrCat;
   /* Clearing form fields. */
      frm.txtCatName.value = "";
      frm.txtCatRmrk.value = "";
   /* Clearing hidden variables used for comparison during the
   update operation. */
      frm.hidCatID.value = "";
      frm.hidCatName.value = "";
      frm.hidCatRmrk.value = "";
   /* Setting the form mode to insert. */
      frm.hidMode.value = "I";
   /* If the form contains a Delete button. */
      if (frm.cmdDelete) {
      /* Enabling the Delete button. */
         frm.cmdDelete.disabled = false;      }
   }
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields */
   function setEditMode(id, name, rmrk) {
   /* A variable holding a reference to the form object
   initialized in this page. */
      frm = document.frmMstrCat;
   /* If the form is not in the update mode. */
      if(frm.hidMode.value != 'U') {
      /* Assigning values passed as parameters to hidden
      variables. These are used for comparison during the
      update operation. */
         frm.hidCatID.value = id;
         frm.hidCatName.value = name;
         frm.hidCatRmrk.value = rmrk;
      /* Populating the form fields. */
         frm.txtCatName.value = name;
         frm.txtCatRmrk.value = rmrk;
      /* Setting the form mode to update. */
         frm.hidMode.value = 'U';
      /* Disabling the Delete button. */
         frm.cmdDelete.disabled = true;
      }
```

```
        /* If the form is in the update mode. */
            else {
                alert('Update in progress.\n An entry has already been selected for
                    modification.');
            }
        }
    /* The function setDelMode() is called to prepare the form to
    remove data from the tabular layout. This function is called
    directly when the Delete button is clicked. */
        function setDelMode() {
        /* Calling the JavaScript function to reset values in the
        form fields. */
            setNewMode();
        /* Setting the form mode for deleting record(s). */
            document.frmMstrCat.hidMode.value = 'D';
        /* Calling the JavaScript function to populate the variable
        holding a list of deleted records. The function
        formDeleteValues() has been defined in the mstrscript.js
        file. */
            formDeleteValues('hidDelRcrdLst', 'frmMstrCat');
        }
    </SCRIPT>
<!-- Outer Table Code Begins. -->
    <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0"
            CellSpacing="0" Name="TlbOuter" Width="90%"><TR>
        <TD Align="center" Border="1" BgColor="#C0C0C0"
            Width="10%">Category</TD>
        <TD Align="center" BgColor="#FFFFFF" ColSpan="9" Width="90%">
            <?=$Message;?></TD>
    </TR>
<!-- Initializing a form object, which will submit data captured
on the form to the processing file. -->
    <FORM Action="ctgry_mstr.php" Method="post" Name="frmMstrCat"
        onSubmit="return chkBlanks();">
    <!-- Declaring a hidden form field used to identify the current
    (i.e. category master) page. -->
        <INPUT Name="hidPage" Type="hidden" Value="ctrgy_mstr">
    <!-- Declaring hidden form fields required for data validation.
    -->
        <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
        <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
        <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
    <!-- Declaring a hidden form field used for determining the
    form mode. It will hold 'I' for Insert, 'U' for Update and 'D'
    for Delete. It will hold 'I' when the page is rendered for the
    1st time. -->
        <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
```

```
  <!-- Declaring a hidden form field used for identifying records
  which have been selected for deletion. -->
     <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
  <TR Height="300" VAlign="top">
     <TD Align="center" Border="1" ColSpan="10"><BR>
     <!-- Form Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
               Name="TlbInner" Width="90%"><TR>
           <TD Align="left" ColSpan="2"><FONT Size="2"
               Color="#0000CC"><B>Master Setup -
               Category</B></FONT></TD>
        </TR><TR>
           <TD Align="center" ColSpan="2"> </TD>
        </TR><TR>
           <TD Align="right" Width="15%">Category:</TD>
           <TD Align="left"><INPUT maxLength="35" Name="txtCatName"
               onBlur="chngCase('txtCatName', 'frmMstrCat');"
               Type="text"></TD>
        </TR><TR>
           <TD Align="right">Remarks:</TD>
           <TD Align="left"><INPUT maxLength="255" Name="txtCatRmrk"
               onBlur="chngCase('txtCatRmrk', 'frmMstrCat');"
               Type="text" Size="50"></TD>
        </TR><TR>
           <TD> </TD>
           <TD Align="left">
              <INPUT Name="cmdSubmit" Type="submit" Value="Save">
              <IMG Height="1" Src="../Images/pixel.gif" Width="30">
              <INPUT Name="cmdReset" onClick="setNewMode();"
                     Type="button" Value="Clear">
           </TD>
        </TR></TABLE><BR>
     <!-- Form Table Code End. -->
<?
/* Checking if data have been retrieved from the previous
instance of the Category form. */
  if ($txtCatName != '') {
?>
     <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
               Width="90%">
        <TR BgColor="#400040">
           <TD Width="15%"><INPUT Name="cmdDelete"
               onClick="setDelMode();" Type="button"
               Value="Delete"></TD>
           <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
           <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
        </TR>
```

```
        <TR BgColor="#E4E4E4">
            <TD><INPUT Name="chk1" Type="checkbox" Value="1"></TD>
            <TD><A HRef="JavaScript:setEditMode('1', '<?=$txtCatName;?>',
                '<?=$txtCatRmrk;?>')"><?echo($txtCatName);?></A></TD>
            <TD><?echo($txtCatRmrk);?></TD>
        </TR>
        </TABLE><BR>
<?   }  ?>
    </TD>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends. -->
</BODY>
</HTML>
```

The contents of the file **mstrscript.js** is as follows:

```
/*
  Date :- 13/05/05
  Author :- Hansel Colaco
  Filename :- mstrscript.js
  Purpose :- Defines Generic JavaScript functions for trimming
  strings and changing string case.
*/
/* Generating the function used for trimming string values. */
  function strtrim() {
  /* Returns the string passed as a parameter after removing
  blank spaces, if any, at both the beginning and the end of the
  parameter. */
     return this.replace(/^\s+/,'').replace(/\s+$/,'');
  }
/* Generating a global alias name for the function used for
trimming string values. */
  String.prototype.trim = strtrim;

/* The chngCase() function changes the first alphabet to an upper
case character for the value held in the field passed as the
first parameter. The second parameter is a reference to the form
holding the field of the first parameter. This function is called
when a form cursor moves away from a field. */
  function chngCase(pFld, pFrm) {
  /* Using the eval() function to extract the value held in the
  field passed as a parameter. */
     var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
  /* Removing extra spaces for the value extracted above and
  restoring into the same variable. */
     mFldVal = mFldVal.trim();
```

```
  /* If the variable containing the extracted value is not empty.
  */
     if (mFldVal != "") {
     /* Changing the string Proper case. */
        mFldVal = mFldVal.charAt(0).toUpperCase() + mFldVal.substr(1,
                  mFldVal.length);
     }
  /* Using the eval() function to display the new value in the
  field passed as a parameter. */
     eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
  }
/* The formDeleteValues() function generates a string of
comma-separated identities of the records selected for deletion.
If no records are selected, a message indicates the same. The
parameters for this function are references to a hidden variable
(used for storing the generated string) and the form
respectively. It is called when the DELETE button is clicked. */
  function formDeleteValues(pHid, pFrm) {
  /* Initializing variables for later use. */
     var selValues = "";
     var firstSelBox = "";
  /* Using the eval() function to create a variable to hold a
  reference to the form object. */
     eval("frm = document." + pFrm);
  /* Iterating through every object on the form. */
     for (i=0; i<frm.elements.length; i++) {
     /* If the current object is a Checkbox. */
        if (frm.elements[i].type == "checkbox") {
        /* If the variable firstSelBox is empty. */
           if (firstSelBox == "") {
           /* Assigning the element number, of the current form
           object, to the variable firstSelBox. */
              firstSelBox = i; }
           /* If the current Checkbox has been selected, (i.e.
           checked). */
           if (frm.elements[i].checked == true) {
           /* Assigning the element number, of the current form
           object, to the variable selValues. */
              selValues = selValues + frm.elements[i].value + ",";      }
        }
     }
  /* If the variable selValues does not holds a value. */
     if (selValues.length < 1) {
     /* Displaying a message indicating to select a checkbox. */
        alert('Please choose records you wish to delete.');
     /* Using the eval() function to place the form cursor on the
     first checkbox. */
        eval("document." + pFrm + ".elements[" + firstSelBox + "].focus();");
     }
```

```
/* If the variable selValues holds a value. */
    else {
    /* Removing the last character (i.e. a comma) for the value
    held in the variable selValues. */
        selValues = selValues.substring(0, selValues.length-1);
    /* Using eval() function to assign a value to the form's
    hidden variable. */
        eval("document." + pFrm + "." + pHid + ".value = '" + selValues + "';");
        frm.submit();
    }
}
```

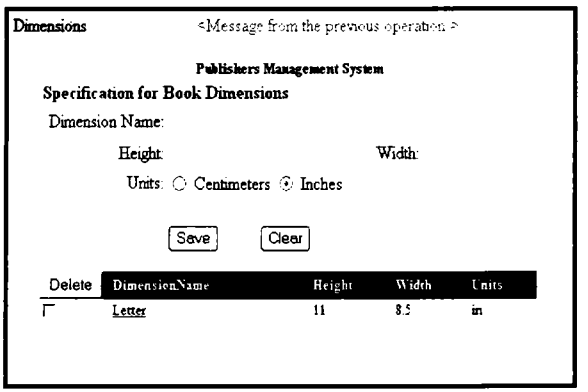


Diagram 12.10: Output for DmsnMstr.php.

Hands On Exercises

1. Develop a form to capture information of Book Dimensions. The appearance of the form is as shown in diagram 12.10.

- Design an HTML form having fields to capture information such as Dimension Name and its height, width and measurement units
- Write JavaScript code blocks to perform the following validations:
  - Remove extra spaces from either side of the value entered in a field
  - Change the contents in a field to title case (i.e. first alphabet of every word should be capitalized)
  - Make data entry mandatory for every field on the form
  - Check for numeric values in the height and width fields
  - Populate the form fields when specific data is selected for editing, (i.e. the hyperlink in the column **Dimension Name** is clicked)
  - Clear the form fields when Clear button is clicked
  - Return values, captured by the form field, to the same page when Save button is clicked
- Develop a PHP module to perform the following operations:
  - Access information captured via the form fields
  - Display information captured by generating a tabular layout below the HTML form
- When the text below the column labeled Dimension Name is clicked, the value appearing in that row should be transferred to the form fields for modification
- When the Delete button is clicked, data in the tabular layout should be permanently erased
- Enhancement of the page by extracting generic JavaScript code to an external file

SECTION III: WORKING WITH PHP

Basics Of File Handling

Handling files in PHP is quite simple. Form (or any other Data) can be streamed to an ASCII file on the web server's hard using a set of built in functions in PHP. This can be put to considerable use in the development of Web applications. File Handling in PHP is not limited only to the reading of ASCII files but also writing to and/or modifying them.

Checking For A File's Existence And Its Size Verification

Always determine the existence of a file before attempting to work with it. To accomplish this two functions are particularly useful: file_exists() and is_file().

file_exists()

The **file_exists()** function accepts a path and filename as a parameter and then searches for the existence of that file, returning **true** if it does exist and **false** if it does not.

Syntax:

```
bool file_exists(<[Path And] Filename Expressed As A String>)
```

Verifying the existence of a file example:

```
$filename = "sharanam.txt";
if (! file_exists($filename))
    print "File $filename does not exist!";
```

is_file()

The **is_file()** function will return true if file exists **and** is a readable/writable file. Essentially, **is_file()** is a bulletproof version of **file_exists()**, verifying not only the file's existence but also whether it can be read from or written to.

Syntax:

```
bool is_file(<[Path And] Filename Expressed As A String>)
```

Verifying the existence and validity of a file syntax example:

```
$file = "shah.txt";
if (is_file($file)) {
    print "The file $file is valid and exists!";}
```

```
else {
    print "Either $file does not exist or it is not a valid file!";        }
```

Once verified that the desired file exists and is capable of having various I/O operations performed on it, it can then be manipulated as appropriate.

Opening A File

The **fopen()** function is used to open files in PHP.

If the file exists, the **fopen()** function will open the file and return an integer, better known as the **file handle**.

Syntax:

```
int fopen(string <FileName>, string <Mode> [, int <UseIncludePath>])
```

The first parameter of this function contains the name of the file to be opened and the second parameter specifies in the mode in which the file should be opened:

```
<HTML><BODY>
<?php    $f=fopen("hansel.txt"."r");    ?>
</BODY></HTML>
```

The file may be opened in one of the following modes:

Modes	Description
R	Read only. File pointer at the start of the file
r+	Read/Write. File pointer at the start of the file
W	Write only. Truncates the file (overwriting it). If the file doesn't exist, fopen() will try to create the file
w+	Read/Write. Truncates the file (overwriting it). If the file doesn't exist, fopen() will try to create the file
A	Append. File pointer at the end of the file. If the file doesn't exist, fopen() will try to create the file
a+	Read/Append. File pointer at the end of the file. If the file doesn't exist, fopen() will try to create the file
X	Create and open for write only. File pointer at the beginning of the file. If the file already exists, the fopen() call will fail and generate an error. If the file does not exist, fopen() will try to create it
x+	Create and open for read/write. File pointer at the beginning of the file. If the file already exists, the fopen() call will fail and generate an error. If the file does not exist, fopen() will try to create it

REMINDER

If **fopen()** is unable to open the specified file, it returns **0** (false).

The following example prints a message if **fopen()**is unable to open the specified file:

```
<HTML><BODY>
<?php
  if (!($f=fopen("colaco.txt","r")))
      exit("Unable to open file!");
?>
</BODY></HTML>
```

Closing A File

The **fclose()** function is used to close a file. The fclose() function closes the file designated by filepointer, returning **true** on success and **false** otherwise

Syntax:

```
int fclose (int <FilePointer>)
```

Example:

```
fclose($f);
```

Reading From A File

The ability to read from a file is of obvious importance. The following are a set of functions geared toward making file reading an efficient process. It will be seen that the syntax of many of the functions are almost replicas of those used for writing.

The **feof()** function is used to determine if the end of file is true.

REMINDER

Files opened in w, a, and x mode cannot be read.

The **fread()** function reads up to length bytes from the file designated by file-pointer, returning the file's contents.

Syntax:

```
string fread (int <FilePointer>, int <Length>)
```

The file pointer must point to an opened file that is readable. Reading will stop either when length bytes have been read **or** when the end of the file has been reached.

Example:

```
$fh = fopen('vaishalishah.txt', "r") or die("Cannot open the file!");
$file = fread($fh, filesize($fh));
print $file;
fclose($fh);
```

The function **filesize()** is used to retrieve the byte size of vaishalishah.txt. This will ensure that fread() will read in the entire contents of the file.

Reading A Character

The **fgetc()** function returns a string containing one character from the file identified by its filepointer or returns false on reaching the end of file.

Syntax:

```
string fgetc (int <FilePointer>)
```

REMINDER

After a call to this function the file pointer moves to the next character in a file.

The example below reads a file character by character, until the end of file (i.e. the **feof()** function) is true:

```
<?php
  if (!($f=fopen("sharanamshah.txt", "r")))
      exit("Unable to open the file.");
  while (!feof($f)) {
      $x=fgetc($f);
      echo $x;
  }
  fclose($f);
?>
```

Writing To A File

The **fwrite()** function will simply write the contents of a string to the file identified by its filepointer.

Syntax:

```
int fwrite (int <FilePointer>, string <String> [, int <Length>])
```

If the option, **input parameter length** is provided, writing will stop either after length characters have been written **or** after the end of string has been reached whichever is earlier.

The following example shows how to write to a file:

```
<?
  $data = "Sharanam Shah. Analyst Designer. Silicon Chip Technologies.";
  $filename = "mybiodata.txt";
// If file exists and Is writable
  if(is_writeable($filename)) {
  // Open file and place file pointer at end of file
      $fh = fopen($filename, "a+");
  // Write $data to file
      $success = fwrite($fh, $data);
  // Close the file
      fclose($fh);
  }
  else {
      print "Could not open $filename for writing"; }
?>
```

REMINDER

The function **fputs()** is an alias to **fwrite()** and can be used by substituting the function name **fwrite** with **fputs**.

Designing A Login Module With Flat File Support

The functionality of the Login module build in the earlier chapter, (refer Chapter 12: **Working With Web Pages** section on *Designing A Login Module In HTML And PHP*) is limited to accessing the Login information entered by the visitors. This is just one half of the actual functionality. The main purpose of a Login module is to authenticate visitors to the Website. This requires that the login name and password be compared with some predefined, valid, set of login information. Only when a match for both the login name and its corresponding password is found, the visitor is allowed access to other modules in a Website.

The most primitive method, of passing valid set of login information, is to hardcode these values into the Login module. These values are then compared with the information entered by the visitor. If the match is a success, the visitor is granted access to the remaining modules bound to the Website. If the match fails, the visitor is expected to re-login until valid Login information is entered.

Although the above technique is the simplest, hard coding values restricts functionality of the module. In the case of the Login module, the number of valid Logins permitted is limited to the LoginID and password sets mentioned in the PHP code, which performs the authentication. If new Login information needs to be added the underlying code spec, which performs the validation must be changed to incorporate the new login information. Such frequent code changes are really not desirable.

To overcome the above problem a different technique is used. This requires that the set of valid login information be stored **external** to the PHP authentication code block. This means a system to store such information will be required. The storage system chosen here is a simple flat file, which stores information in ASCII (**i.e.** text) format. Each set of information comprises of a string with either **comma**-separated (,) or **tab**-separated values. The separator distinguishes individual segments of information held in a single set. For example, a set of login information would include a login name, a password and some description of the person who logs in.

PHP is quite effective when used with the above technique, due to its file handling capabilities. The illustration that follows describes how PHP's file handling techniques can be used to generate web pages based on the contents of an ASCII file.

Authenticating Login Information With Data From Flat Files

The next illustration describes the technique by which data from the flat file (login.txt) is used to perform authentication.

For the purpose of the illustration, create a copy of the **login.html**, **loginscript.js** and **logincheck.php** files from **Chap12_Cds** directory *and store them in the directory named* ***Chap13_Cds*** *under the* ***sct.phptraining.com*** *domain.*

Now modify the **logincheck.php** file as mentioned below:

```
<?
/*Date :- 21/05/05
  Author :- Hansel Colaco.
  Filename :- readlogin.php
  Purpose :- Reads a file (stores visitor's login information)
             and displays its contents.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco';
/* Storing the name of the file to be read. */
  $filename = "login.txt";
/* Storing the system date in the format 1st Jan, 2000. */
  $sysdt = date("jS M, Y");
/* Storing the system time in the format 00:00. */
  $systm = date("H:i");
/* A variable named $match stores the value to indicate a match
for login information; 0 - match not found, 1 - username match or
2 - Login information match. */
  $match = 0;
/* Checking if the variable either username or password is empty.
This means that the visitor has managed to submit a Login form
with empty fields. */
  if(empty($txtUserName) || empty($txtPassword)) {
```

```
    $Message = '<FONT Color="red"><B>Login Information Not
                Submitted!</B></FONT>';      }
else {
/* Using the function file_exists() to check whether PHP is
able to locate the login.txt file. */
    if (file_exists($filename)) {
    /* If the login.txt file is found, use the function file()
    to extract the contents of the login.txt file into a
    variable named $fileContents. */
        $fileContents = file($filename);
    /* Using an iteration to extract and transfer each line held
    in $fileContents into the variable $line. */
        foreach ($fileContents as $line_num => $line) {
        /* If the value of the variable $match is not 2. */
            if ($match != 2) {
            /* Using the function explode() to separate the
            segments of the current line and store them as
            individual elements for the array $segment. */
                $segment = explode("::", $line);

            /* If the first element (i.e. username) matches the
            login name entered by the visitor. */
                if ($txtUserName == $segment[0]) {
                    $Message = '<FONT Color="blue"><B>Login name
                                found!</B></FONT> <BR>';
                /* Changing the value held by $match to 1. */
                    $match = 1;

                /* If the second element (i.e. password) matches
                the password entered by the visitor. */
                    if($txtPassword == $segment[1]) {
                        $Message = $Message.'<FONT Color="blue"><B>With a
                                   valid password!! </B></FONT><BR>';
                    /* Changing the value held by $match to 2. */
                        $match = 2;
                    }
                /* If the second element (i.e. password) does not
                match the password entered by the visitor. */
                    else {
                        $Message = $Message.'<FONT Color="red"><B>But invalid
                                   password!! </B></FONT><BR>';    }
                }

            /* If the first element (i.e. username) does not match
            the login name entered by the visitor. */
                else {
                /* If the value held by $match is not equal to 1.
                */
                    if ($match != 1) {
```

```
                    $Message = '<FONT Color="red"><B>Login name not
                                found!</B> </FONT><BR>';}
                }
            }
        }
    }
  /* If PHP cannot locate the login.txt file. */
    else { $Message = '<FONT Color="red"><B>Unable to open file!</B>
                    </FONT>';  }
  }
?>
<!-- Generating a page to read contents of a file using a PHP
file handle. -->
<HTML>
<HEAD><TITLE>Login Authentication through A Flat File</TITLE></HEAD>
<BODY><?=$Message;?><BR><HR>
<!-- Presenting the login inform in a tabular layout. -->
  <H4>Details of the login attempt are:</H4>
  <TABLE Border="1" CellSpacing="1" Width="200"><TR>
      <TD Align="right" Width="75">Username :</TD>
      <TD Align="left" Width="100"><?="$txtUserName";?></TD>
  </TR><TR>
      <TD Align="right" Width="75">Password :</TD>
      <TD Align="left" Width="100"><?="$txtPassword";?></TD>
  </TR><TR>
      <TD Align="right" Width="75">Server Date :</TD>
      <TD Align="left" Width="100"><?="$sysdt";?></TD>
  </TR><TR>
      <TD Align="right" Width="75">Server Time :</TD>
      <TD Align="left" Width="100"><?="$systm";?></TD>
  </TR></TABLE><HR>
</BODY>
</HTML>
```

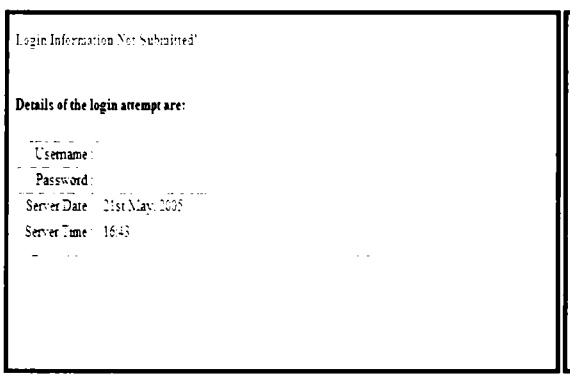


Diagram 13.1.1: Message by the logincheck.php file, displayed when the login information is not submitted.

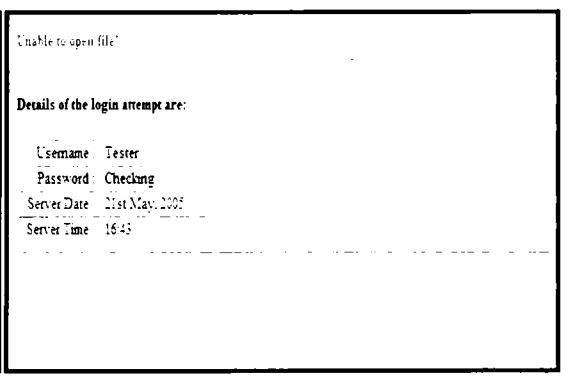


Diagram 13.1.2: Message by the logincheck.php file, displayed when the login.txt file cannot be located.

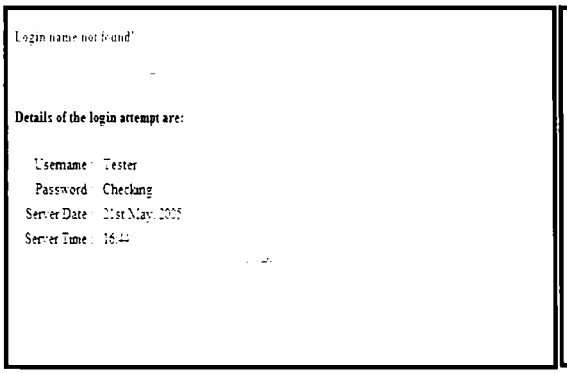


Diagram 13.1.3: Message by the logincheck.php file, displayed when the login name is invalid.

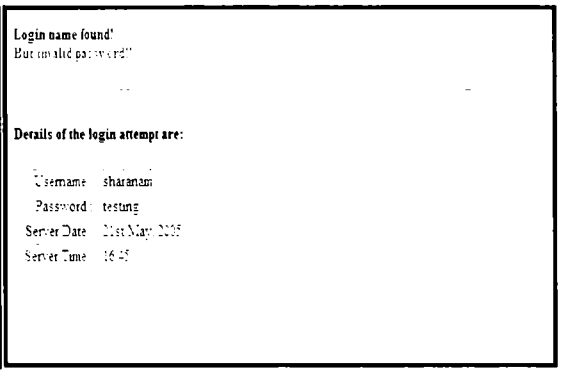


Diagram 13.1.4: Message by the logincheck.php file, displayed when only the password is invalid.

The above example uses the PHP file handling technique to authenticate a visitor. It compares the login information with the contents of the **login.txt** file to determine whether the visitor has a genuine login or not. The processing performed by the above page is as follows:

- A variable **$match** is declared and assigned to hold the value **0** (zero)
- A check is made on the username and password submitted by the visitor. If either one is found empty, an error message indicating an invalid login is generated and stored into a variable named **$Message**. Refer diagram 13.1.1
- If a value has been passed for both the username and password, an attempt is made to access the **login.txt** file. The PHP function **file_exits()** checks whether the file exists or not. If the file is not found, an error message indicating a missing file is generated and stored into a variable named **$Message**. Refer diagram 13.1.2
- If the file is found, the **file()** function is used to extract contents from the **login.txt** file. The extracted lines from the file are stored in the variable named **$fileContents**
- A **foreach** loop is constructed to iterate through each line held in **$fileContents**, (**i.e.** contents of the **login.txt** file). Line by line the contents of the login.txt held by the **$fileContents** is transferred to a variable named **$line** on every iteration
- A check is made to know whether the variable **$match** holds a value other than **2**. If the condition is satisfied, the value held in the variable **$Message** is cleared
- The PHP function **explode()** is used to split the value of the variable **$line**. A pair of colon characters (**::**) are used as the separator and each segment is transferred to individual elements of the array object **$segment**
- A comparison is made between the first element of **$segment** and the **username** passed in the login information. If a match is found:
  - A message indicating a valid login name is generated and stored into the variable **$Message**. Refer diagram 13.1.5

- The value in the variable **$match** is change to hold **1**
- A comparison is made between the second element of **$segment** and the **password** submitted in the login information
- If a match is found, a message indicating a valid password is generated and appended to the value previously stored into the variable **$Message**. Refer diagram 13.1.5
- If a match is not found, an error message indicating an invalid password is generated and appended to the value previously stored into the variable **$Message**. Refer diagram 13.1.4

- If the comparison between the first element of the variable **$segment** and the username fails, an error message indicating login name not found is generated and stored into the variable **$Message**. Refer diagram 13.1.3
- Finally, an HTML page is generated to display the value held by the variable **$Message** and the details of the login information.

The **logincheck.php** file retrieves and displays data on the page. But when it attempts to open the **login.txt** file, an error occurs, as the file is not found. An error message is generated as shown in diagram 13.1.2.

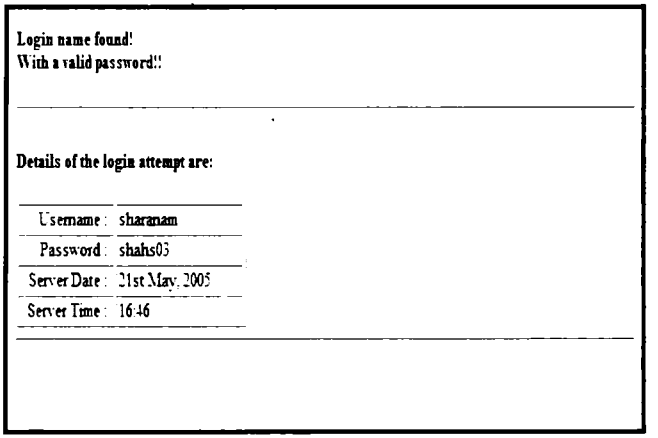


Diagram 13.1.5: Output for logincheck.php, when valid login information is passed to it.

Prevent the above error by creating the **login.txt** file. Populate the new file with the following contents:

```
sharanam::shahs03::03/01/1981
hansel::hclocf::01/01/1982
ivan::sct2306::23/06/1952
```

As seen above each line in the **login.txt** file consists of three segments, each separated by the pair of colons (**::**). Each entry comprises of a login name, a password and the date of birth of the person with the corresponding login name.

Now, re-run the above test by submitting data to the **logincheck.php** file. This time the **login.txt** file is found.

Using PHP To Write To Flat Files

Having dealt with reading from flat files, use this technique to display data in tabular layout in the Category Master form.

Skeleton Of The Code Undertaken For Development

To understand the development of the Category master form, glance through the pseudo structure for the same, as shown below:

```
<?
```

1 Commented statements providing a description and purpose of the file.

```
/*
  Date :- <Creation Date >
  Author :- <Developer's Name>
  Filename :- <Filename>
  Purpose :- <Brief Description About File Functionality>
*/
```

2 Common PHP variables.

```
/* A variable holding the developer's name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Category Master Form";
```

3.1 PHP variables bound to Flat File.

3.2 PHP variables bound to Processing Control.

4 PHP code block to establish Flat File access.

5 PHP code block for data retrieval on establishing access on Flat File.

5.1 PHP code block to perform the INSERT operation.

```
    /* Verifying that the form is currently in the INSERT mode.
    */
       if ($hidMode == "I") {
```

5.1.1 PHP code block controlling INSERT operation based on duplication.

5.2	PHP code block to perform the UPDATE operation.

```
/* Verifying if the form is currently in the UPDATE mode. */
   if ($hidMode == "U") {
```

5.2.1	PHP code block controlling UPDATE operation based on duplication.

5.3	PHP code block to perform the DELETE operation.

```
/* Checking if the form is in the delete mode. */
   if ($hidMode == 'D') {
```

5.3.1	PHP code block to retrieve multiple records selection for deletion.

6	PHP variables re-initialization bound to the page processing/layout.

```
/* A variable named $hidMode is initialised to 'I'. This variable
holds the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
?>
```

7	HTML code block rendering the actual page begins.

```
<HTML>
<HEAD>
  <TITLE><?=$title;?></TITLE>
```

7.1	META tags bound to the HTML page.

7.2	SCRIPT tag holding JavaScript under the HEAD section.

```
</HEAD>
```

8	BODY section.

```
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
```

9	SCRIPT tag holding JavaScript unique to the current page under the BODY section.

9.1	JavaScript function to check contents of form field before submitting data contained in them.

```
/* The chkBlanks() function verifies that the form field of
category name is not left empty. If the field is empty or
contains blank spaces, a message indicates to enter a valid
value. If the field is not empty, the data is submitted. */
   function chkBlanks() {
```

9.2 JavaScript function to validate contents of form field before submitting data contained in them.

```
/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
    function vldtFrmFlds() {
```

9.3 JavaScript function to prepare the form to accept a new set of data.

```
/* The function setNewMode() is called to prepare the form to
except a new set for data. This function is called directly
when the Clear button is clicked. */
    function setNewMode() {
```

9.4 JavaScript function to prepare the form to accept changes in the selected set of data.

```
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
    function setEditMode(id, name, rmrk) {
```

9.5 JavaScript function to prepare the form to delete selected set(s) of data.

```
/* The function setDelMode() is called to prepare the form to
remove data from the tabular layout. This function is called
directly when the Delete button is clicked. */
    function setDelMode() {
```

9.6 SCRIPT tag holding JavaScript under the BODY section ends.

```
</SCRIPT>
```

10 Parent Table begins.

```
<!-- Outer Table Code Begins. -->
  <TABLE  Align="center"  BgColor="#E4E4E4"  Border="0"  CellPadding="0"
CellSpacing="0" Name="TlbOuter" Width="90%">
```

10.1 Parent Table → First Row:
Columns to hold page title and a message header.

Category	<Message from the previous operation>

Diagram 13.2.1

11	**Form Section holding Data and Control objects begins.**

```
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="<FileName>" Method="post" Name="frmMstrCat"
        onSubmit="return chkBlanks();">
```

11.1	**Form Hidden Variable to identify the current page.**

```
  <!-- Declaring a hidden form field used to identify the current
  (i.e. category master) page. -->
     <INPUT Name="hidPage" Type="hidden" Value="<Page Name>">
```

11.2	**Form Hidden variables to maintain a copy of the data held by the form fields just before changes attempted. This is useful to confirm change(s) made to the form fields before an update is fired. (To avoid database or flat files concern as the case may be)**

```
  <!-- Declaring hidden form fields required for data validation.
  -->
     <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
     <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
     <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
```

11.3	**Form Hidden Variable to determine the form operation. Default, being Insert operation.**

```
  <!-- Declaring a hidden form field used for determining the
  form mode. It holds 'I' for Insert, 'U' for Update and 'D' for
  Delete. It will hold 'I' when the page is rendered for the 1st
  time. -->
     <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
```

11.4	**Form Hidden Variable holding entries (checkbox values) selected for deletion.**

```
  <!-- Declaring a hidden form field used for identifying records
  which have been selected for deletion. -->
     <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
```

12	**Parent Table → Second Row:** **Columns to hold the actual d/e form.**

```
  <TR Height="300" VAlign="top">
     <TD Align="center" Border="1" ColSpan="10"><BR>
```

13	**First Child Table holding d/e Form objects, to Capture / Control data begins.**

```
    <!-- Form Table Code Begins. -->
       <TABLE  Align="center"  Border="0"  CellPadding="2"  CellSpacing="0"
Name="TlbInner" Width="90%">
```

13.1	First Child Table → First and Second Rows: Columns to hold form title.

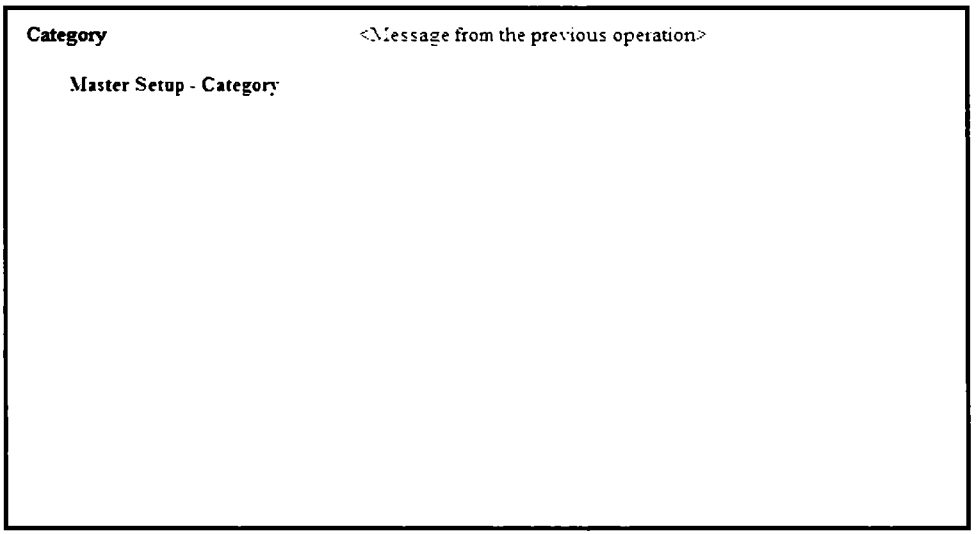


Diagram 13.2.2

13.2	First Child Table → Third & Forth Rows: Columns holding label and data capture objects to capture data.

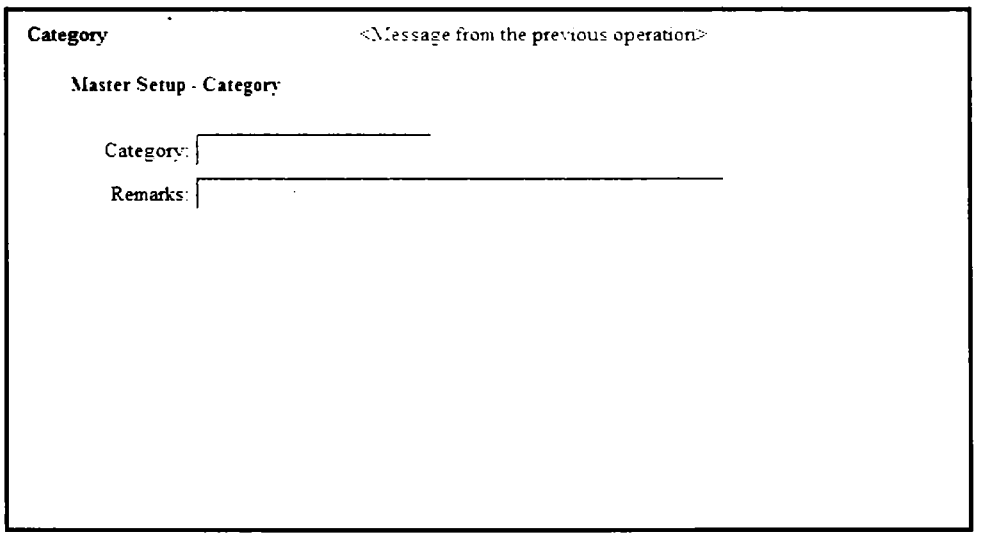


Diagram 13.2.3

13.3	First Child Table → Last Row: Columns holding data control objects.

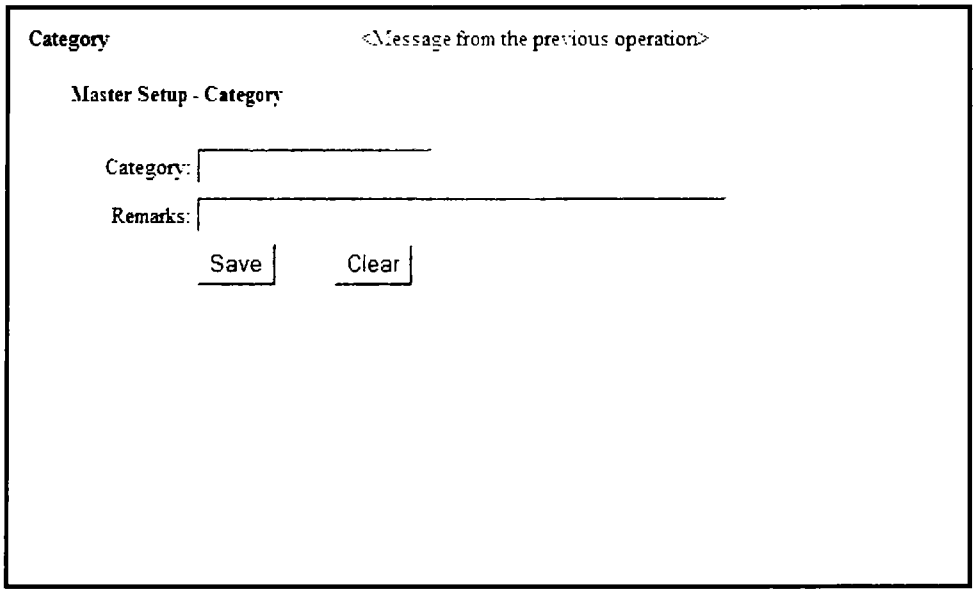


Diagram 13.2.4

13.4	First Child Table ends.

```
<?
```

14	PHP code block retrieving data to create the tabular layout by File access. The tabular layout is generated only after data gets retrieved.

```
?>
```

15.1	Second Child Table holding the tabular layout, to display / control data captured begins.

```
<!-- Data Table Code Begins. -->
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
        Width="90%">
```

15.2	Second Child Table → First Row: Columns to header row.

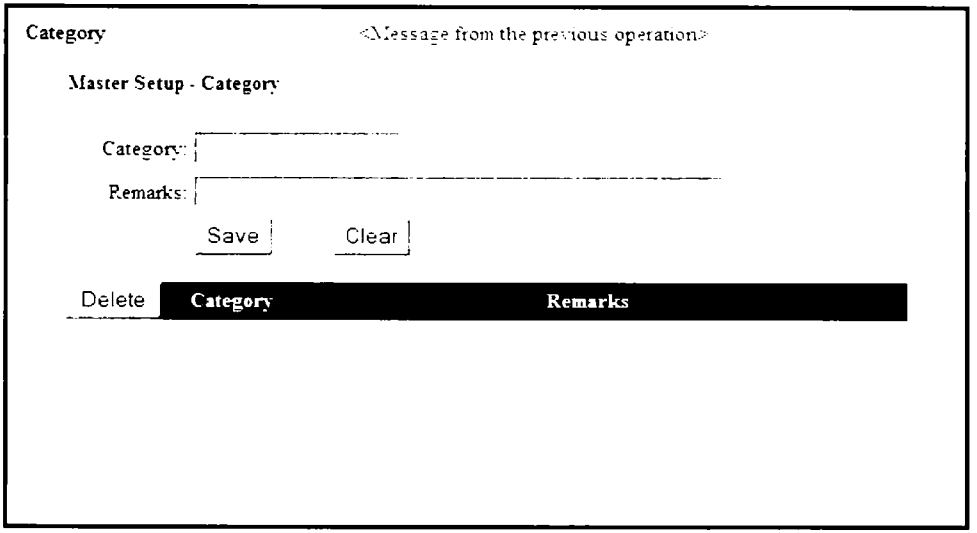


Diagram 13.2.5

```
<?
```

15.3 PHP code block to populate data in the tabular layout using a loop.

```
/* Using an iteration to extract and transfer each line held in
$fileContents into the variable $line. */
   foreach ($fileContents as $line_num => $line) {
```

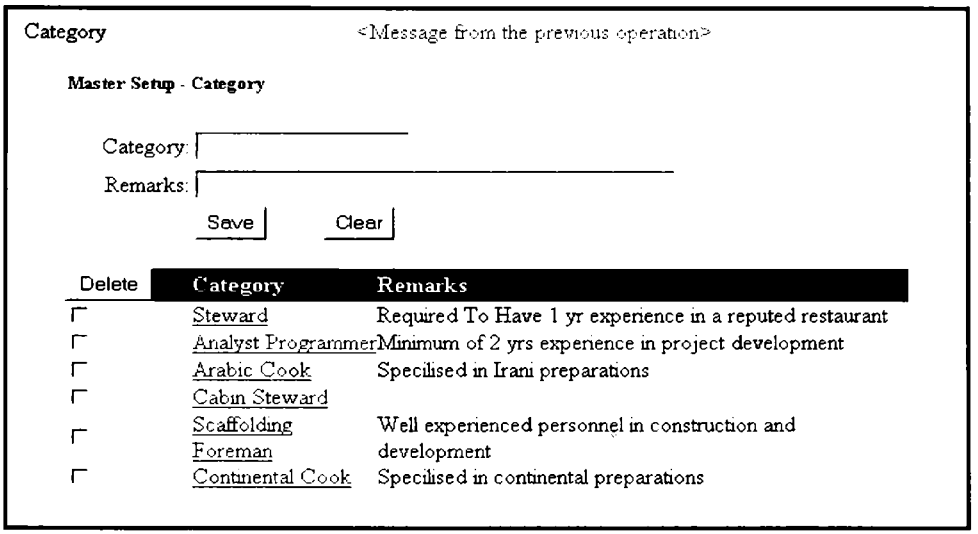


Diagram 13.2.6

15.4	Loop and the Second Child table ends.

```
<?      } ?>
        </TABLE><BR>
```

16	Form holding Data and Control objects ends.

```
</FORM>
```

17	Parent Table ends.

```
</TABLE>
<!-- Outer Table Code Ends. -->
```

18	HTML code block along with BODY section rendering the actual page ends.

```
</BODY>
</HTML>
```

Consider the contents of **ctgrymstr.txt** file, which contains data related to the various employment categories:

```
1:Steward:Required To Have 1 yr experience in a reputed restaurant
2:Analyst Programmer:Minimum of 2 yrs experience in project development
3:Arabic Cook:Specialised in Irani preparations
4:Cabin Steward:
5:Scaffolding Foreman:Well experienced personnel in construction and maintenance
6:Continental Cook:Specialised in Continental preparations
```

To be able to access and display the above information, the PHP page for the Category Master form needs modifications. To avoid rewriting the entire page, create a copy of the **ctgry_mstr.php** and **mstrscript.js** files *from the **Chap12_Cds** directory and store it in the **Chap13_Cds** directory*. Perform the changes as mentioned below:

- Delete the block of PHP code, which ensures that the form is in the delete mode (i.e. **$hidMode** holds D). This is done as the code spec for the delete operation will be built later. The block of PHP code to be deleted is found in the declaration section. (Refer points **2, 5.3 and 6** under the sub-title **Skeleton Of The Code Undertaken For Development.**)

```
...
/* A variable holding the title for the page. */
   $title = "Category Master Form";
/* If the form mode is set to Delete. */
   if ($hidMode == 'D') {
      /* Variable $Message is initialised to hold a message, which
      has occurred during the previous (delete) operation. */
         $Message = '<BR><FONT Color="RED">* Details for '.$hidDelRcrdLst.' has
                    been deleted.</FONT>';      }
```

```
/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
   $hidMode = 'I';
?>
...
```

- Declare a new variable after the declaration for the variable **$title**, (Refer point **3.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
/* A variable holding the developers name. */
   $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
   $title = "Category Master Form";
```

```
/* Storing the name of the file to be read. */
   $filename = "ctgrymstr.txt";
```

```
/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
   $hidMode = 'I';
?>
...
```

- Replace the code block placed after **
** tag following the **</TABLE>** tab for the inner table containing the element of the form and the **</TD>** for the parent column containing the **<FORM>** object. (Refer points **13.4, 14, 15.1, 15.2, 15.3 and 15.4** under the sub-title **Skeleton Of The Code Undertaken For Development.**) The contents of the new code will be as below:

```
                <INPUT Name="cmdReset" onClick="setNewMode();" Type="button" Value="Clear">
            </TD>
        </TR></TABLE><BR>
    <!-- Form Table Code End. -->
```

```
<?
/* Using the function file_exists() to check whether PHP is able
to locate the login.txt file. */
  if (file_exists($filename)) {
  /* If the ctgrymstr.txt file is found. */
?>
    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
               Width="90%">
        <TR BgColor="#400040">
            <TD Width="15%"><INPUT Name="cmdDelete"
                onClick="setDelMode();" Type="button"
                Value="Delete"></TD>
            <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
        </TR>
<?
  /* Use the function file() to extract the contents of the
  ctgrymstr.txt file into a variable named $fileContents. */
     $fileContents = file($filename);
```

```
  /* Using an iteration to extract and transfer each line held in
  $fileContents into the variable $line. */
     foreach ($fileContents as $line_num => $line) {
     /* Using the function explode() to separate the segments of
     the current line and store them as individual elements for
     the array $segment. */
        $segment = explode(":", $line);
?>
        <TR BgColor="#E4E4E4">
        <!-- Creating a check box using the value held by the
        first element of the array $segment. -->
           <TD><INPUT Name="chk<?=($segment[0]);?>" Type="checkbox"
                Value="<?=($segment[0]);?>"></TD>
        <!-- Creating a hyperlink using the value held by the
        second element of the array $segment. -->
           <TD><A HRef="JavaScript:setEditMode('<?=($segment[0]);?>',
                '<?=($segment[1]);?>', '<?=($segment[2]);?>')">
                <?echo($segment[1]);?></A></TD>
        <!-- Displaying the value held by the third element of
        the array $segment. -->
           <TD><?=($segment[2]);?></TD>
        </TR>
<?      } ?>
        </TABLE><BR>
<?   }
/* If PHP cannot locate the login.txt file. */
  else { ?>
  <FONT Color="red"><B>Unable to open file!</B></FONT>
<?
  /* Terminating the any further processes on the page. */
     exit();
  } ?>
```

```
     </TD>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends. -
->
</BODY>
</HTML>
```

The above modifications will allow the Category master page to display the contents of the **ctgrymstr.txt** file in a tabular layout. The processing of the new code block is as follows:

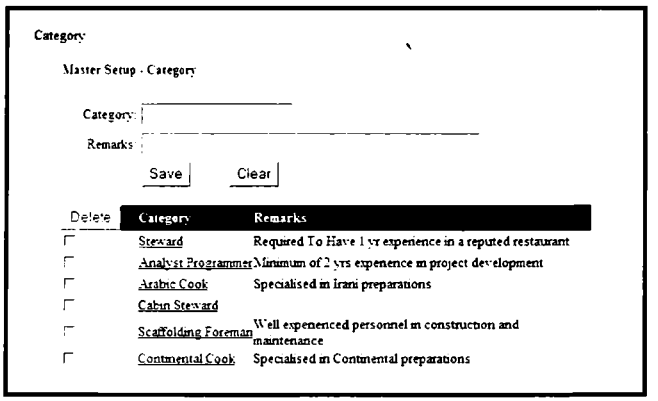


Diagram 13.3: Output for ctgry_mstr.php.

- A check is made whether the text file containing category information (**i.e. ctgrymstr.txt**) exist. If yes, a table with three columns is created using HTML technique.

The first row, being the header bar, contains the following:

- A button object [Delete] is placed in the first column
- The second is labeled **Category**
- The third column is labeled **Remarks**

- The **file()** function is used to extract each line in the **ctgrymstr.txt** file into a variable named **$fileContents**
- A loop is created to traverse through each line in **$fileContents**
  - The contents of the current line is transferred into **$line**
  - The **explode()** function is used to split the contents of the current line and store it into array named **$segment**. The splitting is done on the bases of a colon (**:**) and since there are two colons in each line, the array is populated with three elements
  - The data section is created using Table rows. The first column holds a check box named using the value held by the first element of the array $**segment**. The second column displays the hyperlink labeled as the category name held in the second element of the **$segment** array
  - The third column displays a description, if any. This is extracted from the third element of the array **$segment**)

*If the check for locating the text file (**ctgrymstr.txt**) fails, an error message is displayed and the process is terminated.*

Having made the above changes, save the file **ctgry_mstr.php** and call it from a web browser. The page rendered will appear as shown in diagram 13.3.

Storing Data Into Flat Files

The change brought in by the above code block is limited to displaying, (**i.e.** reading) data contained in a flat file. It does not add any features of inserting data. This means that data entered into the form fields will be lost.

The illustration below improvises the Category master page and allows it to use a flat file to store data (captured by the form fields). To continue with the illustration ensure that the flat file (**i.e. ctgrymstr.txt**) is editable. If not, give write permission to the file as:

```
System Prompt> chmod 777
                    /var/www/sct/phptraining/Chap13_Cds/ctgrymstr.txt ↵
```

In the **ctgry_mstr.php** file, insert the following PHP code block immediately after the declaration of PHP variables at the beginning of the file, (Refer points **3.2, 4, 5, 5.1 and 5.1.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
/* Storing the name of the file to be read. */
  $filename = "ctgrymstr.txt";
/* A variable holding the message to be displayed on the page. */
  $Message = "";
/* Storing the maximum record number assigned to an entry in the
text file. */
  $maxRcrd = 0;
/* Initialising a variable named $insert to determine whether to
continue the insert operation or not. If it holds 0, the insert
operation is cancelled. */
  $insert = 1;

/* If Category name has been passed through the variable
$txtCatName, (i.e. the $txtCatName is not empty). */
  if (!empty($txtCatName)) {
  /* Using the function file_exists() to check whether PHP is
  able to locate the ctgrymstr.txt file. */
     if (file_exists($filename)) {
     /* If the ctgrymstr.txt file is found. Use the function
     file() to extract the contents of the ctgrymstr.txt file
     into a variable named $fileContents. */
        $fileContents = file($filename);
     /* Using an iteration to extract and transfer each line held
     in $fileContents into the variable $line. */
        foreach ($fileContents as $line_num => $line) {
        /* Using the function explode() to separate the segments
        of the current line and store them as individual elements
        for the array $segment. */
           $segment = explode(":", $line);
        /* Checking if the form is in the insert mode. */
           if ($hidMode == 'I') {
           /* Checking for duplication in category, i.e. category
           name already exists in the text file. */
              if ($segment[1] == $txtCatName) {
              /* Assigning $insert with a 0 (zero), to cancel the
              insert operation. */
                 $insert = 0;
              /* Terminating the iteration used for traversing
              through the each line in the text file. */
                 break;
              }
           /* If a match for the new category name was not found
           in the text file. */
              else {
              /* Checking if the current entry number is greater
              than the value held in $maxRcrd. */
                 if ($segment[0] > $maxRcrd) {
```

```
                    /* Storing the highest entry number into the
                    variable $maxRcrd. */
                        $maxRcrd = $segment[0];
                    }
                }
            }
        }
    }
  }
/* Checking if the form is in the insert mode. */
  if ($hidMode == 'I') {
  /* Checking the value held by $insert. If zero the insert
  operation is cancelled. */
     if ($insert == 0) {
     /* Generating and storing an error message which indicates a
     duplication of category. */
        $Message = '<FONT Color="red">Entry for <B>'.$txtCatName.'</B>
                    already exists. </FONT>';    }
  /* If the insert operation is permitted. */
     else {
     /* If the ctgrymstr.txt file is editable, a file handle is
     created using the fopen() function. The handle is created
     using the 'a' switch to allow data to be appended to the
     text file. */
        if (is_writable($filename) and $fileHandle = fopen($filename, 'a')) {
        /* Storing the next highest entry number into the
        variable $newRcrd. */
           $newRcrd = $maxRcrd + 1;
        /* Concatenating a new line character to a string that
        will be appended into the text file and storing it into
        the variable $append. */
           $append = "$newRcrd:$txtCatName:$txtCatRmrk".chr(13).chr(10);
        /* Attempting to append a line into the text file and
        checking if the write operation was successful. */
           if(!fwrite($fileHandle, $append) === FALSE) {
           /* Generating and storing a message that indicates the
           insert operation was successful. */
              $Message = '<FONT Color="blue">Entry for
                          <B>'.$txtCatName.'</B> added
                          successfully.</FONT>';      }
        /* If the write operation on the ctgrymstr.txt file
        fails. */
           else {
           /* Generating and storing an error message, which
           indicates a failure in insert the new category. */
              $Message = '<FONT Color="red">Failure in adding entry for
                          <B>'.$txtCatName.'</B>.</FONT>';       }
```

```
        /* Closing the file handle which is used to access and
        append the ctgrymstr.txt file. */
            fclose($fileHandle);
        }
    /* If data cannot be written to the file ctgrymstr.txt or an
    error is encountered while creating a file handle for the
    file. */
        else {
        /* Generating and storing an error message that indicates
        failure while accessing the text file. */
            $Message = '<FONT Color="red">Unable to access text file.</FONT>';
        }
    }
}
```

```
/* A variable named $hidMode is
initialised to 'I'. This
variable holds the form status
in terms of I - Insert, U -
Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

Save the changes made to the **ctgry_mstr.php** file and run it within the web browsers. When the page appears, enter a new category and click Save. Refer diagram 13.4.1.

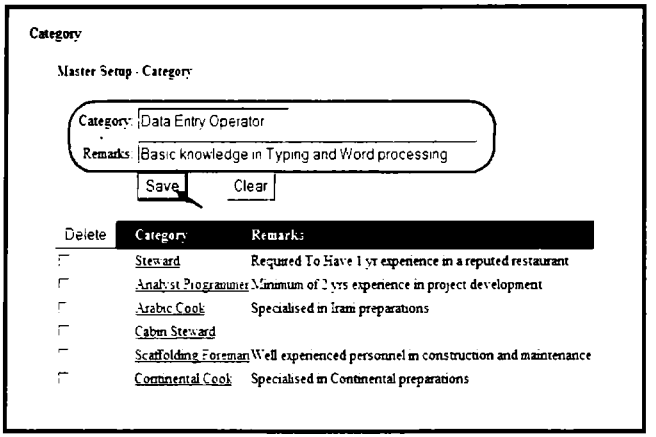


Diagram 13.4.1: Adding a new entry with category information via the ctgry_mstr.php file.

When a call is made to the **ctgry_mstr.php** file with new category information the following processes occur:

- Variables such as $filename, $Message, $maxRcrd, $insert are initialized
- A check is made to ensure that category name field holds data
- If it holds a value, a check is made to ensure that the **ctgrymstr.txt** file exists to write data to it. If the file exits:
  - The file contents are transferred into a variable **$fileContents**
  - The **file()** function is used to extract each line in the **ctgrymstr.txt** file into a variable named **$fileContents**
  - A loop is created to traverse through each line (Record) in **$fileContents**. This loop will hold the code spec for identifying duplicate records during the insert or the update mode and perform the following:
    - The contents of the current line (Record) is transferred into variable named **$line**

- The **explode()** function is used to split the contents of the current line and store it into an array named **$segment**. The splitting is done on the bases of a colon (:) and since there are two colons in each line, the array is populated with three elements

The processes defined above are applicable for insert and update operations

- A check is made to verify whether the form mode is insert. If the mode is insert then:
  - The newly entered record details (in this case, the category name) is checked for duplication against the data retrieved from the file. If the category name is being duplicated a flag named **$insert** is set to **0** (zero). This simply indicate that the record will not be inserted into the file
  - If there is no duplication then the highest category identity is extracted

The processing which follows the above code is strictly bound to the insert mode. This means that the actual insert operation will be performed here. As a result, a check is required to ensure that the form is in the insert mode before continuing:

- The flag **$insert** is used to determine the insert operation. If it holds **0** (zero):
  - An error message indicating that *the record is being duplicated* is generated and stored into **$Message**
  - The insert operation terminates
- If **$insert** holds **1** (No duplication):
  - A check is made to ensure that the **ctgrymstr.txt** file is writable in terms of file permissions.
  - A file handle is returned by the **fopen()** function. The file is opened using the '**a**' switch to allow data to be appended to the text file
    - The data captured is now appended to the file along with the category identity generated earlier
    - The **fwrite()** function is called to perform the insert operation. Depending on the success or failure of the write operation, a message indicates the same is generated and stored into **$Message**
    - The **fclose()** function is used to release the file handler bound to ctgrymstr.txt file
- If the **ctgrymstr.txt** file is not found, an error message is generated and stored into **$Message**, which will be rendered later
- Finally, the previous functionality of **ctgry_mstr.php** is processed. This will include:
  - Displaying the value stored in **$Message**
  - Laying out the form for capturing category information
  - Laying out the grid for displaying category information stored in **ctgrymstr.txt** file

The new category will be listed as the last entry in the tabular layout / grid. Refer diagram 13.4.2.

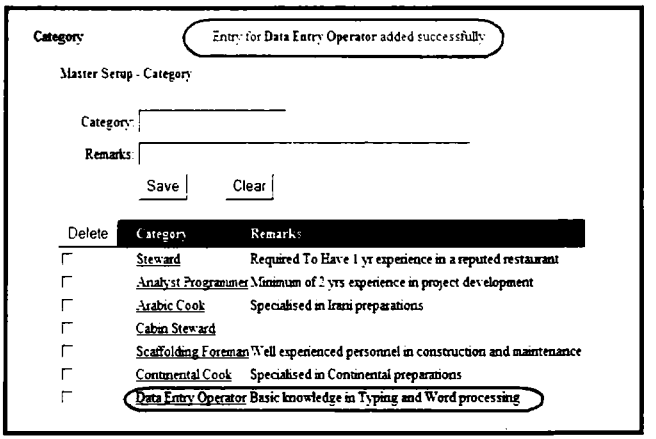


Diagram 13.4.2: New entry added by ctgry_mstr.php.

Modifying Data Stored In Flat Files

Having completed the operation for inserting new sets of information into the text file, the next step is to update existing information. As illustrated in the previous chapter (refer Chapter 12: **Working With Web Pages** section on *Updating Data From The Previous Form Instance*), clicking on the hyper-linked text (in the second column for the tabular layout) will invoke the form's update mode. The values bound to the entry, whose hyperlink is clicked, gets passed to the form fields permitting data modification.

When Save is clicked, the data held in the form fields gets submitted to the same page.

To illustrate the Update technique using flat file, the code in **ctgry_mstr.php** will be used as base. Insert / modify PHP code block as mentioned below:

- At the starting section of the file ctgry_mstr.php, insert the declaration of a PHP variable as, (Refer point **3.2** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
/* Storing the maximum record number assigned to an entry in the text file. */
  $maxRcrd = 0;
/* Initialising a variable named $insert to determine whether to continue the
insert operation or not. If it holds 0, the insert operation is cancelled. */
  $Insert = 1;
```

```
/* Initialising a variable named $update to determine whether to
continue the update operation or not. If it holds 0, the update
operation is cancelled. */
  $update = 1;
```

```
/* If Category name has been passed through the variable $txtCatName, (i.e. the
$txtCatName is not empty). */
  if (!empty($txtCatName)) {
  /* Using the function file_exists() to check whether PHP is able to locate the
  login.txt file. */
      if (file_exists($filename)) {
...
```

- At the point where access to the flat file for data retrieval is attempted, locate the check made for the form's insert mode. Insert the following code, immediately after the code block executed only when the form is in the insert mode (after the **explode()** function). (Refer point **5** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
        /* Using the function explode() to separate the segments of the current
        line and store them as individual elements for the array $segment. */
            $segment = explode(":", $line);
        /* Checking if the form is in the insert mode. */
            if ($hidMode == 'I') {
            /* Checking for duplication in category, i.e. category name already
            exists in the text file. */
                if ($segment[1] == $txtCatName) {       ...     }
            /* If a match for the new category name was not found in the text
            file. */
                else {
                    ... }
                }
            }
```

```
        /* Checking if the form is in the update mode. */
            if ($hidMode == 'U') {
            /* Checking if modified category name already exists
            as a different entry in the text file. */
                if ($segment[1] == $txtCatName
                    and $segment[0] != $hidCatID) {
                /* Assigning $update with a 0 (zero), to cancel the
                update operation. */
                    $update = 0;
                /* Terminating the iteration used for traversing
                through the each line in the text file. */
                    break;
                }
            }
```

```
        }
    }
}
/* Checking if the form is in the insert mode. */
  if ($hidMode == 'I') {
  /* Checking the value held by $insert. If zero the insert operation is
  cancelled. */
      if ($insert == 0) {
...
```

The above code spec is responsible to check for record duplication, caused due to the update operation.

- Insert the following code block immediately after the last PHP changes, done previously for the Insert operation, (Refer point **5.2 and 5.2.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
/* Checking if the form is in the insert mode. */
  if ($hidMode == 'I') {
  /* Checking the value held by $insert. If zero the insert operation is
  cancelled. */
      if ($insert == 0) {
      /* Generating and storing an error message which indicates a duplication of
      category. */
          $Message = '<FONT Color="red">Entry for <B>'.$txtCatName.'</B> already exists.
                     </FONT>';  }
  /* If the insert operation is permitted. */
      else {
      ...
      /* If data cannot be written to the file ctgrymstr.txt or an error is
      encountered while creating a file handle for the file. */
          else {
          /* Generating and storing an error message that indicates failure while
          accessing the text file. */
              $Message = '<FONT Color="red">Unable to access text file.</FONT>';}
      }
  }
```

```
/* Checking if the form is in the update mode. */
  if ($hidMode == 'U') {
  /* Checking the value held by $update. If zero the update
  operation is cancelled. */
      if ($update == 0) {
      /* Generating and storing an error message which indicates a
      duplication of category. */
          $Message = '<FONT Color="red">The modified category
                     <B>'.$txtCatName.'</B> already exists.</FONT>';  }
  /* If the update operation is permitted to continue. */
      else {
      /* If the ctgrymstr.txt file is found. Use the function
      file() to extract the contents of the ctgrymstr.txt file
      into a variable named $fileContents. */
          $fileContents = file($filename);
      /* If the ctgrymstr.txt file is editable, a file handle is
      created using the fopen() function. The handle is created
      using the 'w' switch to allow file data to be truncated,
      (i.e. the file holds nothing). */
          if (is_writable($filename) and $fileHandle = fopen($filename, 'w')) {
          /* Using an iteration to extract and transfer each line
          held in $fileContents into the variable $line. */
              foreach ($fileContents as $line_num => $line) {
              /* Using the function explode() to separate the
              segments of the current line and store them as
              individual elements for the array $segment. */
                  $segment = explode(":", $line);
              /* If the first element in $segment is not empty. */
                  if (!empty($segment[0])) {
```

```
            /* Checking whether the first element for $segment
            hold a value identical to record number for the
            entry selected for modification. */
               if ($segment[0] == $hidCatID) {
               /* Transferring values from the form fields into
               the second and third segments for the array
               object $segment. */
                  $segment[1] = $txtCatName;
                  $segment[2] = $txtCatRmrk;
               }
            /* Ensuring that the last element for $segment ends
            with a new line character. */
               if (ord(substr($segment[2], -1)) == 10) {
               /* Generating the string that will be appended
               into the text file and storing it into the
               variable $append. */
                  $append = "$segment[0]:$segment[1]:$segment[2]";
               }
            /* When the last element for $segment does not end
            with a new line character. */
               else {
               /* Concatenating a new line character to the
               string that will be appended into the text file
               and storing it into the variable $append. */
                  $append = "$segment[0]:$segment[1]:$segment[2]"
                            .chr(13).chr(10);     }
            /* Attempting to append a line into the text file
            and checking if the write operation was successful.
            */
               if(!fwrite($fileHandle, $append) === FALSE) {
               /* Generating and storing a message that
               indicates the update operation was successful.
               */
                  $Message = '<FONT Color="blue">Entry for
                              <B>'.$txtCatName.'</B> modified
                              successfully.</FONT>';      }
            /* If the write operation on the ctgrymstr.txt file
            fails. */
               else {
               /* Generating and storing an error message,
               which indicates a failure in update the new
               category. */
                  $Message = '<FONT Color="red">Failure in changing entry
                              for <B>'.$txtCatName.'</B>.</FONT>';
                    }
            }
         }
```

```
        /* Closing the file handle which is used to access and
        append the ctgrymstr.txt file. */
            fclose($fileHandle);
        }
    /* If data cannot be written to the file ctgrymstr.txt or an
    error is encountered while creating a file handle for the
    file. */
        else {
        /* Generating and storing an error message that indicates
        failure while accessing the text file. */
            $Message = '<FONT Color="red">Unable to access text file.</FONT>';
        }
    }
}
```

```
/* A variable named $hidMode is
initialised to 'I'. This
variable holds the form status
in terms of I - Insert, U -
Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

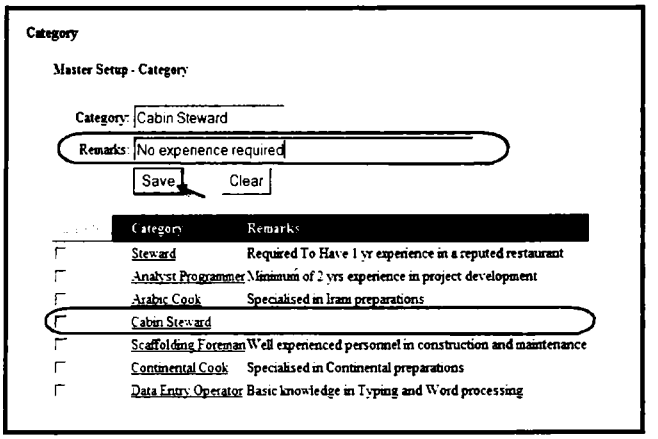


Diagram 13.5.1: Editing an entry with category information via the ctgry_mstr.php file.

Save the changes made to the **ctgry_mstr.php** file and run it within the web browsers. When the page appears click one of the hyperlinks representing the record requiring modifications in the tabular layout. This will populate the form fields with data bound to the selected entry, change the data as required and click Save. Refer diagram 13.5.1.

On submit, the control will be passed to the same page (i.e. the **ctgry_mstr.php** file) with the modified category information and the following processes occur:

- Variables such as $filename, $Message, $maxRcrd, $insert, $update are initialized
- A check is made to ensure that category name field holds data
- If it holds a value, a check is made to ensure that the **ctgrymstr.txt** file exists to write data to it. If the file exits:
  - The file contents are transferred into a variable **$fileContents**
  - The **file()** function is used to extract each line in the **ctgrymstr.txt** file into a variable named **$fileContents**
  - A loop is created to traverse through each line (Record) in **$fileContents**. This loop will hold the code spec for identifying duplicate records during the insert or the update mode and perform the following:

- The contents of the current line (Record) is transferred into variable named **$line**
- The **explode()** function is used to split the contents of the current line and store it into an array named **$segment**. The splitting is done on the bases of a colon (:) and since there are two colons in each line, the array is populated with three elements

The processes defined above are applicable for insert and update operations

- A check is made to verify whether the form mode is update. If the mode is update then:
  - The newly entered record details, in this case the category name, is checked for duplication against the data retrieved from the file. Additionally it is ensured that the category name duplication check is avoided against the same record. If the category name is being duplicated, a flag named **$update** is set to zero. This simply indicates that the record will not be updated into the file

Next, the code block bound to the insert operation is encountered and ignored as the form is in the update mode (i.e. the value held in **hidMode** is U). The processing which follows is strictly bound to the update mode and a check is made to ensure that the form is in the update mode before continuing as follows:

- The flag **$update** is used to determine the insert operation. If it holds **0** (zero):
  - An error message indicating that *the record is being duplicated* is generated and stored into **$Message**
  - The update operation terminates
- If **$update** holds **1** (No duplication):
  - Using the **file()** function, the entire contents of the file **ctgrymstr.txt** is transferred into a variable **$fileContents**
  - A file handle is created using the **fopen()** function. The handle is created using the '**w**' switch to allow data to be written to the text file. Using the **'w'** switch it truncates the contents of file opened. This means that the records will be updated in an array and finally transferred back to the file
  - A loop is created to traverse through each line (Record) in **$fileContents** and perform the following:
    - The contents of the current line (Record) is transferred into **$line**
    - The **explode()** function is used to split the contents of the current line (Record) and stored it into an array named **$segment**. The splitting is on the bases of a colon (:) and since there are two colons in each line, the array is populated with three elements
    - A check is made to verify whether the first element in **$segment** is empty. If it holds a value:

- The update operation is performed when the Category identity matches the current element of **$segment**
- A check is made to ensure that the last element of **$segment** ends with a new line character. Bases on the result of the check, the contents to be appended into the text file is generated and stored in a variable
- The **fwrite()** function is now called to apply the changes in terms of writing data back to the file. If the write operation fails, a message is generated and stored into **$Message** and finally rendered

- The **fclose()** function is called to release the file handler bound to **ctgrymstr.txt** file
- A message is generated to indicate the categories deleted and stored it into **$Message** which is later on rendered

- If the **ctgrymstr.txt** file is not found, an error message is generated and stored into **$Message**
- Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:
  - Displaying the value stored in **$Message**
  - Laying out the form for capturing category information
  - Laying out the grid for displaying category information stored in **ctgrymstr.txt** file

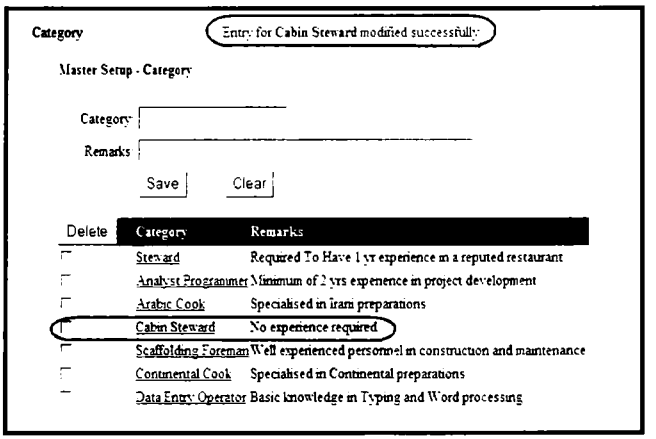


Diagram 13.5.2: Entry modified by ctgry_mstr.php.

The modified category will be listed as an entry in the tabular layout / grid. Refer diagram 13.5.2.

Deleting Data Stored In Flat Files

Having completed the data insert and the data manipulation operations using flat file, it is time to move on to the delete operation. As seen in the previous chapter (refer Chapter 12: **Working With Web Pages** section on *Deleting Data From The Previous Form Instance*), the delete functionality at the HTML level will remain the same. The change occurs in the PHP processing, when a list of record selected for deletion is received from the page's previous instance.

Earlier when Delete was clicked, a list of record(s) marked for deletion was created and the form elements were cleared. The list was then submitted to the next instance of the same page, displaying a message containing the list of deleted record(s).

Now the use of flat file will bring in changes in PHP processing. To understand the delete operation, perform the following modification in PHP code block:

- Insert the following code block immediately after the PHP code strictly for the update operation, (Refer points **5.3 and 5.3.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
/* Checking if the form is in the update mode. */
  if ($hidMode == 'U') {
  /* Checking the value held by $update. If zero the update operation is
cancelled. */
      if ($update == 0) {
      ...
      }
  /* If the update operation is permitted to continue. */
      else {
      ...
      }
  }
```

```
/* Checking if the form is in the delete mode. */
  if ($hidMode == 'D') {
  /* Using the function explode() to separate the entry number(s)
  stored in $hidDelRcrdLst and store them as individual elements
  of the array $delRcrd. */
      $delRcrd = explode(",", $hidDelRcrdLst);
  /* Use the function file() to extract the contents of the
  ctgrymstr.txt file into a variable named $fileContents. */
      $fileContents = file($filename);
  /* If the ctgrymstr.txt file is editable, a file handle is
  create using the fopen() function. The handle is created using
  the 'w' switch to allow file data to be truncated, (i.e. the
  file holds nothing). */
      if (is_writable($filename) and $fileHandle = fopen($filename, 'w')) {
      /* Initialising the variable $Rcrd to act as a counter for
      the entry number. */
          $Rcrd = 1;
      /* Using an iteration to extract and transfer each line held
      in $fileContents into the variable $line. */
          foreach ($fileContents as $line_num => $line) {
          /* Using the function explode() to separate the segments
          of the current line and store them as individual elements
          for the array $segment. */
              $segment = explode(":", $line);
          /* If the first element in $segment is not empty. */
              if (!empty($segment[0])) {
              /* A variable to act as a flag to determine whether
              the current entry is delete or not. If 1 the record is
              not deleted. */
                  $writeRcrd = 1;
```

```
/* An iteration through the element in the array
holding the list of entry number(s) marked for
deletion. */
    for($i=0; $i<=count($delRcrd); $i++) {
    /* Check if the current element in $delRcrd is
    empty. */
        if (!empty($delRcrd[$i])) {
        /* Comparing the current record number held in
        $delRcrd and first element in $segment. */
            if ($segment[0] == $delRcrd[$i]) {
            /* If a match is found, set the variable
            $writeRcrd to hold 0. */
                $writeRcrd = 0;
            /* Checking if the variable $delEntry is
            empty, to determine the method for storing
            the current category into the variable
            $delEntry. */
                if(empty($delEntry)) {
                    $delEntry = $segment[1]; }
                else {
                    $delEntry = $delEntry.", ".$segment[1];   }
            }
        }
    }
/* Checking the value held by $writeRcrd, to determine
whether the record is retained or not. */
    if ($writeRcrd == 1) {
    /* Ensuring that the last element for $segment ends
    with a new line character. */
        if (ord(substr($segment[2], -1)) == 10) {
        /* Generating the string that will be appended
        into the text file and storing it into the
        variable $append. */
            $append = "$Rcrd:$segment[1]:$segment[2]";
        }
    /* When the last element for $segment does not end
    with a new line character. */
        else {
        /* Concatenating a new line character to the
        string that will be appended into the text file
        and storing it into the variable $append. */
            $append =
                "$Rcrd:$segment[1]:$segment[2]".chr(13).chr(10);
        }
    /* Attempting to append a line into the text file
    and checking if the write operation fails. */
        if(fwrite($fileHandle, $append) === FALSE) {
```

```
                /* Generating and storing an error message,
                which indicates a failure in insert the new
                category. */
                    $Message = '<FONT Color="red">Failure in re-creating the
                                file.</FONT>';      }
            /* Incrementing the entry number and storing it
            into the variable $Rcrd. */
                $Rcrd = $Rcrd + 1;
            }
        }
    }
    /* Closing the file handle which is used to access and
    append the ctgrymstr.txt file. */
        fclose($fileHandle);

    /* Checking if the variable $delEntry is not empty. */
        if(!empty($delEntry)) {
        /* A variable $Message is initialised to hold a message,
        which list the record(s) deleted. */
            $Message = '<BR><FONT Color="RED">* Details for '.$delEntry.' has
                        been deleted.</FONT>';
        }
    }
/* If data cannot be written to the file ctgrymstr.txt or an
error is encountered while creating a file handle for the file.
*/
    else {
    /* Generating and storing an error message that indicates
    failure while accessing the text file. */
        $Message = '<FONT Color="red">Unable to access text file.</FONT>';
    }
}
```

```
/* A variable named $hidMode is
initialised to 'I'. This
variable holds the form status
in terms of I - Insert, U -
Update and D - Delete. */
   $hidMode = 'I';
?>
...
```

Save the changes made to the **ctgry_mstr.php** file and run it within the web browsers. When the page appears, select the checkbox(es) from the tabular layout and click Delete. Refer diagram 13.6.1.

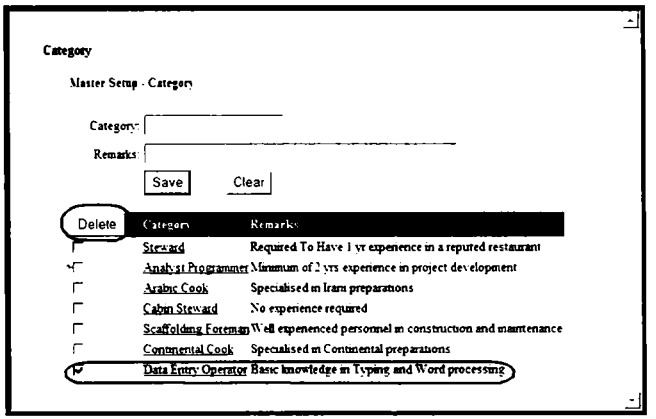


Diagram 13.6.1: Deleting an entry with category information via the ctgry_mstr.php file.

When a call is returned to the **ctgry_mstr.php** file with the list of categories selected for deletion the following processes (strictly bound to the delete mode) occur:

- The **explode()** function is used to split the contents of the variable **$hidDelRcrdLst** and store it into array named **$delRcrd**. The split is on the bases of a comma (,)
- Using the **file()** function, the entire contents of the file **ctgrymstr.txt** is transferred into a variable **$fileContents**
- A file handle is create using the **fopen()** function. The handle is created using the **'w'** switch to allow data to be written to the text file. Using the **'w'** switch it truncates the contents of file opened. This means only those records that are not deleted in this process will be written back to the file
  - A counter variable named **$Rcrd** is initialised with the value **1**. This is required to generate new category identities for the records inserted
  - A loop is created to traverse through each line in **$fileContents** and perform the following:
    - The contents of the current line is transferred into **$line**
    - The **explode()** function is used to split the contents of the current line (Record) and stored it into an array named **$segment**. The splitting is on the bases of a colon (:) and since there are two colons in each line, the array is populated with three elements
    - A check is made to verify whether the first element in **$segment** is empty. If it holds a value:
      - A flag named **$writeRcrd** is initialized with the value **1**. This value will determine whether the current record in the loop will be written back to file
      - A loop is created which traverses through the elements in **$delRcrd** (the array holding the list of entry number(s) marked for deletion) and compares that value with category identities extracted from the file
        - Category identities which match the current element of **$delRcrd** are stored in variable called **$delEntry.** This is done to generate a message indicating the records deleted
        - Category identities which do not match are written back to file along with the new category identities generated (using **$Rcrd).** While writing back these records it is ensured that each and every record is appended with a new line character which will help distinguishing records
        - The value held in **$Rcrd** is incremented by 1
        - The **fwrite()** function is now called to apply the changes in terms of writing data back to the file. If the write operation fails, a message is generated and stored into **$Message** and finally rendered
  - The **fclose()** function is called to release the file handler bound to **ctgrymstr.txt** file

- A message is generated to indicate the categories deleted and stored it into **$Message** which is later on rendered

- If the **ctgrymstr.txt** file is not found, an error message is generated and stored into **$Message** which will be rendered later
- Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:
  - Displaying the value stored in **$Message**
  - Laying out the form for capturing category information
  - Laying out the grid for displaying category information stored in **ctgrymstr.txt** file

The name(s) of category deleted will be listed as shown in diagram 13.6.2.

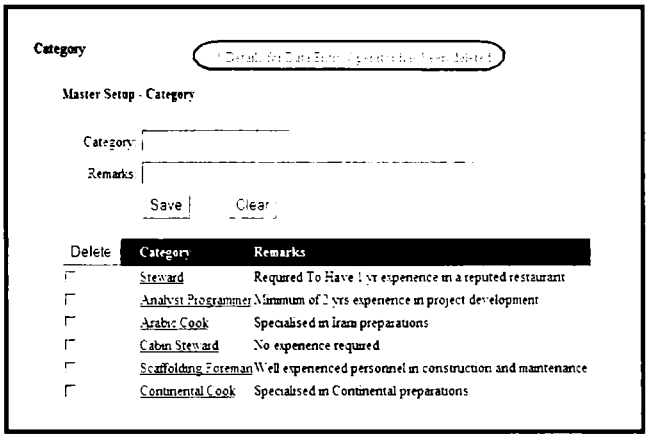


Diagram 13.6.2: Entry deleted by ctgry_mstr.php.

This completes the Category Master module using flat file. The entire code for the **ctgry_mstr.php** appears as listed below:

```
<?
/*
  Date :- 28/05/05
  Author :- Hansel Colaco.
  Filename :- ctgry_mstr.php
  Purpose :- Allows display and insert, updates and delete of
             category master table.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Category Master Form";
/* Storing the name of the file to be read. */
  $filename = "ctgrymstr.txt";
/* A variable holding the message to be displayed on the page. */
  $Message = "";
/* Storing the maximum record number assigned to an entry in the
text file. */
  $maxRcrd = 0;
```

```
/* Initialising a variable named $insert to determine whether to
continue the insert operation or not. If it holds 0, the insert
operation is cancelled. */
  $insert = 1;
/* Initialising a variable named $update to determine whether to
continue the update operation or not. If it holds 0, the update
operation is cancelled. */
  $update = 1;

/* If Category name has been passed through the variable
$txtCatName, (i.e. the $txtCatName is not empty). */
  if (!empty($txtCatName)) {
  /* Using the function file_exists() to check whether PHP is
  able to locate the login.txt file. */
     if (file_exists($filename)) {
     /* If the ctgrymstr.txt file is found. Use the function
     file() to extract the contents of the ctgrymstr.txt file
     into a variable named $fileContents. */
        $fileContents = file($filename);

     /* Using an iteration to extract and transfer each line held
     in $fileContents into the variable $line. */
        foreach ($fileContents as $line_num => $line) {
        /* Using the function explode() to separate the segments
        of the current line and store them as individual elements
        for the array $segment. */
           $segment = explode(":", $line);

        /* Checking if the form is in the insert mode. */
           if ($hidMode == 'I') {
           /* Checking of duplication in category, i.e. category
           name already exists in the text file. */
              if ($segment[1] == $txtCatName) {
              /* Assigning $insert with a 0 (zero), to cancel the
              insert operation. */
                 $insert = 0;
              /* Terminating the iteration used for traversing
              through the each line in the text file. */
                 break;
              }

           /* If a match for the new category name was not found
           in the text file. */
              else {
              /* Checking if the current entry number is greater
              than the value held in $maxRcrd. */
                 if ($segment[0] > $maxRcrd) {
```

```
                /* Storing the highest entry number into the
                variable $maxRcrd. */
                    $maxRcrd = $segment[0];
                }
            }
        }

    /* Checking if the form is in the update mode. */
        if ($hidMode == 'U') {
        /* Checking if modified category name already exists
        as a different entry in the text file. */
            if ($segment[1] == $txtCatName
                and $segment[0] != $hidCatID) {
            /* Assigning $update with a 0 (zero), to cancel the
            update operation. */
                $update = 0;
            /* Terminating the iteration used for traversing
            through the each line in the text file. */
                break;
            }
        }
    }
  }
/* Checking if the form is in the insert mode. */
  if ($hidMode == 'I') {
  /* Checking the value held by $insert. If zero the insert
  operation is cancelled. */
    if ($insert == 0) {
    /* Generating and storing an error message which indicates a
    duplication of category. */
        $Message = '<FONT Color="red">Entry for <B>'.$txtCatName.'</B>
                    already exists. </FONT>';
    }
  /* If the insert operation is permitted to continue. */
    else {
    /* If the ctgrymstr.txt file is editable, a file handle is
    create using the fopen() function. The handle is created
    using the 'a' switch to allow data to be appended to the
    text file. */
        if (is_writable($filename) and $fileHandle = fopen($filename, 'a')) {
        /* Storing the next highest entry number into the
        variable $newRcrd. */
            $newRcrd = $maxRcrd + 1;
        /* Concatenating a new line character to a string that
        will be appended into the text file and storing it into
        the variable $append. */
            $append = "$newRcrd:$txtCatName:$txtCatRmrk".chr(13).chr(10);
```

```
        /* Attempting to append a line into the text file and
        checking if the write operation was successful. */
            if(!fwrite($fileHandle, $append) === FALSE) {
            /* Generating and storing a message that indicates the
            insert operation was successful. */
                $Message = '<FONT Color="blue">Entry for
                            <B>'.$txtCatName.'</B> added
                            successfully.</FONT>';
            }
        /* If the write operation on the ctgrymstr.txt file
        fails. */
            else {
            /* Generating and storing an error message, which
            indicates a failure in insert the new category. */
                $Message = '<FONT Color="red">Failure in adding entry for
                            <B>'.$txtCatName.'</B>.</FONT>';
            }
        /* Closing the file handle which is used to access and
        append the ctgrymstr.txt file. */
            fclose($fileHandle);
        }
    /* If data cannot be written to the file ctgrymstr.txt or an
    error is encountered while creating a file handle for the
    file. */
        else {
        /* Generating and storing an error message that indicates
        failure while accessing the text file. */
            $Message = '<FONT Color="red">Unable to access text file.</FONT>';
        }
    }
}

/* Checking if the form is in the update mode. */
if ($hidMode == 'U') {
/* Checking the value held by $update. If zero the update
operation is cancelled. */
    if ($update == 0) {
    /* Generating and storing an error message which indicates a
    duplication of category. */
        $Message = '<FONT Color="red">The modified category
                    <B>'.$txtCatName.'</B> already exists.</FONT>';
    }
/* If the update operation is permitted to continue. */
    else {
    /* Use the function file() to extract the contents of the
    ctgrymstr.txt file into a variable named $fileContents. */
        $fileContents = file($filename);
```

```
/* If the ctgrymstr.txt file is editable, a file handle is
create using the fopen() function. The handle is created
using the 'w' switch to allow file data to be truncated,
(i.e. the file holds nothing). */
   if (is_writable($filename) and $fileHandle = fopen($filename, 'w')) {
   /* Using an iteration to extract and transfer each line
   held in $fileContents into the variable $line. */
      foreach ($fileContents as $line_num => $line) {
      /* Using the function explode() to separate the
      segments of the current line and store them as
      individual elements for the array $segment. */
         $segment = explode(":", $line);

      /* If the first element in $segment is not empty. */
         if (!empty($segment[0])) {
         /* Checking whether the first element for $segment
         hold a value identical to record number for the
         entry selected for modification. */
            if ($segment[0] == $hidCatID) {
            /* Transferring values from the form fields into
            the second and third segments for the array
            object $segment. */
               $segment[1] = $txtCatName;
               $segment[2] = $txtCatRmrk;
            }

         /* Ensuring that the last element for $segment ends
         with a new line character. */
            if (ord(substr($segment[2], -1)) == 10) {
            /* Generating the string that will be appended
            into the text file and storing it into the
            variable $append. */
               $append = "$segment[0]:$segment[1]:$segment[2]";
            }
         /* When the last element for $segment does not end
         with a new line character. */
            else {
            /* Concatenating a new line character to the
            string that will be appended into the text file
            and storing it into the variable $append. */
               $append =
                        "$segment[0]:$segment[1]:$segment[2]".chr(
                        13).chr(10);  }

         /* Attempting to append a line into the text file
         and checking if the write operation was successful.
         */
            if(!fwrite($fileHandle, $append) === FALSE) {
```

```
                    /* Generating and storing a message that
                    indicates the update operation was successful.
                    */
                        $Message = '<FONT Color="blue">Entry for
                                    <B>'.$txtCatName.'</B> modified
                                    successfully.</FONT>';
                    }
                /* If the write operation on the ctgrymstr.txt file
                fails. */
                    else {
                    /* Generating and storing an error message,
                    which indicates a failure in update the new
                    category. */
                        $Message = '<FONT Color="red">Failure in changing entry
                                    for <B>'.$txtCatName.'</B>.</FONT>';
                    }
                }
            }
        /* Closing the file handle which is used to access and
        append the ctgrymstr.txt file. */
            fclose($fileHandle);
        }
    /* If data cannot be written to the file ctgrymstr.txt or an
    error is encountered while creating a file handle for the
    file. */
        else {
        /* Generating and storing an error message that indicates
        failure while accessing the text file. */
            $Message = '<FONT Color="red">Unable to access text file.</FONT>';
        }
    }
}

/* Checking if the form is in the delete mode. */
if ($hidMode == 'D') {
/* Using the function explode() to separate the entry number(s)
stored in $hidDelRcrdLst and store them as individual elements
of the array $delRcrd. */
    $delRcrd = explode(",", $hidDelRcrdLst);
/* Use the function file() to extract the contents of the
ctgrymstr.txt file into a variable named $fileContents. */
    $fileContents = file($filename);

/* If the ctgrymstr.txt file is editable, a file handle is
create using the fopen() function. The handle is created using
the 'w' switch to allow file data to be truncated, (i.e. the
file holds nothing). */
    if (is_writable($filename) and $fileHandle = fopen($filename, 'w')) {
```

```
/* Initialising the variable $Rcrd to act as a counter for
the entry number. */
   $Rcrd = 1;

/* Using an iteration to extract and transfer each line held
in $fileContents into the variable $line. */
   foreach ($fileContents as $line_num => $line) {
   /* Using the function explode() to separate the segments
   of the current line and store them as individual elements
   for the array $segment. */
      $segment = explode(":", $line);

   /* If the first element in $segment is not empty. */
      if (!empty($segment[0])) {
      /* A variable to act as a flag to determine whether
      the current entry is delete or not. If 1 the record is
      not deleted. */
         $writeRcrd = 1;
      /* An iteration through the element in the array
      holding the list of entry number(s) marked for
      deletion. */
         for($i=0; $i<=count($delRcrd); $i++) {
         /* Checking if the current element in $delRcrd is
         empty. */
            if (!empty($delRcrd[$i])) {
            /* Comparing the current record number held in
            $delRcrd and first element in $segment. */
               if ($segment[0] == $delRcrd[$i]) {
               /* If a match is found, set the variable
               $writeRcrd to hold 0. */
                  $writeRcrd = 0;
               /* Checking if the variable $delEntry is
               empty, to determine the method for storing
               the current category into the variable
               $delEntry. */
                  if(empty($delEntry)) {
                     $delEntry = $segment[1]; }
                  else {
                     $delEntry = $delEntry.", ".$segment[1];   }
               }
            }
         }

      /* Checking the value held by $writeRcrd, to determine
      whether the record is retained or not. */
         if ($writeRcrd == 1) {
```

```
            /* Ensuring that the last element for $segment ends
            with a new line character. */
               if (ord(substr($segment[2], -1)) == 10) {
               /* Generating the string that will be appended
               into the text file and storing it into the
               variable $append. */
                  $append = "$Rcrd:$segment[1]:$segment[2]";
               }
            /* When the last element for $segment does not end
            with a new line character. */
               else {
               /* Concatenating a new line character to the
               string that will be appended into the text file
               and storing it into the variable $append. */
                  $append = "$Rcrd:$segment[1]:$segment[2]"
                            .chr(13).chr(10);
               }
            /* Attempting to append a line into the text file
            and checking if the write operation fails. */
               if(fwrite($fileHandle, $append) === FALSE) {
               /* Generating and storing an error message,
               which indicates a failure in insert the new
               category. */
                  $Message = '<FONT Color="red">Failure in re-creating the
                             file.</FONT>';
               }
            /* Incrementing the entry number and storing it
            into the variable $Rcrd. */
               $Rcrd = $Rcrd + 1;
            }
         }
      }
   /* Closing the file handle which is used to access and
   append the ctgrymstr.txt file. */
      fclose($fileHandle);
   /* Checking if the variable $delEntry is not empty. */
      if(!empty($delEntry)) {
      /* A variable $Message is initialised to hold a message,
      which list the record(s) deleted. */
         $Message = '<BR><FONT Color="RED">* Details for '.$delEntry.' has
                    been deleted.</FONT>';
      }
   }
/* If data cannot be written to the file ctgrymstr.txt or an
error is encountered while creating a file handle for the file.
*/
   else {
```

```
    /* Generating and storing an error message that indicates
    failure while accessing the text file. */
       $Message = '<FONT Color="red">Unable to access text file.</FONT>';
    }
  }

/* A variable named $hidMode is initialised to 'I'. This variable
hold the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
?>

<HTML>
<HEAD>
  <TITLE><?=$title;?></TITLE>
  <META Content="<?=$author;?>" Name="Author">
  <META Content="" Name="Keywords">
  <META Content="" Name="Description">
  <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- JavaScript code-spec to trim a string and change string
case. -->
  <SCRIPT Language="JavaScript" Src="mstrscript.js"
          Type="text/javascript"></SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
  <BR><BR>
<!-- JavaScript code-spec unique to the category master form. -->
  <SCRIPT Language="JavaScript">
  /* The chkBlanks() function verifies that the form field of
  category name is not left empty. If the field is empty or
  contains blank spaces, a message indicates to enter a valid
  value. If the field is not empty, the data is submitted. */
     function chkBlanks() {
     /* A variable holding a reference to the form object
     initialised in this page. */
        frm = document.frmMstrCat;
     /* Extracting the contents of the category name field into a
     variable named str. */
        var str = frm.txtCatName.value;

     /* If the category name field is empty or contains blank
     spaces. */
        if(str.trim() == "") {
        /* Displaying a message indicates to enter a valid value.
        */
           alert('Please enter category name.');
        /* Placing the form cursor on the category name field. */
           frm.txtCatName.focus();
```

```
        /* Preventing submission of data held in form fields. */
            return false;
        }
    /* If the category name field holds a value. */
        else {
        /* Calling the function vldtFrmFlds() to validate data
        captured by the form. If the function returns TRUE. */
            if (vldtFrmFlds() == true) {
            /* Allowing the data in the form field to be
            submitted. */
                return true; }
        /* If the function vldtFrmFlds() returns FALSE during
        validation of data captured by the form. */
            else {
            /* Preventing the data in the form field from being
            submitted. */
                return false; }
        }
    }

/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
    function vldtFrmFlds() {
    /* A variable holding a reference to the form object
    initialised in this page. */
        frm = document.frmMstrCat;
    /* If the form is in the update mode. */
        if (frm.hidMode.value == 'U') {
        /* Initialising a variable named mNoChng to hold TRUE.
        This value of this variable determines whether data in
        the form field has been modified or not. */
            var mNoChng = true;
        /* Checking the form objects individually for modified
        information. If any changes have been made, the value of
        the variable named mNoChng is set to hold FALSE. */
            if (frm.hidCatName.value != frm.txtCatName.value) {
                mNoChng = false; }
            if (frm.hidCatRmrk.value != frm.txtCatRmrk.value) {
                mNoChng = false; }
        /* If data in the form fields have remained unchanged. */
            if (mNoChng) {
                alert('Update failed.\n No changes have been made during
                    modification.');
            /* Calling the JavaScript function to reset values in
            the form fields. */
                setNewMode();
```

```
            /* Placing the form cursor on the category name field.
            */
               frm.txtCatName.focus();
            /* Preventing the data in the form field from being
            submitted. */
               return false;
            }

         /* If data in the form fields have been changed. */
            else {
            /* Allowing the data in the form field to be
            submitted. */
               return true;  }
         }

      /* If the form is not in the update mode. */
         else {
      /* Allowing the data in the form field to be submitted. */
         return true;  }
      }

   /* The function setNewMode() is called to prepare the form to
   except a new set for data. This function is called directly
   when the Clear button is clicked. */
      function setNewMode() {
      /* A variable holding a reference to the form object
      initialised in this page. */
         frm = document.frmMstrCat;
      /* Clearing form fields. */
         frm.txtCatName.value = "";
         frm.txtCatRmrk.value = "";
      /* Clearing hidden variables used for comparison during in
      the update operation. */
         frm.hidCatID.value = "";
         frm.hidCatName.value = "";
         frm.hidCatRmrk.value = "";
      /* Setting the form mode to insert. */
         frm.hidMode.value = "I";

      /* If the form contains a Delete button. */
         if (frm.cmdDelete) {
         /* Enabling the Delete button. */
            frm.cmdDelete.disabled = false;      }
      }
```

```
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
   function setEditMode(id, name, rmrk) {
   /* A variable holding a reference to the form object
   initialised in this page. */
      frm = document.frmMstrCat;
   /* If the form is not in the update mode. */
      if(frm.hidMode.value != 'U') {
      /* Assigning values passed as parameters to hidden
      variables. These are used for comparison during in the
      update operation. */
         frm.hidCatID.value = id;
         frm.hidCatName.value = name;
         frm.hidCatRmrk.value = rmrk;
      /* Populating the form fields. */
         frm.txtCatName.value = name;
         frm.txtCatRmrk.value = rmrk;
      /* Setting the form mode to update. */
         frm.hidMode.value = 'U';
      /* Disabling the Delete button. */
         frm.cmdDelete.disabled = true;
      }
   /* If the form is in the update mode. */
      else {
         alert('Update in progress.\n An entry has already been selected for
               modification.'); }
   }

/* The function setDelMode() is called to prepare the form to
remove data from the tabular layout. This function is called
directly when the Delete button is clicked. */
   function setDelMode() {
   /* Calling the JavaScript function to reset values in the
   form fields. */
      setNewMode();
   /* Setting the form mode for deleting record(s). */
      document.frmMstrCat.hidMode.value = 'D';
   /* Calling the JavaScript function to populate the variable
   holding a list of deleted records. The function
   formDeleteValues() has been defined in the mstrscript.js
   file. */
      formDeleteValues('hidDelRcrdLst', 'frmMstrCat');
   }
```

```
  </SCRIPT>
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0"
         CellSpacing="0" Name="TlbOuter" Width="90%"><TR>
      <TD Align="center" Border="1" BgColor="#C0C0C0"
         Width="10%">Category</TD>
      <TD Align="center" BgColor="#FFFFFF" ColSpan="9"
         Width="90%"><?=$Message;?></TD>
  </TR>
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="ctgry_mstr.php" Method="post" Name="frmMstrCat"
         onSubmit="return chkBlanks();">
  <!-- Declaring a hidden form field used to identify the current
  (i.e. category master) page. -->
      <INPUT Name="hidPage" Type="hidden" Value="ctrgy_mstr">
  <!-- Declaring hidden form fields required for data validation.
  -->
      <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
      <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
      <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
  <!-- Declaring a hidden form field used for determining the
  form mode. It will holds 'I' for Insert, 'U' for Update and 'D'
  for Delete. It will hold 'I' when the page is rendered for the
  1st time. -->
      <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
  <!-- Declaring a hidden form field used for identifying records
  which have been selected for deletion. -->
      <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
  <TR Height="300" VAlign="top">
      <TD Align="center" Border="1" ColSpan="10"><BR>
      <!-- Form Table Code Begins. -->
          <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
                 Name="TlbInner" Width="90%"><TR>
              <TD Align="left" ColSpan="2"><FONT Size="2"
                 Color="#0000CC"><B>Master Setup -
                 Category</B></FONT></TD>
          </TR><TR><TD Align="center" ColSpan="2"> </TD>
          </TR><TR>
              <TD Align="right" Width="15%">Category:</TD>
              <TD Align="left"><INPUT maxLength="35" Name="txtCatName"
                 onBlur="chngCase('txtCatName', 'frmMstrCat');"
                 Type="text"></TD>
          </TR><TR>
              <TD Align="right">Remarks:</TD>
```

```
            <TD Align="left"><INPUT maxLength="255" Name="txtCatRmrk"
                onBlur="chngCase('txtCatRmrk', 'frmMstrCat');" Type="text"
                Size="50"></TD>
        </TR><TR>
            <TD> </TD>
            <TD Align="left">
                <INPUT Name="cmdSubmit" Type="submit" Value="Save">
                <IMG Height="1" Src="../Images/pixel.gif" Width="30">
                <INPUT Name="cmdReset" onClick="setNewMode();"
                      Type="button" Value="Clear">
            </TD>
        </TR></TABLE><BR>
    <!-- Form Table Code End. -->
<?
/* Using the function file_exists() to check whether PHP is able
to locate the login.txt file. */
  if (file_exists($filename)) {
  /* If the ctgrymstr.txt file is found. */
?>
    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
              Width="90%">
        <TR BgColor="#400040">
            <TD Width="15%"><INPUT Name="cmdDelete"
                 onClick="setDelMode();" Type="button"
                 Value="Delete"></TD>
            <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
        </TR>
<?
  /* Use the function file() to extract the contents of the
  ctgrymstr.txt file into a variable named $fileContents. */
     $fileContents = file($filename);
  /* Using an iteration to extract and transfer each line held in
  $fileContents into the variable $line. */
     foreach ($fileContents as $line_num => $line) {
     /* Using the function explode() to separate the segments of
     the current line and store them as individual elements for
     the array $segment. */
        $segment = explode(":", $line);
     /* If the second element in $segment is not empty. */
        if (!empty($segment[1])) {
?>
        <TR BgColor="#E4E4E4">
            <TD><INPUT Name="chk<?=($segment[0]);?>" Type="checkbox"
                 Value="<?=($segment[0]);?>"></TD>
```

```
            <TD><A HRef="JavaScript:setEditMode('<?=($segment[0]);?>',
                '<?=($segment[1]);?>', '<?=($segment[2]);?>')">
                <?echo($segment[1]);?></A></TD>
            <TD><?=($segment[2]);?></TD>
        </TR>
<?
        }
    }
?>
        </TABLE><BR>
<?  }
/* If PHP cannot locate the login.txt file. */
  else {
?>
  <FONT Color="red"><B>Unable to open file!</B></FONT>
<?
  /* Terminating the any further processes on the page. */
    exit();
  }
?>
    </TD>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends. -->
</BODY>
</HTML>
```

As seen in the above code, the JavaScript functions for the HTML page rendered by **ctgry_mstr.php** is stored in the **mstrscript.js** file. The file will be downloaded by the client's web browser and will contain the following:

```
/*
  Date :- 13/05/05
  Author :- Hansel Colaco
  Filename :- mstrscript.js
  Purpose :- Defines Generic JavaScript functions for trimming
             strings and changing string case.
*/
/* Generating the function used for trimming string values. */
  function strtrim() {
  /* Returns the string passed as a parameter after removing
  blank spaces, if any, at both the beginning and the end of the
  parameter. */
    return this.replace(/^\s+/,'').replace(/\s+$/,'');
  }
/* Generating a global alias name for the function used for
trimming string values. */
  String.prototype.trim = strtrim;
```

```
/* The chngCase() function changes the first alphabet to an upper
case character for the value held in the field passed as the
first parameter. The second parameter is a reference to the form
holding the field of the first parameter. This function is called
when a form cursor moves away from a field. */
  function chngCase(pFld, pFrm) {
  /* Using the eval() function to extract the value held in the
  field passed as a parameter. */
     var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
  /* Removing extra spaces for the value extracted above and
  restoring into the same variable. */
     mFldVal = mFldVal.trim();
  /* If the variable containing the extracted value is not empty.
  */
     if (mFldVal != "") {
     /* Changing the string Proper case. */
        mFldVal = mFldVal.charAt(0).toUpperCase() + mFldVal.substr(1,
                  mFldVal.length);   }
  /* Using the eval() function to display the new value in the
  field passed as a parameter. */
     eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
  }

/* The formDeleteValues() function generates a string of comma
separated identities of the records selected for deletion. If no
records are selected, a message indicates the same. The
parameters for this function are references to a hidden variable
(used for storing the generated string) and the form
respectively. It is called when the DELETE button is clicked. */
  function formDeleteValues(pHid, pFrm) {
  /* Initialising variables for later use. */
     var selValues = "";
     var firstSelBox = "";
  /* Using the eval() function to create a variable to hold a
  reference to the form object. */
     eval("frm = document." + pFrm);
  /* Iterating through every object on the form. */
     for (i=0; i<frm.elements.length; i++) {
     /* If the current object is a Checkbox. */
        if (frm.elements[i].type == "checkbox") {
        /* If the variable firstSelBox is empty. */
           if (firstSelBox == "") {
           /* Assigning the element number, of the current form
           object, to the variable firstSelBox. */
              firstSelBox = i; }
           /* If the current Checkbox has been selected, (i.e.
           checked). */
           if (frm.elements[i].checked == true) {
```

```
            /* Assigning the element number, of the current form
            object, to the variable selValues. */
               selValues = selValues + frm.elements[i].value + ",";      }
         }
      }
   /* If the variable selValues does not holds a value. */
      if (selValues.length < 1) {
      /* Displaying a message indicating to select a checkbox. */
         alert('Please choose records you wish to delete.');
      /* Using the eval() function to place the form cursor on the
      first checkbox. */
         eval("document." + pFrm + ".elements[" + firstSelBox + "].focus();");
      }
   /* If the variable selValues holds a value. */
      else {
      /* Removing the last character (i.e. a comma) for the value
      held in the variable selValues. */
         selValues = selValues.substring(0, selValues.length-1);
      /* Using eval() function to assign a value to the form's
      hidden variable. */
         eval("document." + pFrm + "." + pHid + ".value = '" + selValues + "';");
         frm.submit();
      }
   }
```

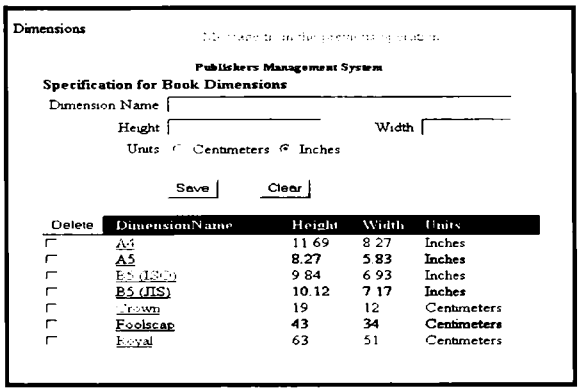


Diagram 13.7: Output for DmsnMstr.php.

Hands On Exercises

1. Based on the Hands on exercise developed in the previous Chapter 12, modify the form to capture information for Book Dimensions. After modifications, the PHP program should be able to store captured data into a flat file named **dmsnmstr.txt**. This file is available at the same location as that of the PHP program executed.

- The contents held in the flat file is displayed in a tabular layout, immediately after the data entry form. Refer diagram 13.7
- Besides storing new entries, the program should allow modification of previously stored data
- Similarly, the PHP program should allow (single / multiple) deletion of entries held in the flat file
- The page (containing the data entry form and the tabular layout) generated, will display a message indicating the previous operation performed by the PHP program

SECTION III: WORKING WITH PHP

Integration Between PHP And Oracle

Programmer's pickup a Web server side scripting language like PHP largely to be able to interact with databases. PHP and Oracle is really the best combination for robust data-driven Web sites. PHP understands and supports ANSI SQL, it also compiles on a number of platforms, has multithreading abilities on Unix servers, all of which make for great performance. On Windows NT/XP, Oracle can be run as a service and as a normal process in Windows 95/98 machines.

Till fairly recently MySQL and Postgres, was the only real option for database software run on Linux. Today however, there are a number of commercial database products officially supported on Linux, including the industry heavyweights **Oracle** and **DB2**. These products, combined with Apache Web server and PHP, make Linux a very attractive platform for developing and running Web-enabled database applications of all sizes.

The PHP-Oracle combination is truly cross-platform and free. This means an application can be developed on Windows and then copied to a Unix platform where it will run cheerfully. PHP can be run as an external CGI process, a standalone script interpreter or an embedded Apache module, real flexibility.

The ability to efficiently store and retrieve large amounts of information on demand has contributed enormously to the success of the Internet. This is almost always implemented through the use of a database. Sites such as Yahoo, IndiaTimes and Rediff depend heavily on the reliability of their databases for storing and retrieving enormous amounts of information accurately.

A database is very useful when used with a website. There are a huge variety of things that can be done with this combination, from displaying simple lists to running a complete website from a database.

When a database is properly implemented, it can be used for storing, retrieving and manipulating data. Adding site search and information sorting features really simplify when a database is used. Control over viewing permissions becomes a non-issue because of the access control features in many database systems. Data replication and backup are also simple.

Connecting To AN Oracle Database

Before any interactive HTML form based work can be done, PHP must be able to connect to an Oracle database. If PHP is not connected to the Oracle database, then all commands forwarded to the database via PHP, will fail.

REMINDER

A good practice for using databases is to specify the username, password and connection string name right at the top of the code spec so that if a change is required at a later date, only one line which is at the top of the code spec needs changing.
For example:

```
$username="username";
$password="password";
$connstring="connection_string";
```

HINT

Keeping the database password in a PHP file is not a security risk at all. The Web server processes PHP code spec before anything is sent to a Browser, hence it is impossible for a user to see the PHP script's source.

Getting Started

There are several ways for PHP to talk to an Oracle database, including using ODBC libraries, as well as two types of Oracle library support built into PHP. This chapter concentrates on using the Oracle native libraries specifically the Oracle 8 function calls. The older Oracle libraries still exist in PHP, but are largely deprecated by the OCI8 functions.

Here is the general order of events that take place during the Oracle server communications process:

1. Establish a connection with the Oracle server, by specifying the login information. If the connection attempt fails, display an appropriate message and exit the process
2. Perform necessary queries on the selected database(s)
3. Once querying is complete, close the database server connection

Connecting To the Database

There are several basic steps to running any Oracle SQL command under PHP, which include connecting to the database, parsing the SQL command, executing the command and returning any results. Connecting to the database is accomplished using the **ocilogon** command. The command takes three arguments: database user name, password and the name of the database (Usually specified in form of a connection string).

Issue the following commands to the PHP interpreter to create a connection with the Oracle database:

```
/* Variables used to store Database access information. */
  $dbuser = "dba_pm";
  $dbpass = "sct2306";
  $connStr = "OraLin";  (Only if the Database is on some other
                        machine and communication is via a client)
```

If Oracle Database is on same machine:

```
  $rcq = ocilogon($dbuser, $dbpass);
```

If Oracle Database is on some other machine and is communicated with via a client:

```
  $rcq = ocilogon($dbuser, $dbpass, $connStr);
```

The above lines tell PHP to connect to the Oracle database engine running in memory (RAM) with a specific username and password, which have been passed as parameters.

The function **ocilogon()** accepts the oracle login information and creates the actual connection to the database. This connection can be made available whenever required by referring to **$rcq**, which is the connection object created and returned by the **ocilogon()** function. The database name parameter is optional and if it is not specified, the value of **$ORACLE_SID** (registered in the environment earlier) is assumed.

Executing Commands

Once connected to a database, the next thing is to run some SQL commands, which is accomplished with a combination of functions namely **OCIParse** and **OCIExecute** to generate and return the results in a usable form.

Syntax:

```
  int OCIParse(int <Connection>, string <Query>)
```

OCIParse takes the database connection generated earlier and an SQL string as arguments and returns the statement ID of the parsed statement.

Syntax:

```
  int OCIExecute(int <StatementID>, int <Mode>)
```

OCIExecute takes the results of **OCIParse** and an optional mode, which tells the database whether to commit the results of the command (The default being **COMMIT**) and returns **TRUE** on success, **FALSE** on failure.

Other than the above functions, there are two more functions used often.

Syntax:

```
int OCIFetchStatement(int <StatementID>, array <Variable>)
```

OCIFetchStatement takes the statement ID of an executed statement and an array to populate with the results of the query and returns the number of rows in the result set.

Syntax:

```
int OCIRowCount(int <StatementID>)
```

OCIRowCount takes a statement ID and returns the number of rows altered by an update.

Inserting Data

Add a block of information to the database table **category:** (Assumed that the **category** table already exists. Refer **Chapter 06: Basic Interaction With SQL**)

```
/* Building and executing a SQL query to INSERT the new category
into the category table. */
$query = "INSERT INTO category (category_ID, category_name, category_remarks)
        VALUES(1, '".$txtCatName."', '".$txtCatRmrk."')";
/* Parsing the SQL Query. */
  $parsed = ociparse($rcq, $query);
/* Executing the SQL Query. */
  ociexecute($parsed);
/* Generating and storing a message that indicates the insert
operation was successful. */
  $Message = '<FONT Color="blue">Entry for <B>'.$txtCatName.'</B> added
  successfully.</FONT>';
  echo $Message;
```

The variable **$query** is used because the ANSI SQL's **INSERT INTO** statement is being assigned to this variable.

The **INSERT INTO** statement tells the PHP interpreter to append into the **category** table (found within the database) with the values enclosed within parenthesis, **i.e.** the parenthesis contain all the information to be added to the table **category**.

Example: (ctgryInsert.php)

```
<?php
/* Variables used to store Database access information. */
  $dbuser = "dba_pm";
  $dbpass = "sct2306";
```

```
/* Use only if the Database is on some other machine and
communication is via a client. OraLin will be replaced with the
Service Name created using Net Configuration Assistant. */
  $connStr = "OraLin";

/* Connect as, if Oracle Database on same machine */
  $rcq = ocilogon($dbuser, $dbpass);

/* Building and executing a SQL query to INSERT the new category
into the category table. */
  $query = "INSERT INTO category (category_ID, category_name,
            category_remarks) VALUES(6, 'Analyst Designer', 'With 2 Years
            Experience')";

/* Parsing the SQL Query. */
  $parsed = ociparse($rcq, $query);
/* Executing the SQL Query. */
  ociexecute($parsed);

/* Generating and storing a message that indicates the insert
operation was successful. */
  $Message = 'Entry added successfully.';
/* Displaying the message. */
  echo "$Message\n";
?>
```

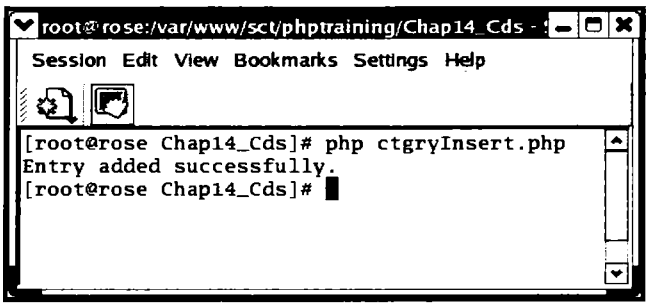


Diagram 14.1: Output for ctgryInsert.php.

Output: (Refer diagram 14.1)

HTML Input

Inputting data using HTML pages is almost identical to inserting it using a PHP script. The benefit, though, is that the script need not be changed for each block of data that has to be placed into the Category table. Users can also be allowed to input their own data using this technique.

The following code spec (saved as **demo.html**) will display an HTML page with textboxes, via which appropriate details that can be entered into the table **category:**

```
<HTML><BODY>
  <FORM Action="insert.php" Method="post">
      Category ID: <INPUT Type="text" Name="txtCategory_ID"><BR>
      Category Name: <INPUT Type="text" Name="txtCategory_Name"><BR>
      Category Remarks: <INPUT Type="text" Name="txtCategory_Remarks"><BR>
      <INPUT Type="Submit" Value="Submit">
  </FORM>
</BODY></HTML>
```

Edit the PHP script written earlier, i.e. the **ctgryInsert.php** file. Replace the code block with hard coded information, with the new one where data is input into the database table, using variables:

```
<?php
/* Variables used to store Database access information. */
  $dbuser = "dba_pm";
  $dbpass = "sct2306";
/* Use only if the Database is on some other machine and communication
is via a client. OraLin will be replaced with the Service Name created
using Net Configuration Assistant. */
  $connStr = "OraLin";

/* Connect as, if Oracle Database on same machine */
  $rcq = ocilogon($dbuser, $dbpass);

/* Building and executing a SQL query to INSERT the new category into
the category table. */
  $query = "INSERT INTO category (category_ID, category_name,
            category_remarks) VALUES(".$_POST['txtCategory_ID'].",
            '".$_POST['txtCategory_Name']."',
            '".$_POST['txtCategory_Remarks']."')";
/* Parsing the SQL Query. */
  $parsed = ociparse($rcq, $query);
/* Executing the SQL Query. */
  ociexecute($parsed);

/* Generating and storing a message that indicates the insert operation
was successful. */
  $Message = '<FONT Color="blue">Entry for
              <B>'.$_POST['txtCategory_Name'].'</B> added
              successfully.</FONT>';
/* Displaying the message. */
  echo $Message;
?>
```

This script should then be saved as **insert.php** so that an HTML form can invoke it on demand.

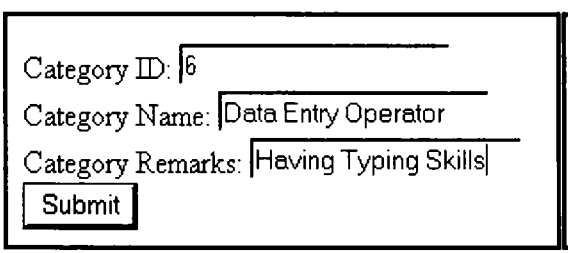


Diagram 14.2.1: Entering values into the page generated by demo.html.

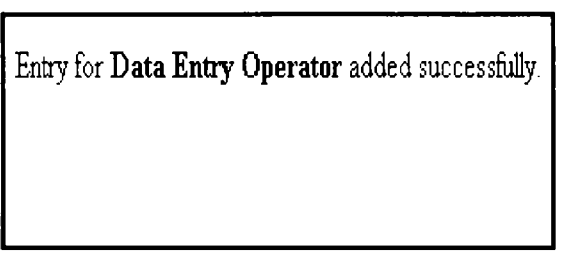


Diagram 14.2.2: Message displayed on the page generated by insert.php.

It works because, instead of the data being hard coded, it is being entered via the HTML form and stored in variables, which are then passed to the PHP interpreter for further processing.

Extracting Data

Now that there is at least one record, (if not many more), in the database table Category this data needs to be extracted using PHP.

The first command will be a query as shown below:

```
SELECT * FROM category;
```

This is a basic command, which tells the PHP script to select all the records from the Category table.

```
$query="SELECT * FROM category";
$parsed = ociparse($rcq, $query);
ociexecute($parsed);
```

Setting Up The Loop

Set up a loop to take each row of the **$parsed** array and print out the data held there to the VDU.

```
<?
  do {
      echo($line['CATEGORY_ID'] . " || ". $line['CATEGORY_NAME'] . " || " .
          $line['CATEGORY_REMARKS'] . "<BR>");
  } while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM))
?>
```

This is a basic PHP loop, which executes the code for the number of records retrieved.

Combining The Script

A full PHP script (saved as **select.php**), which outputs data:

```
<?php
/* Variables used to store Database access information. */
  $dbuser = "dba_pm";
  $dbpass = "sct2306";
/* Use only if the Database is on some other machine and
communication is via a client. OraLin will be replaced with the
Service Name created using Net Configuration Assistant. */
  $connStr = "OraLin";
```

```
/* Connect as, if Oracle Database on same machine */
  $rcq = ociologon($dbuser, $dbpass);
// Declare query
  $query="SELECT * FROM Category";
  $parsed = ociparse($rcq, $query);
  ociexecute($parsed);

// Generating Output
  echo "<B><CENTER>Database Output</CENTER></B><BR><BR>";
// Setting up a loop
  while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) {
      echo($line['CATEGORY_ID'] . " || ". $line['CATEGORY_NAME'] . " || " .
          $line['CATEGORY_REMARKS'] . "<BR>");
  }
?>
```

In this script the data is not formatted when it is rendered. Refer diagram 14.3

Database Output

1 || Continental Cook || Specilised in continental preparations
2 || Arabic Cook || Specilised in Irani preparations
3 || Analyst Programmer || Specilised in Irani preparation
4 || Scaffolding Foreman || Well experienced personnel in construction and development
5 || Cabin Steward ||
6 || Analyst Designer || With 2 Years Experience
7 || Data Entry Operator || Having Typing Skills

Diagram 14.3: Output for select.php.

Designing A Login Module With Database Support

The functionality of the Login module built in the earlier chapter, (Chapter 13: **Filehandling In PHP** section on *Designing A Login Module With Flat File Support*) is a full-fledged login authentication system for visitors to a Website. The module is capable of comparing a login name and password, submitted by a visitor with a valid set of login information stored in a flat file on the Web server. Only when a match for both the login name and its corresponding password is found, a visitor is allowed access to other modules in the Website.

Although better than using hard coded values in the script itself, the technique of using flat files has security drawbacks. Contents of a normal file are easily accessible via an ASCII editor. Even though a file can be locally protected by a user login system, (as found in UNIX, Linux, Windows Servers and so on) yet the risk of external access to confidential file data is very high.

The introduction of databases reduced security risk considerably. Higher end RDBMS like Oracle, MS SQL, MySQL and so on provide an additional layer of security to confidential data. The example, which follows is a Login module, which refers to data stored in table controlled by the Oracle database server.

Since Oracle is the database in use, access to a table is permitted only after logging in to Oracle. A user of the Oracle database system is give part or whole access to resources under Oracle depending on the login used to access the system. Every table under Oracle is bound to a Tablespace and a user login. Hence, every user login is in turn automatically bound to a Tablespace.

Database Structure For The Login Module

Since the example to follow demonstrates the technique of accessing data stored in Oracle table. It is necessary to create the underlying tablespace, users bound to the tablespace and data tables within the tablespace.

REMINDER

All objects and resources under Oracle are created, accessed and manipulated via SQL statements. These SQL statements get processed and executed by the Oracle Database Engine, which listens to commands passed by other tools. SQL *PLUS is the simplest interactive SQL tool used for passing SQL statements to the Oracle Database Engine.

The first step is to create a Tablespace. The following SQL statement, when passed via SQL *PLUS, will create a Tablespace named **PM_SYS:**

```
CREATE TABLESPACE PM_SYS
  DATAFILE 'PM_SYS.dat' SIZE 50M
  DEFAULT STORAGE(INITIAL 10K Next 50K MINEXTENTS 1 MAXEXTENTS 999
                  PCTINCREASE 10)
  ONLINE;
```

The second step is to create a user login. The following SQL statement will create a user login under Oracle named **DBA_PM**, (The user is bound to the **PM_SYS** tablespace)**:**

```
CREATE USER "DBA_PM"
  PROFILE "DEFAULT"
  IDENTIFIED BY "sct2306"
  DEFAULT TABLESPACE "PM_SYS"
  TEMPORARY TABLESPACE "TEMP"
  ACCOUNT UNLOCK;
```

The creation of the user login is followed by setting permissions. The following SQL statement converts the user **DBA_PM** into an Oracle superuser or Database Administrator**:**

```
GRANT "DBA" TO "DBA_PM" WITH ADMIN OPTION;
```

The final step is to create a table which will store valid login information used/referred by the website.

The structure of the Oracle table will be as follows:

Table Definition:

Table Name : **USERS**
Primary Key : USER_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
USER_ID	VARCHAR2	7	NOT NULL	
USER_USERNAME	VARCHAR2	15	NOT NULL	
USER_PASSWORD	VARCHAR2	20	NOT NULL	
USER_TYPE	VARCHAR2	5	NOT NULL	admn / cand / clnt
FOR_PASS_QUESTION	VARCHAR2	100		NULL
FOR_PASS_ANSWER	VARCHAR2	50		NULL

Table Description:

Column Name	Description
USER_ID	Auto Generated unique user number beginning with **US**, followed by 5 digits
USER_USERNAME	Username for the user to login
USER_PASSWORD	Password for the user
USER_TYPE	User types: candidate (cand), client (clnt), administrator (admn)
FOR_PASS_QUESTION	The question for Forgot Password Option
FOR_PASS_ANSWER	The answer for forgot password option

Explanation:
The USERS table stores authentication information of system users (clients and candidates).

The SQL statement used to create the table with the above description is as follows:

```
CREATE TABLE USERS (
   USER_ID VarChar2(7) NOT NULL,
   USER_USERNAME VarChar2(15) NOT NULL,
   USER_PASSWORD VarChar2(20) NOT NULL,
   USER_TYPE VarChar2(5) CHECK (USER_TYPE IN('cand', 'clnt', 'admn')),
   FOR_PASS_QUESTION VarChar2(100) DEFAULT NULL,
   FOR_PASS_ANSWER VarChar2(50) DEFAULT NULL,
   PRIMARY KEY (USER_ID)
);
```

The creation of the table is followed by populating it with sample data. The following SQL statements will insert four valid login for the website:

```
INSERT INTO USERS VALUES ('US00000', 'admin', 'sct2306', 'admn',
   'admin', 'password');
INSERT INTO USERS VALUES ('US00001', 'sharanam', 'sct2306', 'cand',
   'Whats up', 'nothing');
```

```
INSERT INTO USERS VALUES ('US00002', 'hansel', 'sct2306', 'cand',
  'Whats my moms name?', 'mom');
INSERT INTO USERS VALUES ('US00003', 'ivan', 'sct2306', 'cand',
  'what is your wife name?', 'cynthia');
```

Converting The Login Module To Support A Database Management System

Having created the tables for the Login System, changes in PHP code can commence. The example used in **Chapter 13: Filehandling In PHP**, is used as a base, hence do the following:

- Create a directory named **Chap14_Cds** under **sct.phptraining.com** domain
- Create a copy of the files **login.html**, **loginscript.js** and **logincheck.php** into the **Chap14_Cds** directory
- Since the **logincheck.php** file performs the authentication of the visitor's login information. Open the file in an ASCII editor and change its contents as show below:

```
<?
/*
  Date :- 29/05/05
  Author :- Hansel Colaco.
  Filename :- logincheck.php
  Purpose :- Reads database to authenticate the login information
  provided by the visitor.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco';

/* Storing the system date in the format 1st Jan, 2000. */
  $sysdt = date("jS M, Y");
/* Storing the system time in the format 00:00. */
  $systm = date("H:i");

/* Variables used to store Database access information. */
  $dbuser = "dba_pm";
  $dbpass = "sct2306";

/* Checking if the variable either username or password is empty.
This means that the visitor has managed to submit form
information with empty fields. */
  if(empty($txtUserName) || empty($txtPassword)) {
      $Message = '<FONT Color="red"><B>Login Information Not
                  Submitted!</B></FONT>';
  }
```

```
else {
/* The function ocilogon() accepts the oracle login information
to create the actual connection to the database. This
connection made available whenever required by referencing
$rcq, which the connection object created by the ocilogon()
function. */
   $rcq = ocilogon($dbuser, $dbpass);
/* Checking whether the creation of the connection object
failed or not. If failed, an error message is generated. */
   if(!$rcq){
      $Message = '<FONT Color="red"><B>Database connection error!</B>
                  <BR>Please contact the Oracle Database
                  Administrator.</FONT><BR>';
   }
/* If the creation of the connection object succeeded. */
   else {
   /* Building a SQL query to retrieve values held in USER_ID
   and USER_TYPE column from the USERS table. The data
   retrieved is filtered by finding the record which satisfies
   the following conditions completely: a) the USER_USERNAME
   column matches the value captured by the User Name field;
   and b) the USER_PASSWORD column matches the value captured
   by the password field. */
      $query = "SELECT user_ID, user_type FROM users WHERE
               user_username='$txtUserName' AND
               user_password='$txtPassword'";
   /* Parsing the SQL query. */
      $parsed = ociparse($rcq, $query);
   /* Executing the SQL query. */
      ociexecute($parsed);
   /* Checking if any record was retrieved by the above SQL
   query. If a valid user is found, a message is displayed,
   which indicates a Valid login attempt. */
      if(ocifetchinto($parsed, $rslt_arr, OCI_NUM)) {
         $Message = '<FONT Color="red"><B>Welcomes To The
                     Website!</B><BR>A Valid Login information has been
                     submitted.</FONT><BR>';
      }
   /* If a valid user is not found, an error message indicating
   the Incorrect login attempt is displayed with a link to the
   login page. */
      else {
         $Message = '<FONT Color="red"><B>Incorrect
                     Login!</B><BR>Please <A
                     HRef="/Chap14_Cds/login.html">re-log</A> with
                     valid information.</FONT><BR>';       }
   }
}
?>
```

```
<!-- Generating a page to read contents of a database using a
PHP's Oracle function. -->
<HTML><HEAD>
  <TITLE>Login Authentication Through A Database Table</TITLE>
</HEAD><BODY><?=$Message;?><BR><HR>
<!-- Presenting the login inform in a tabular layout. -->
  <H4>Details of the login attempt are:</H4>
  <TABLE Border="1" CellSpacing="1" Width="200"><TR>
      <TD Align="right" Width="75">Username :</TD>
      <TD Align="left" Width="100"><?="$txtUserName";?></TD>
  </TR><TR>
      <TD Align="right" Width="75">Password :</TD>
      <TD Align="left" Width="100"><?="$txtPassword";?></TD>
  </TR><TR>
      <TD Align="right" Width="75">Server Date :</TD>
      <TD Align="left" Width="100"><?="$sysdt";?></TD>
  </TR><TR>
      <TD Align="right" Width="75">Server Time :</TD>
      <TD Align="left" Width="100"><?="$systm";?></TD>
  </TR></TABLE><HR>
</BODY></HTML>
```

Save the above changes made to **logincheck.php** and run **login.html** via a Web Browser. When the login form appears enter the login information and click Login to submit data. The resulting page will appear as one of the following possibilities:

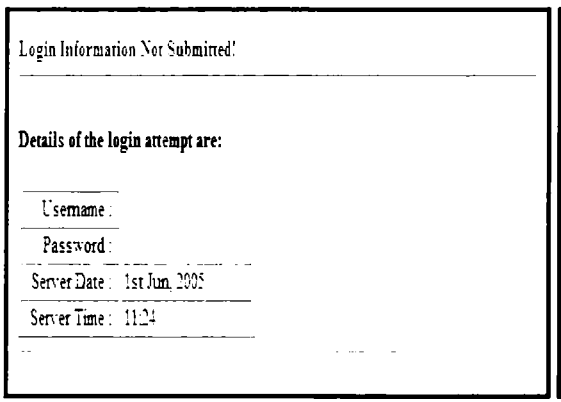


Diagram 14.4.1: Message by the logincheck.php, displayed when the login information is not submitted.

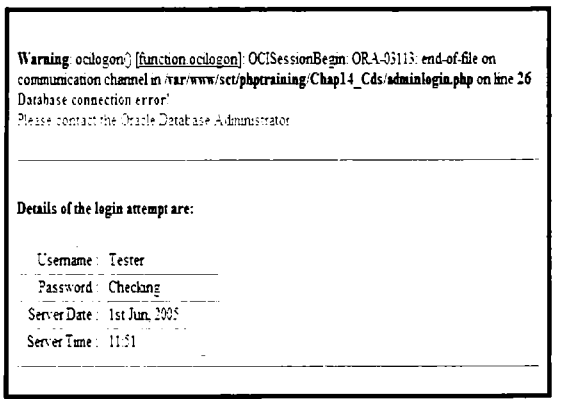


Diagram 14.4.2: Message by the logincheck.php, displayed when the connection object to the Oracle database cannot be established.

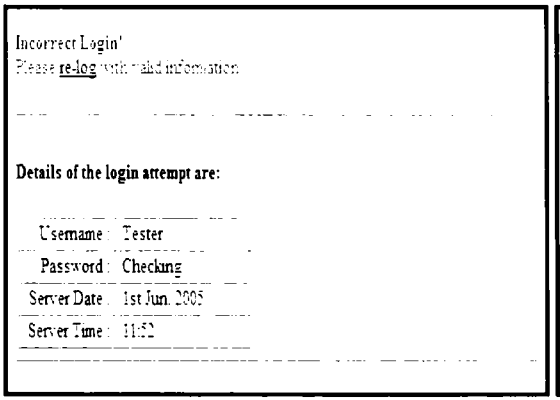


Diagram 14.4.3: Message by the logincheck.php, displayed for an invalid login attempt.

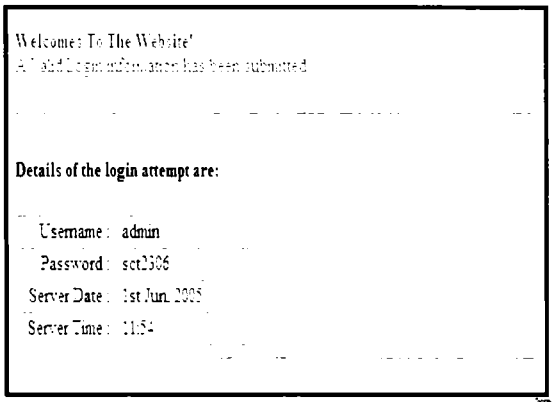


Diagram 14.4.4: Message by the logincheck.php, displayed for a valid login attempt.

The **logincheck.php** file retrieves data passed to it via **login.html**. PHP then begins processing the instructions contained within the file. The processing is as follows:

- Variables **$dbuser** and **$dbpass** are initialized and used to store the oracle user name and its corresponding password respectively
- The username and password fields are verified to see if they hold data. If either one is found empty, an error message indicating an invalid login is generated and stored into a variable named **$Message** which will be rendered later. Refer diagram 14.4.1
- If a value has been passed for both the username and password, an attempt is made to connect to and access the **Oracle Database**. A PHP function **ocilogon()** is used to perform the database connection based on **$dbuser** and **$dbpass** variables passed as parameter to create the connection object. This connection object obtained is stored in the variable **$rcq**
- A check is made whether the variable **$rcq** holds a valid connection object. If not, an error message indicating an error in database connectivity is generated and stored into a variable named **$Message** which will be rendered later. Refer diagram 14.4.2
- If creation of the connection object succeeds:
  - A SQL query is built and executed to validate if the username and password captured exists in the database
  - If the query retrieves the record, a message is displayed, which indicates a **Valid** login attempt. Refer diagram 14.4.4
  - If the query does not retrieve the record, an error message indicating **Incorrect** login attempt is displayed with a link to the login page. Refer diagram 14.4.3
- Finally, an HTML page is generated to display the value held by the variable **$Message** along with the details of the login information

Changing Category Master Module To Support A Database

Retrieving information from a database is done using a SELECT statement. Similarly INSERT, UPDATE and DELETE statements are required to manipulate table information held within the Oracle database.

Having dealt with reading from a database, the same technique can be used to display data in tabular layout for the Category Master form. Consider the following table structure for storing information bound to the Category Master:

Table Definition:

Table Name : **CATEGORY**
Primary Key : CATEGORY_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CATEGORY_ID	NUMBER	2	NOT NULL	
CATEGORY_NAME	VARCHAR2	35	NOT NULL	
CATEGORY_REMARKS	VARCHAR2	255		NULL

Table Description:

Column Name	Description
CATEGORY_ID	Internal ID
CATEGORY_NAME	Name of the category
CATEGORY_REMARKS	Remarks for the category if any

Explanation:
The CATEGORY table is a lookup table, which records the job categories available.

The SQL statement used to create the table with the above specifications is as follows:

```
CREATE TABLE category (
  category_ID Number(2) NOT NULL,
  category_name VarChar2(35) NOT NULL,
  category_remarks VarChar2(255) DEFAULT NULL,
  PRIMARY KEY(category_ID)
);
```

The following SQL statements will populate the **CATEGORY** table with sample data:

```
INSERT INTO category (category_ID, category_name, category_remarks)
  VALUES (1, 'Continental Cook', 'Specilised in continental preparations');
INSERT INTO category (category_ID, category_name, category_remarks)
  VALUES (2, 'Arabic Cook', 'Specilised in Irani preparations');
```

```
INSERT INTO category (category_ID, category_name, category_remarks)
  VALUES (3, 'Analyst Programmer', 'Specilised in Irani preparations');
INSERT INTO category (category_ID, category_name, category_remarks)
  VALUES (4, 'Scaffolding Foreman', 'Well experienced personnel in
            construction and development');
INSERT INTO category (category_ID, category_name, category_remarks)
  VALUES (5, 'Cabin Steward', '');
```

To understand the development of the Category master form, glance through the pseudo structure for the same as shown below:

```
<?
```

1	Commented statements providing a description and purpose of the file.

```
/*
  Date :- <Creation Date >
  Author :- <Developer's Name>
  Filename :- <Filename>
  Purpose :- <Brief Description About File Functionality>
*/
```

2	Common PHP variables.

```
/* A variable holding the developer's name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Category Master Form";
```

3.1	PHP variables bound to Database.

3.2	PHP variables bound to HTML page aesthetics.

4	PHP code block to establish Database connection.

5	PHP code block for data manipulation on establishing connection.

5.1	PHP code block to perform the INSERT operation.

```
    /* Verifying that the form is currently in the INSERT mode.
    */
       if ($hidMode == "I") {
```

5.1.1	PHP code block controlling INSERT operation based on duplication.

5.2	PHP code block to perform the UPDATE operation.

```
/* Verifying that the form is currently in the UPDATE mode.
*/
   if ($hidMode == "U") {
```

5.2.1	PHP code block controlling UPDATE operation based on duplication.

5.3	PHP code block to perform the DELETE operation.

```
/* Checking if the form is in the delete mode. */
   if ($hidMode == 'D') {
```

5.3.1	PHP code block to retrieve multiple records selection for deletion.

6	PHP variables re-initialization bound to the page processing/layout.

```
/* A variable named $hidMode is initialised to 'I'. This variable
holds the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
/* A variable $count is initialised to hold zero. This is used in
while loops to generate unique checkbox names for the tabular
layout. */
  $count = 0;
?>
```

7	HTML code block rendering the actual page begins.

```
<HTML>
<HEAD>
  <TITLE><?=$title;?></TITLE>
```

7.1	META tags bound to the HTML page.

7.2	SCRIPT tag holding JavaScript under the HEAD section.

```
</HEAD>
```

8	BODY section.

```
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
```

9	SCRIPT tag holding JavaScript unique to the current page under the BODY section.

9.1 JavaScript function to check contents of form field before submitting data contained in them.

```
/* The chkBlanks() function verifies that the form field of
category name is not left empty. If the field is empty or
contains blank spaces, a message indicates to enter a valid
value. If the field is not empty, the data is submitted. */
   function chkBlanks() {
```

9.2 JavaScript function to validate contents of form field before submitting data contained in them.

```
/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
   function vldtFrmFlds() {
```

9.3 JavaScript function to prepare the form to accept a new set of data.

```
/* The function setNewMode() is called to prepare the form to
except a new set for data. This function is called directly
when the Clear button is clicked. */
   function setNewMode() {
```

9.4 JavaScript function to prepare the form to accept changes in the selected set of data.

```
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
   function setEditMode(id, name, rmrk) {
```

9.5 JavaScript function to prepare the form to delete selected set(s) of data.

```
/* The function setDelMode() is called to prepare the form to
remove data from the tabular layout. This function is called
directly when the Delete button is clicked. */
   function setDelMode() {
```

9.6 SCRIPT tag holding JavaScript under the BODY section ends.

```
</SCRIPT>
```

10	**Parent Table begins.**

```
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="<?=$gl_mstr_frst_bar_color;?>" Border="0"
         CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="90%">
```

10.1	**Parent Table → First Row:** **Columns to hold page title and a message header.**

Category	<Message from the previous operation>

Diagram 14.5.1

11	**Form Section holding Data and Control objects begins.**

```
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="<FileName>" Method="post" Name="frmMstrCat"
        onSubmit="return chkBlanks();">
```

11.1	**Form Hidden Variable to identify the current page.**

```
  <!-- Declaring a hidden form field used to identify the current
  (i.e. category master) page. -->
    <INPUT Name="hidPage" Type="hidden" Value="<Page Name>">
```

11.2	**Form Hidden variables to maintain a copy of the data held by the form fields just before changes attempted. This is useful to confirm change(s) made to the form fields before an update is fired. (To avoid database or flat files concern as the case may be)**

```
  <!-- Declaring hidden form fields required for data validation.
  -->
    <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
    <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
    <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
```

11.3	**Form Hidden Variable to determine the form operation. Default, being Insert operation.**

```
  <!-- Declaring a hidden form field used for determining the
  form mode. It holds 'I' for Insert, 'U' for Update and 'D' for
  Delete. It will hold 'I' when the page is rendered for the 1st
  time. -->
    <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
```

11.4	Form Hidden Variable holding entries (checkbox values) selected for deletion.

```
<!-- Declaring a hidden form field used for identifying records
which have been selected for deletion. -->
   <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
```

12	Parent Table → Second Row: Columns to hold the actual d/e form.

```
<TR Height="300" VAlign="top">
   <TD Align="center" Border="1" ColSpan="10"><BR>
```

13	First Child Table holding d/e Form objects, to Capture / Control data begins.

```
   <!-- Form Table Code Begins. -->
      <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
             Name="TIbInner" Width="90%">
```

13.1	First Child Table → First and Second Rows: Columns to hold form title.

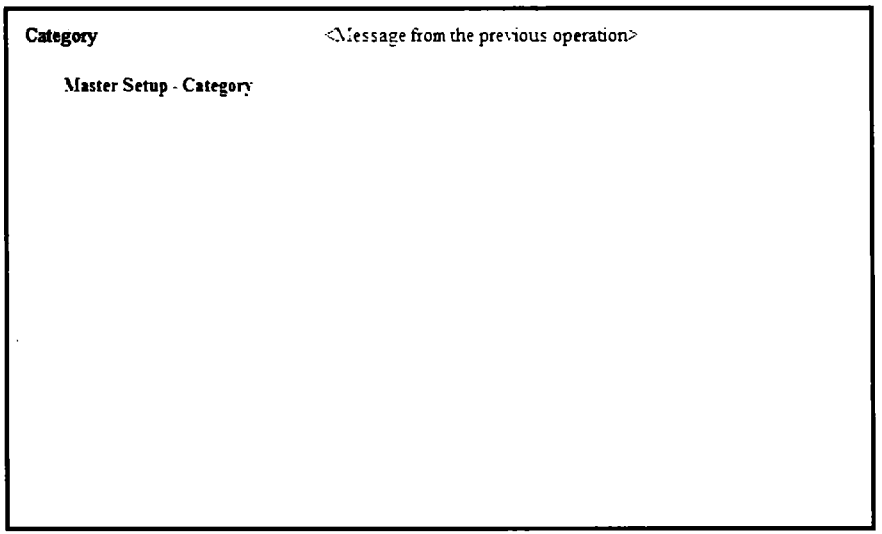


Diagram 14.5.2

13.2	First Child Table → Third & Forth Rows: Columns holding label and data capture objects to capture data.

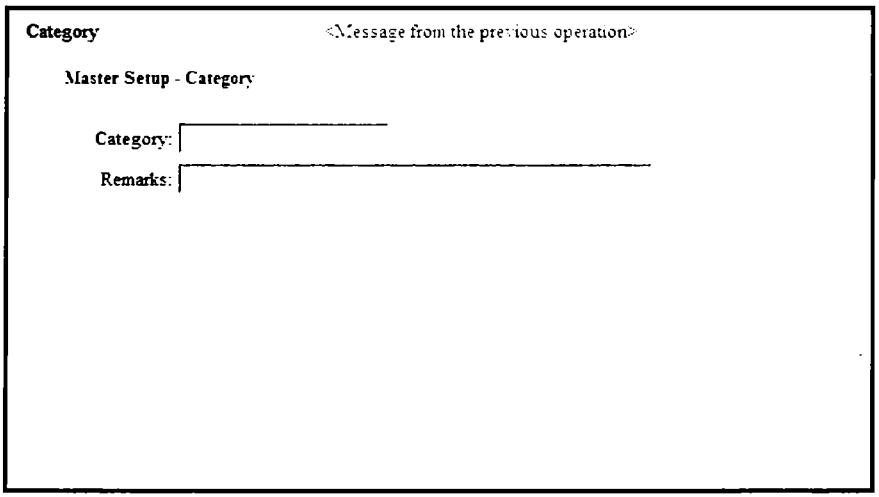


Diagram 14.5.3

13.3	First Child Table → Last Row: Columns holding data control objects.

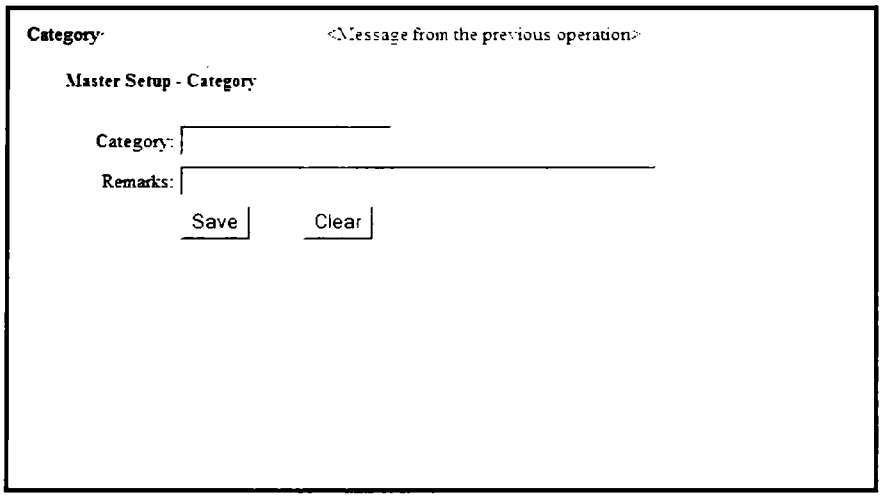


Diagram 14.5.4

13.4	First Child Table ends.

```
<?
```

14	PHP code block retrieving data to create the tabular layout by Database connection. The tabular layout is generated only after data gets retrieved.

```
?>
```

15	Second Child Table holding the tabular layout, to display / control data captured begins.

```
<!-- Data Table Code Begins. -->
    <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
           Width="90%">
```

16	Second Child Table → First Row: Columns to header row.

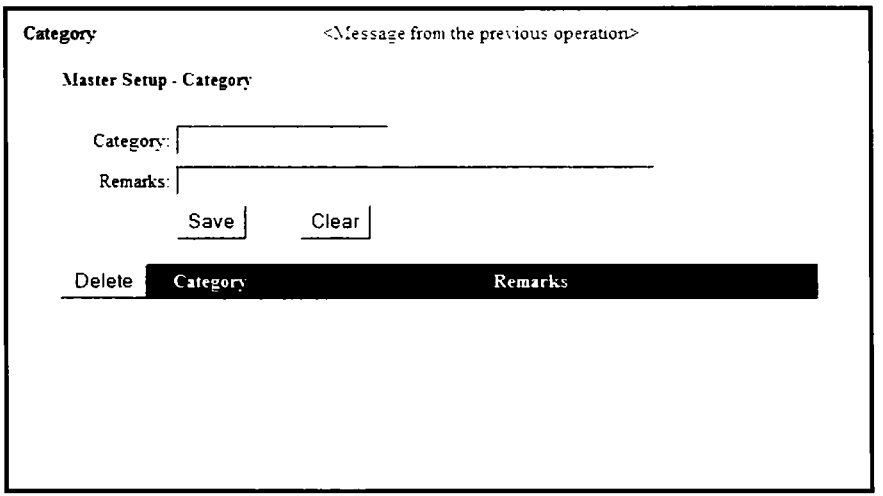


Diagram 14.5.5

```
<?
```

17	PHP code block to populate data in the tabular layout using a loop.

```
 /* A variable named $rowBgColor hold the hexadecimal colour
value. This variable is used decided the background colour of
rows for data in the tabular format. */
    $rowBgcolor = $gl_mstr_scnd_bar_color;
 /* An iterative code performed as long as a single row of
    category data is available. */
    do {
    /* Incrementing the value of the variable used for assigning
    unique names to check boxes display of each row of data. */
       $count = $count + 1;
```

17.1	PHP code block to control the background colour for the current row.

```
    /* If the colour code in for the current row matches the
    value for the second row. */
       if ($rowBgcolor == $gl_mstr_scnd_bar_color) {
```

```
        /* Colour code for the current row set to match the value
        for the colour code for the first bar. */
           $rowBgcolor = $gl_mstr_frst_bar_color;       }
     /* If the colour code in for the current row matches the
     value for the first row. */
        else {
        /* Colour code for the current row set to match the value
        for the colour code for the second bar. */
           $rowBgcolor = $gl_mstr_scnd_bar_color;       }
?>
```

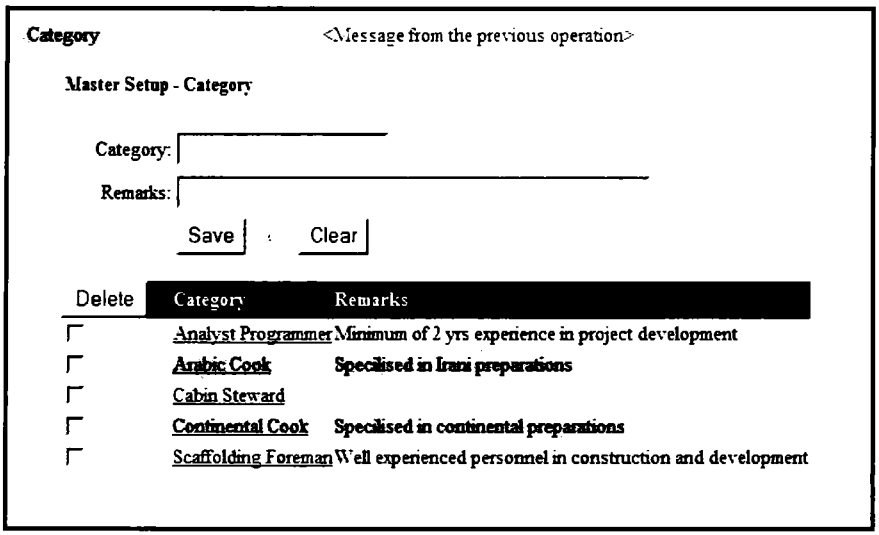


Diagram 14.5.6

17.2	**Loop and the Second Child table ends.**

```
<?      } while(ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM))   ?>
        </TABLE><BR>
```

18	**Form holding Data and Control objects ends.**

```
</FORM>
```

19	**Parent Table ends.**

```
</TABLE>
<!-- Outer Table Code Ends. -->
```

20	**HTML code block along with BODY section rendering the actual page ends.**

```
</BODY></HTML>
```

To begin building database support to the Category Master Module created earlier, the **ctgry_mstr.php** file will be modified as specified below. To avoid rewriting the entire page create a copy of the **ctgry_mstr.php** and **mstrscript.js** files *from the* ***Chap12_Cds*** *directory and store it in the* ***Chap14_Cds*** *directory.*

Open the **ctgry_mstr.php** file in an ASCII editor and perform the following change in the PHP code block:

- Delete the following block of PHP code, which verifies the form delete mode (i.e. $hidMode holds D). This is done as the code spec for the delete operation will be built later. The block of PHP code to be deleted is found in the declaration section. (Refer point **5.3** under the sub-title **Skeleton Of The Code Undertaken For Development**.)

```
...
/* A variable holding the title for the page. */
  $title = "Category Master Form";
/* If the form mode is set to Delete. */
  if ($hidMode == 'D') {
     /* A variable $Message is initialised to hold a message,
     which has occurred during the previous (delete) operation.
     */
        $Message = '<BR><FONT Color="RED">* Details for '.$hidDelRcrdLst.' has
                   been deleted.</FONT>';     }
/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

- Insert the code block mentioned below, before the initialization of the **$hidMode** variable. The inserted code will create an Oracle connection object and then check whether it was really created. If the creation fails an error message is generated and stored in the variable **$Message**. (Refer points **3.1 and 4** under the sub-title **Skeleton Of The Code Undertaken For Development**.)

```
...
/* A variable holding the title for the page. */
  $title = "Category Master Form";
/* Variables used to store Database access information. */
  $dbuser = "dba_pm";
  $dbpass = "sct2306";
/* The function ocilogon() accepts the oracle login information
to create the actual connection to the database. This connection
made available whenever required by referencing $rcq, which the
connection object created by the ocilogon() function. */
  $rcq = ocilogon($dbuser, $dbpass);
/* Checking whether the creation of the connection object failed
or not. If failed, an error message is generated. */
  if(!$rcq){
```

```
    $Message = '<FONT Color="red"><B>Database connection
                error!</B><BR>Please contact the Oracle Database
                Administrator.</FONT><BR>';  }
/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

- Scroll to the section where the HTML code block defining the table holding the data capture form ends. This block is immediately followed by PHP code spec, which checks if the form field holds a value. (Refer point **14** under the sub-title **Skeleton Of The Code Undertaken For Development.**). Replace this code block with the following which retrieve data from the category table and stores the same in to a record set:

```
...
                <INPUT Name="cmdReset" onClick="setNewMode();" Type="button" Value="Clear">
            </TD>
        </TR></TABLE><BR>
    <!-- Form Table Code End. -->
```

```
<?
/* Building and executing a SQL query to retrieve records from
the CATEGORY table after sorting the category names in alphabetic
order. */
  $query = "SELECT * FROM category ORDER BY category_name";
  $parsed = ociparse($rcq, $query);
  ociexecute($parsed);
/* Checking if records have been retrieved from the Category
table. */
  if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
?>
```

```
    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%"><TR
                BgColor="#400040">
            <TD Width="15%"><INPUT Name="cmdDelete" onClick="setDelMode();" Type="button"
                Value="Delete"></TD>
...
```

- The above block is followed by the HTML code block, which defines the header row of the tabular layout (grid). Replace the code block that follows immediately after the declaration of the header row upto the end of the column (i.e. the **</TD>** tag) before the **</FORM>** tag. (Refer points **17, 17.1 and 17.2** under the sub-title **Skeleton Of The Code Undertaken For Development.**) The new code spec that will appear will populate the grid i.e. the tabular layout with the records retrieved using a loop:

```
...
    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%"><TR
                BgColor="#400040">
        ...
        </TR>
```

```
<?
/* An iterative code performed as long as a single row of
category data is available. */
     $count = 0;

     do {
        $count = $count + 1;
?>
        <TR BgColor="#E4E4E4">
           <TD><INPUT Name="chk<?=($count);?>" Type="checkbox"
                Value="<?=($line['CATEGORY_ID']);?>"></TD>
           <TD><A
                HRef="JavaScript:setEditMode('<?=$line['CATEGORY_ID'];?>,
                '<?=$line['CATEGORY_NAME'];?>',
                '<?=$line['CATEGORY_REMARKS']?>')"><?
                echo($line['CATEGORY_NAME']); ?></A></TD>
           <TD><? echo($line['CATEGORY_REMARKS']); ?></TD>
        </TR>
<?      } while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM))  ?>
        </TABLE><BR>
<?   }  ?>
```

```
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends.-->
</BODY>
</HTML>
```

Save the changes made to **ctgry_mstr.php** and test the changes by calling the file in a Web Browser.

When the page appears, as shown in diagram 14.6.1, the d/e form along with the populated tabular layout is displayed. The data displayed in the tabular layout is retrieved from the Oracle table named CATEGORY.

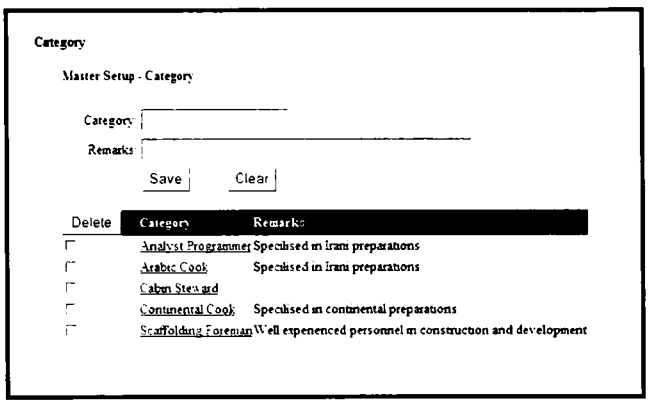


Diagram 14.6.1: Output for ctgry_mstr.php.

REMINDER

In addition to retrieving data from the CATEGORY table the SELECT statement will sort the records on the basis of CATEGORY_NAME column.

Enhancement To Page Aesthetics

Having managed to display data for the Category Master module, an attempt to improve the appearance of the page rendered by **ctgry_mstr.php** can be made. The simplest change would be to display a different background colour for alternate rows in the tabular layout. Refer diagram 14.6.2.

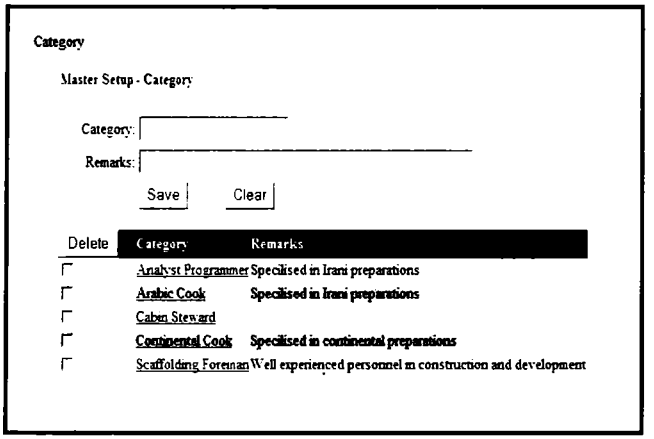


Diagram 14.6.2: New Tabular Layout for ctgry_mstr.php.

To achieve this open the file in an ASCII editor and add the following code blocks:

- Assign hexadecimal colour code to variables by mentioning them in the declaration section of the file, (Refer point **3.2** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
/* A variable holding the title for the page. */
   $title = "Category Master Form";
```

```
/* Variables used to store colours for the master page. */
   $gl_mstr_top_bar_color = '#400040';
   $gl_mstr_frst_bar_color = '#E4E4E4';
   $gl_mstr_scnd_bar_color = '#C0C0C0';
```

```
/* Variables used to store Database access information. */
   $dbuser = "dba_pm";
   $dbpass = "sct2306";
...
```

- In the outer table's **<TABLE>** tag, change value of the **BgColor** attribute as mentioned below, (Refer point **10** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
   </SCRIPT>
<!-- Outer Table Code Begins. -->
```

```
   <TABLE      Align="center"      BgColor="<?=$gl_mstr_frst_bar_color;?>"
Border="0" CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="90%"><TR>
```

```
      <TD Align="center" Border="1" BgColor="#C0C0C0" Width="10%">Category</TD>
      <TD Align="center" BgColor="#FFFFFF" ColSpan="9" Width="90%"><?=$Message;?></TD>
   </TR>
...
```

- In the **<TR>** tag of the Header row in the tabular layout, change the value for the **BgColor** attribute as mentioned below, (Refer point **16** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
      <!-- Data Table Code Begins. -->
          <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Width="90%">
              <TR BgColor="<?=$gl_mstr_top_bar_color;?>">
                  <TD Width="15%"><INPUT Name="cmdDelete" onClick="setDelMode();" Type="button"
                      Value="Delete"></TD>
...
```

- Replace the PHP code block immediately after the declaration of the header row upto the declaration of the data row (i.e. the second row in the tabular layout). (Refer points **17 and 17.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**) The following code spec actually alternates between the two colors defined:

```
...
              <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
              <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
          </TR>
<?
  /* A variable named $rowBgColor hold the hexadecimal colour
  value. This variable is used decided the background colour for
  rows of data in the tabular format. */
     $rowBgcolor = $gl_mstr_scnd_bar_color;
  /* An iterative code performed as long as a single row of
  category data is available. */
     do {
     /* Incrementing the value of the variable used for assigning
     unique names to check boxes display of each row of data. */
        $count = $count + 1;
     /* If the colour code in for the current row matches the
     value for the second row. */
        if ($rowBgcolor == $gl_mstr_scnd_bar_color) {
        /* Colour code for the current row set to match the value
        for the colour code for the first bar. */
           $rowBgcolor = $gl_mstr_frst_bar_color;       }
     /* If the colour code in for the current row matches the
     value for the first row. */
        else {
        /* Colour code for the current row set to match the value
        for the colour code for the second bar. */
           $rowBgcolor = $gl_mstr_scnd_bar_color;       }
?>
          <TR BgColor="#E4E4E4">
              <TD><INPUT Name="chk<?=($count);?>" Type="checkbox"
Value="<?=($line['CATEGORY_ID']);?>"></TD>
...
```

- In the **<TR>** tag of the data row in the tabular layout, change the value of the **BgColor** attribute as mentioned below, (Refer point **17.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
            else {
            /* Colour code for the current row set to match the value for the colour
            code for the second bar. */
                $rowBgcolor = $gl_mstr_scnd_bar_color;        }
?>
```

```
            <TR BgColor="<?=$rowBgcolor;?>">
```

```
                <TD><INPUT Name="chk<?=($count);?>" Type="checkbox"
                        Value="<?=($line['CATEGORY_ID']);?>"></TD>
...
```

This completes the changes for the new appearance of the tabular layout in the Category Master page.

Using PHP To Insert Data Into The Database

The change brought in by the above code block is limited to retrieving data from the database and displaying with aesthetics. Apart from this the form does not allow any data entry. This means that data entered into the form fields will be lost unless it is saved into the database.

The illustration below improvises the Category master page and allows it to use the Oracle Database to store data (captured by the form fields). To continue with the illustration create a copy of the **ctgry_mstr.php** file.

Insert the following PHP code block immediately after the code spec that verifies the connection object at the beginning of the file, (Refer point **5, 5.1 and 5.1.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
/* Checking whether the creation of the connection object failed or not. If
failed, an error message is generated. */
  if(!$rcq){
      $Message = '<FONT Color="red"><B>Database connection error!</B><BR>Please contact the Oracle
                  Database Administrator.</FONT><BR>';
  }
```

```
/* If the creation of the connection object succeeded. */
  else {
  /* If Category name has been passed through the variable
  $txtCatName, (i.e. the $txtCatName is not empty). */
     if (!empty($txtCatName)) {
     /* Verifying that the form is currently in the INSERT mode.
     */
        if ($hidMode == "I") {
```

```
        /* Building and executing a SQL query to retrieve records
        for the Category table such that the contents of the
        Category_Name matches the value entered in the category
        name field in the form. */
            $query = "SELECT * FROM Category WHERE
                    TRIM(LOWER(category_name))=TRIM(LOWER('"
                    .$txtCatName."'))";
            $parsed = ociparse($rcq, $query);
            ociexecute($parsed);

        /* Checking if any record was retrieved by the above SQL
        query. If the same category name exists in the database,
        an error message is displayed to the user, which
        indicates that the Insert operation has failed. */
            if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
            /* Generating and storing an error message which
            indicates a duplication of category. */
                $Message = '<FONT Color="red">Entry for
                            <B>'.$txtCatName.'</B> already exists. </FONT>';
            }
        /* When the same category name does not exist in the
        databases, the new category is stored into the database.
        */
            else {
            /* Building and executing a SQL query to INSERT the
            new category into the category table. The new category
            is assigned a unique ID, which is the highest ID
            assigned to a category incremented by one. */
                $query = "INSERT INTO category (category_ID, category_name,
                        category_remarks) VALUES((SELECT
                        COALESCE(MAX(category_ID), 0)+1 FROM
                        category),'".$txtCatName."', '".$txtCatRmrk."')";
                $parsed = ociparse($rcq, $query);
                ociexecute($parsed);
            /* Generating and storing a message that indicates the
            insert operation was successful. */
                $Message = '<FONT Color="blue">Entry for
                            <B>'.$txtCatName.'</B> added
                            successfully.</FONT>';
            }
        }
    }
  }
```

```
/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

Save the changes made to the **ctgry_mstr.php** file and run it within the web browser. When the page appears enter a new category and click Save. Refer diagram 14.7.1.

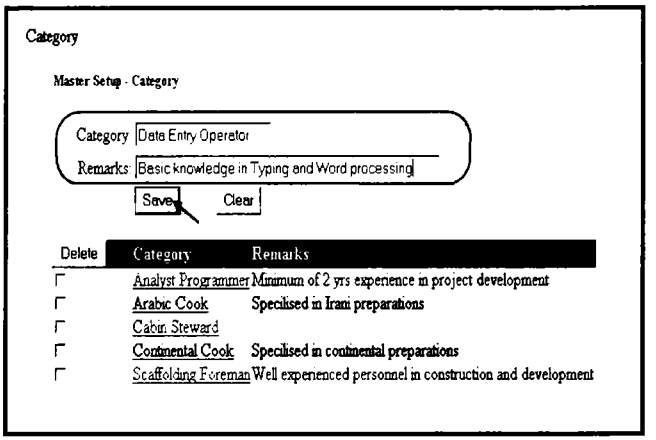


Diagram 14.7.1: Adding a new entry with category information via the ctgry_mstr.php file.

When a call is made to the **ctgry_mstr.php** file with new category information the following processes occur:

- A check is made to ensure that **$rcq** holds a reference to the Oracle connection object
- A check is made to ensure that data is submitted by the previous instance of the page:
  - A check is made to ensure that the form is in the insert mode (**i.e. $hidMode** holds the value **I**). If yes, then:
    - An SQL query is built and executed to check for duplication in the record submitted. This is done by retrieving the data filtered on the basis of the new category submitted
    - If the above SQL query retrieved any record(s) i.e. the new category name submitted already exists in the database:
      - An error message, indicating an Insert operation failure due to duplication of category name is generated and stored into **$Message**
    - If no records are returned by the above SQL query:
      - An SQL query is built and executed to INSERT the new category into the category table. The new category is assigned a unique ID
      - A message indicating a successful insert is generated and stored into **$Message**
- Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:
  - Displaying the value stored in **$Message**
  - Laying out the form for capturing category information
  - Laying out the grid for displaying category information stored in category table

The new category will be listed as an entry in the tabular layout / grid. Refer diagram 14.7.2.

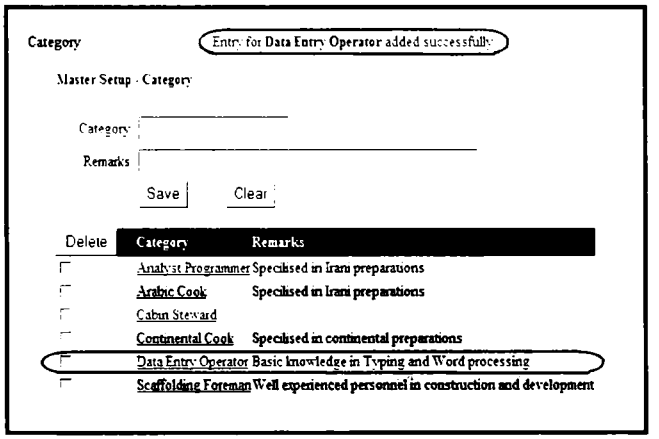


Diagram 14.7.2: Entry added by ctgry_mstr.php.

Using PHP To Change Data Held In The Database

A successful Insert operation is followed by building the update operation for the form. This means that data held in the database will be accessed and changed. The changes will then be reflected in the table by updating the data held within them.

Clicking on the hyper-linked text (in the second column in the tabular layout) will invoke the form's update mode. The values bound to the entry, whose hyperlink is clicked, is passed to the form fields permitting data modification.

To illustrate the Update technique using database tables, the code in **ctgry_mstr.php** will be used as base. Insert / modify PHP code block as mentioned below:

- After the PHP code block checking for form's insert mode ends, (Refer points **5.2 and 5.2.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**) place the following:

```
...
            /* Generating and storing a message that indicates the insert
            operation was successful. */
                $Message = '<FONT Color="blue">Entry for <B>'.$txtCatName.'</B> added
                        successfully.</FONT>';
            }
        }
```

```
    /* Verifying that the form is currently in the UPDATE mode.
    */
        if ($hidMode == "U") {
        /* Building and executing a SQL query to retrieve records
        for the category table such that the contents of the
        Category_Name matches the value entered in the category
        name field in the form, while the unique record number
        does not match the unique record number held in the
        hidden form field $hidCatID. */
            $query = "SELECT * FROM category WHERE
                    TRIM(LOWER(category_name))=TRIM(LOWER('".$txtCat
                    Name."')) AND NOT (category_ID=".$hidCatID.")";
            $parsed = ociparse($rcq, $query);
            ociexecute($parsed);
```

```
        /* Checking if any record was retrieved by the above SQL
        query. If the same category name exists in another
        record, an error message is displayed to the user, which
        indicates that the Update operation has failed. */
            if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
            /* Generating and storing an error message which
            indicates a duplication of category. */
                $Message = '<FONT Color="red">The modified category
                            <B>'.$txtCatName.'</B> already exists.</FONT>';
        }
        /* When the same category name does not exist in another
        record, the modified category data is stored into the
        table. */
            else {
            /* Building and executing a SQL query to Update the
            category table with the change in the category data.
            The record to be changed is determined on the bases of
            the unique ID assigned to the category data. */
                $query = "UPDATE category SET
                        category_name='".$txtCatName."',
                        category_remarks='".$txtCatRmrk."' WHERE
                        category_ID=".$hidCatID;
                $parsed = ociparse($rcq, $query);
                ociexecute($parsed);
            /* Generating and storing a message that indicates the
            update operation was successful. */
                $Message = '<FONT Color="blue">Entry for
                            <B>'.$txtCatName.'</B> modified
                            successfully.</FONT>';
            }
        }
    }
}
/* A variable named $hidMode is
initialised to 'I'. This
variable holds the form status
in terms of I - Insert, U -
Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

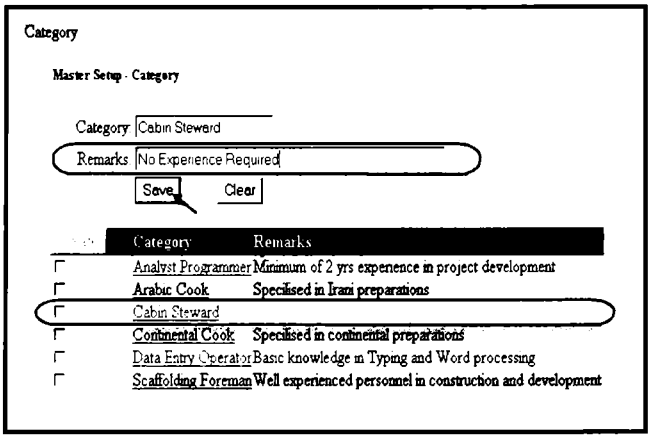


Diagram 14.8.1: Editing an entry with category information via the ctgry_mstr.php file.

Save the changes made to the **ctgry_mstr.php** file and run it within the web **browser**. When the page appears click on one of the hyperlinks in the tabular layout. This will populate form fields with the data bound to the selected entry, modify the data as desired and click Save. Refer diagram 14.8.1.

When a call is returned to the **ctgry_mstr.php** file with the modified category information the following processes occur:

- A check is made to ensure that **$rcq** holds a reference to the Oracle connection object
- A check is made to ensure that category name field holds data. If it holds data:
  - A check is made to ensure that the form is in the update mode, (**i.e. $hidMode** holds the value **U**). If yes, then:
    - The newly entered record details, in this case the category name, is checked for duplication against the data retrieved from the file. Additionally it is ensured that the category name duplication check is avoided against the same record.
    - If the category name is being duplicated
      - An error message, indicating an Update operation failure due to duplication of category name is generated and stored into **$Message**
    - If there is no duplication taking place:
      - A SQL query is built and executed which will perform the update operation on the category table
      - A message, indicating a successful update, is generated and stored into **$Message**

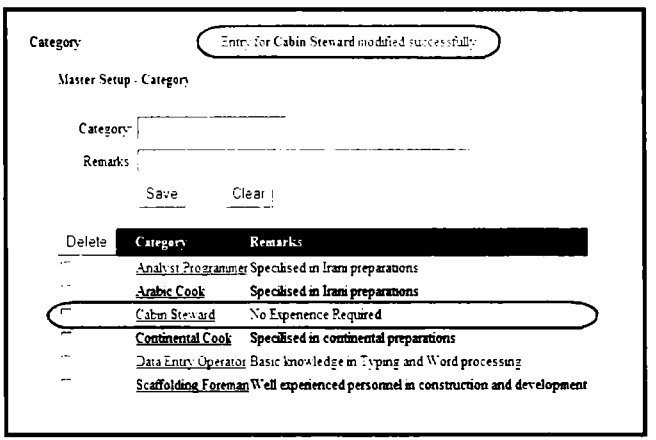


Diagram 14.8.2: Entry changed by ctgry_mstr.php.

- Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:
  - Displaying the value stored in **$Message**
  - Laying out the form for capturing category information
  - Laying out the grid for displaying category information stored in category table

The modified category will be listed as an entry in the tabular layout / grid. Refer diagram 14.8.2.

Using PHP To Remove Data Held In The Database

Having completed the data insert and the data manipulation operations using database tables, it is time to move on to the delete operation. Although, the delete functionality at the HTML level will remain the same, the change occurs in the PHP processing.

When Delete is clicked, a list of record(s) marked for deletion is created and submitted to the next instance of the same page.

As the database holds category data, when a list of records selected for deletion is received from the page's previous instance, the delete operation requires the following modification in the PHP code block:

- After the PHP code block, which checks for captured data submitted via **$txtCatName** ends, replace the check for the page in delete mode with the following, (Refer point **5.3 and 5.3.1** under the sub-title **Skeleton Of The Code Undertaken For Development.**):

```
...
  /* If Category name has been passed through the variable $txtCatName, (i.e. the
  $txtCatName is not empty). */
      if (!empty($txtCatName)) {
      /* Verifying that the form is currently in the INSERT mode. */
          if ($hidMode == "I") {
...
              }
          }

      /* Verifying that the form is currently in the UPDATE mode. */
          if ($hidMode == "U") {
...
          }
      }
  }
```

```
  /* Checking if the form is in the delete mode. */
      if ($hidMode == 'D') {
      /* Building and executing a SQL query to retrieve category
      names from the category table such that the unique record
      number matches the entries selected for deletion, (i.e.
      value held in $hidDelRcrdLst). */
          $query = "SELECT category_name FROM category WHERE category_ID
                    IN(".$hidDelRcrdLst.")";
          $parsed = ociparse($rcq, $query);
          ociexecute($parsed);

      /* An iterative code performed as long as a single row of
      category data is available. */
          while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) {
          /* Checking if the variable $delEntry is empty, to
          determine the method for storing the current category
          into the variable $delEntry. */
              if(empty($delEntry)) {
                  $delEntry = $line['CATEGORY_NAME'];     }
              else {  $delEntry = $delEntry.", ".$line['CATEGORY_NAME'];  }
          }
      /* Checking if the variable $delEntry is not empty. */
          if(!empty($delEntry)) {
```

```
        /* Building and executing a SQL query to delete record(s)
        for the category table. The condition of deleting
        record(s) is that the unique record number should exists
        in the list of entries selected for deletion, (i.e. value
        held in $hidDelRcrdLst). */
            $query = "DELETE FROM category WHERE category_ID
                    IN(".$hidDelRcrdLst.")";
            $parsed = ociparse($rcq, $query);
            ociexecute($parsed);
        /* A variable $Message is initialised to hold a message,
        which list the record(s) deleted. */
            $Message = '<BR><FONT Color="RED">Details for
                        <B>'.$delEntry.'</B> has been deleted.</FONT>';
        }
    /* If the variable $delEntry is empty. */
        else {
        /* A variable $Message is initialised to hold an error
        message, which indicates that records for deletion do not
        exists in the database. */
            $Message = '<BR><FONT Color="RED">Records selected for deletion
                        do not exists. </FONT>';
        }
    }
  }
/* A variable named $hidMode is initialised to 'I'. This variable holds the form
status in terms of I - Insert, U - Update and D - Delete. */
  $hidMode = 'I';
?>
...
```

Save the changes made to the **ctgry_mstr.php** file and run it within the web browser. When the page appears, select the checkbox(es) bound to the records desired to be deleted from the tabular layout and click Delete. Refer diagram 14.9.1.

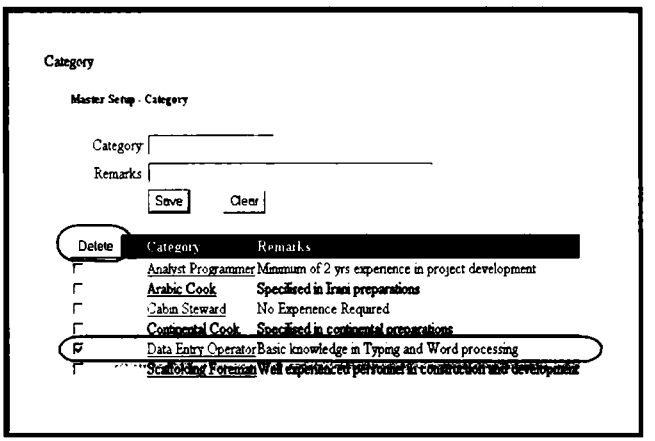


Diagram 14.9.1: Deleting an entry with category information via the ctgry_mstr.php file.

When a call is returned to the **ctgry_mstr.php** file with list of categories selected for deletion the following processes occur:

- A check is made to ensure that **$rcq** holds a reference to the Oracle connection object
- A check is made to ensure that the form is in the delete mode, (**i.e. $hidMode** holds the value **D**). If yes, then:

- An SQL query is built and executed to retrieve category names from the category table for the unique record number held in **$hidDelRcrdLst**
- A loop traverses through the record(s) retrieved by the above query to perform the following:
  - The category names retrieved by the above query is stored in a comma separated form in the **$delEntry** variable. This variable will hold the category names to be displayed as the records deleted after the actual delete operation is performed
- A check is made to ensure that the variable **$delEntry** holds a value. If yes, then:
  - An SQL query is built and executed to delete record(s) from the category table based on the unique record values held in **$hidDelRcrdLst**
  - A message indicating a successful delete along with the category names deleted is generated and stored into **$Message**
- If the variable **$delEntry** does not hold a value, a message indicating that records selected for deletion do not exists, is generated and stored into **$Message**

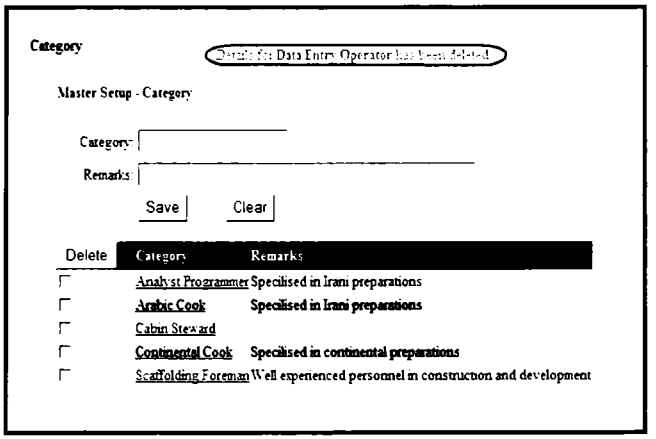


Diagram 14.9.2: Entry deleted by ctgry_mstr.php.

- Finally, the functionality of rendering a HTML page for **ctgry_mstr.php** is processed. This will include:
  - Displaying the value stored in **$Message**
  - Laying out the form for capturing category information
  - Laying out the grid for displaying category information stored in the **category** table found in the Oracle database

The name(s) of category deleted (using the values held by the **$delEntry** variable) will be listed as shown in diagram 14.9.2.
This completes the Category Master module using database table. The entire code for the **ctgry_mstr.php** appears as listed below:

```
<?
/*Date :- 28/05/05
  Author :- Hansel Colaco.
  Filename :- ctgry_mstr.php
  Purpose  :-  Allows  display,  insert,  updates  and  delete  of
  category master table.
*/
```

```
/* A variable holding the developers name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Category Master Form";

/* Variables used to store Database access information. */
  $dbuser = "dba_pm";
  $dbpass = "sct2306";

/* Variables used to store colours for the master page. */
  $gl_mstr_top_bar_color = '#400040';
  $gl_mstr_frst_bar_color = '#E4E4E4';
  $gl_mstr_scnd_bar_color = '#C0C0C0';

/* The function ocilogon() accepts the oracle login information
to create the actual connection to the database. This connection
made available whenever required by referencing $rcq, which the
connection object created by the ocilogon() function. */
  $rcq = ocilogon($dbuser, $dbpass);

/* Checking whether the creation of the connection object failed
or not. If failed, an error message is generated. */
  if(!$rcq) {
     $Message = '<FONT Color="red"><B>Database connection
                 error!</B><BR>Please contact the Oracle Database
                 Administrator.</FONT><BR>';
  }
/* If the creation of the connection object succeeded. */
  else {
  /* If Category name has been passed through the variable
  $txtCatName, (i.e. the $txtCatName is not empty). */
     if (!empty($txtCatName)) {
     /* Verifying if the form is currently in the INSERT mode. */
        if ($hidMode == "I") {
        /* Building and executing a SQL query to retrieve records
        for the Category table such that the contents of the
        Category_Name matches the value entered in the category
        name field in the form. */
           $query = "SELECT * FROM Category WHERE
                    TRIM(LOWER(category_name))=TRIM(LOWER('".$txtCat
                    Name."'))";
           $parsed = ociparse($rcq, $query);
           ociexecute($parsed);

        /* Checking if any record was retrieved by the above SQL
        query. If the same category name exists in the database,
        an error message is displayed to the user, which
        indicates that the Insert operation has failed. */
           if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
```

```
            /* Generating and storing an error message which
            indicates a duplication of category. */
                $Message = '<FONT Color="red">Entry for
                            <B>'.$txtCatName.'</B> already exists. </FONT>';
            }
        /* When the same category name does not exist in the
        databases, the new category is stored into the database.
        */
            else {
            /* Building and executing a SQL query to INSERT the
            new category into the category table. The new category
            is assigned a unique ID, which is the highest ID
            assigned to a category incremented by one. */
                $query = "INSERT INTO category (category_ID, category_name,
                        category_remarks) VALUES((SELECT
                        COALESCE(MAX(category_ID), 0)+1 FROM
                        category),'".$txtCatName."', '".$txtCatRmrk."')";
                $parsed = ociparse($rcq, $query);
                ociexecute($parsed);
            /* Generating and storing a message that indicates the
            insert operation was successful. */
                $Message = '<FONT Color="blue">Entry for
                            <B>'.$txtCatName.'</B> added
                            successfully.</FONT>';
            }
        }

    /* Verifying that the form is currently in the UPDATE mode.
    */
        if ($hidMode == "U") {
        /* Building and executing a SQL query to retrieve records
        for the category table such that the contents of the
        Category_Name matches the value entered in the category
        name field in the form, while the unique record number
        does not match the unique record number held in the
        hidden form field $hidCatID. */
            $query = "SELECT * FROM category WHERE
                    TRIM(LOWER(category_name))=TRIM(LOWER('".$txtCat
                    Name."')) AND NOT (category_ID=".$hidCatID.")";
            $parsed = ociparse($rcq, $query);
            ociexecute($parsed);

        /* Checking if any record was retrieved by the above SQL
        query. If the same category name exists in another
        record, an error message is displayed to the user, which
        indicates that the Update operation has failed. */
            if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
```

```
            /* Generating and storing an error message which
            indicates a duplication of category. */
                $Message = '<FONT Color="red">The modified category
                            <B>'.$txtCatName.'</B> already exists.</FONT>';
            }
        /* When the same category name does not exist in another
        record, the modified category data is stored into the
        table. */
            else {
            /* Building and executing a SQL query to Update the
            category table with the change in the category data.
            The record to be changed is determined on the bases of
            the unique ID assigned to the category data. */
                $query = "UPDATE category SET
                        category_name='".$txtCatName."',
                        category_remarks='".$txtCatRmrk."' WHERE
                        category_ID=".$hidCatID;
                $parsed = ociparse($rcq, $query);
                ociexecute($parsed);
            /* Generating and storing a message that indicates the
            update operation was successful. */
                $Message = '<FONT Color="blue">Entry for
                            <B>'.$txtCatName.'</B> modified
                            successfully.</FONT>';
            }
        }
    }

/* Checking if the form is in the delete mode. */
    if ($hidMode == 'D') {
    /* Building and executing a SQL query to retrieve category
    names from the category table such that the unique record
    number matches the entries selected for deletion, (i.e.
    value held in $hidDelRcrdLst). */
        $query = "SELECT category_name FROM category WHERE category_ID
                IN(".$hidDelRcrdLst.")";
        $parsed = ociparse($rcq, $query);
        ociexecute($parsed);
    /* An iterative code performed as long as a single row of
    category data is available. */
        while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) {
        /* Checking if the variable $delEntry is empty, to
        determine the method for storing the current category
        into the variable $delEntry. */
            if(empty($delEntry)) {
                $delEntry = $line['CATEGORY_NAME'];     }
            else { $delEntry = $delEntry.", ".$line['CATEGORY_NAME']; }
        }
```

```
    /* Checking if the variable $delEntry is not empty. */
       if(!empty($delEntry)) {
       /* Building and executing a SQL query to delete record(s)
       for the category table. The condition of deleting
       record(s) is that the unique record number should exists
       in the list of entries selected for deletion, (i.e. value
       held in $hidDelRcrdLst). */
          $query = "DELETE FROM category WHERE category_ID
                   IN(".$hidDelRcrdLst.")";
          $parsed = ociparse($rcq, $query);
          ociexecute($parsed);
       /* A variable $Message is initialised to hold a message,
       which list the record(s) deleted. */
          $Message = '<BR><FONT Color="RED">Details for
                   <B>'.$delEntry.'</B> has been deleted.</FONT>';
       }
    /* If the variable $delEntry is empty. */
       else {
       /* A variable $Message is initialised to hold an error
       message, which indicates that records for deletion do not
       exists in the database. */
          $Message = '<BR><FONT Color="RED">Records selected for deletion
                   do not exists. </FONT>';
       }
    }
  }
/* A variable named $hidMode is initialised to 'I'. This variable
hold the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
/* A variable $count is initialised to hold zero. This is used in
while loops to generate unique checkbox names for the tabular
layout. */
  $count = 0;
?>
<HTML>
<HEAD><TITLE><?=$title;?></TITLE>
  <META Content="<?=$author;?>" Name="Author">
  <META Content="" Name="Keywords">
  <META Content="" Name="Description">
  <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- JavaScript code-spec to trim a string and change string
case. -->
  <SCRIPT Language="JavaScript" Src="mstrscript.js" Type="text/javascript">
  </SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0"
      TopMargin="0"><BR><BR>
```

```
<!-- JavaScript code-spec unique to the category master form. -->
  <SCRIPT Language="JavaScript">
  /* The chkBlanks() function verifies that the form field of
  category name is not left empty. If the field is empty or
  contains blank spaces, a message indicates to enter a valid
  value. If the field is not empty, the data is submitted. */
     function chkBlanks() {
     /* A variable holding a reference to the form object
     initialised in this page. */
        frm = document.frmMstrCat;
     /* Extracting the contents of the category name field into a
     variable named str. */
        var str = frm.txtCatName.value;
     /* If the category name field is empty or contains blank
     spaces. */
        if(str.trim() == "") {
        /* Displaying a message indicates to enter a valid value.
        */
           alert('Please enter category name.');
        /* Placing the form cursor on the category name field. */
           frm.txtCatName.focus();
        /* Preventing the data in the form field from being
        submitted. */
           return false;
        }
     /* If the category name field holds a value. */
        else {
        /* Calling the function vldtFrmFlds() to validate data
        captured by the form. If the function returns TRUE. */
           if (vldtFrmFlds() == true) {
           /* Allowing the data in the form field to be
           submitted. */
              return true; }
        /* If the function vldtFrmFlds() returns FALSE during
        validation of data captured by the form. */
           else {
           /* Preventing the data in the form field from being
           submitted. */
              return false; }
        }
     }

  /* The function vldtFrmFlds() is called when the form data is
  about to be submitted. This function verifies the presence of
  changes in the form fields during the update mode. If changes
  are not encountered, then a indicating the same is displayed
  and any further processing gets terminated. */
     function vldtFrmFlds() {
```

```
    /* A variable holding a reference to the form object
    initialised in this page. */
       frm = document.frmMstrCat;
    /* If the form is in the update mode. */
       if (frm.hidMode.value == 'U') {
       /* Initialising a variable named mNoChng to hold TRUE.
       This value of this variable determines whether data in
       the form field has been modified or not. */
          var mNoChng = true;
       /* Checking the form objects individually for modified
       information. If any changes have been made, the value of
       the variable named mNoChng is set to hold FALSE. */
          if (frm.hidCatName.value != frm.txtCatName.value) {
             mNoChng = false; }
          if (frm.hidCatRmrk.value != frm.txtCatRmrk.value) {
             mNoChng = false; }
       /* If data in the form fields have remained unchanged. */
          if (mNoChng) {
             alert('Update failed.\n No changes have been made during
                   modification.');
          /* Calling the JavaScript function to reset values in
          the form fields. */
             setNewMode();
          /* Placing the form cursor on the category name field.
          */
             frm.txtCatName.focus();
          /* Preventing the data in the form field from being
          submitted. */
             return false;
          }
       /* If data in the form fields have been changed. */
          else {
          /* Allowing the data in the form field to be
          submitted. */
             return true; }
       }
    /* If the form is not in the update mode. */
       else {
    /* Allowing the data in the form field to be submitted. */
       return true; }
    }

/* The function setNewMode() is called to prepare the form to
except a new set for data. This function is called directly
when the Clear button is clicked. */
   function setNewMode() {
   /* A variable holding a reference to the form object
   initialised in this page. */
      frm = document.frmMstrCat;
```

```
    /* Clearing form fields. */
       frm.txtCatName.value = "";
       frm.txtCatRmrk.value = "";
    /* Clearing hidden variables used for comparison during in
    the update operation. */
       frm.hidCatID.value = "";
       frm.hidCatName.value = "";
       frm.hidCatRmrk.value = "";
    /* Setting the form mode to insert. */
       frm.hidMode.value = "I";
    /* If the form contains a Delete button. */
       if (frm.cmdDelete) {
       /* Enabling the Delete button. */
          frm.cmdDelete.disabled = false;      }
    }

/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
    function setEditMode(id, name, rmrk) {
    /* A variable holding a reference to the form object
    initialised in this page. */
       frm = document.frmMstrCat;
    /* If the form is not in the update mode. */
       if(frm.hidMode.value != 'U') {
       /* Assigning values passed as parameters to hidden
       variables. These are used for comparison during in the
       update operation. */
          frm.hidCatID.value = id;
          frm.hidCatName.value = name;
          frm.hidCatRmrk.value = rmrk;
       /* Populating the form fields. */
          frm.txtCatName.value = name;
          frm.txtCatRmrk.value = rmrk;
       /* Setting the form mode to update. */
          frm.hidMode.value = 'U';
       /* Disabling the Delete button. */
          frm.cmdDelete.disabled = true;
       }
    /* If the form is in the update mode. */
       else {
          alert('Update in progress.\n An entry has already been select for
               modification.');  }
    }
```

```
  /* The function setDelMode() is called to prepare the form to
  remove data from the tabular layout. This function is called
  directly when the Delete button is clicked. */
     function setDelMode() {
     /* Calling the JavaScript function to reset values in the
     form fields. */
        setNewMode();
     /* Setting the form mode for deleting record(s). */
        document.frmMstrCat.hidMode.value = 'D';
     /* Calling the JavaScript function to populate the variable
     holding a list of deleted records. The function
     formDeleteValues() has been defined in the mstrscript.js
     file. */
        formDeleteValues('hidDelRcrdLst', 'frmMstrCat');
     }
  </SCRIPT>
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="<?=$gl_mstr_frst_bar_color;?>" Border="0"
        CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="90%"><TR>
     <TD Align="center" Border="1" BgColor="#C0C0C0"
          Width="10%">Category</TD>
     <TD Align="center" BgColor="#FFFFFF" ColSpan="9"
          Width="90%"><?=$Message;?></TD>
  </TR>
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="ctgry_mstr.php" Method="post" Name="frmMstrCat"
        onSubmit="return chkBlanks();">
  <!-- Declaring a hidden form field used to identify the current
  (i.e. category master) page. -->
     <INPUT Name="hidPage" Type="hidden" Value="ctrgy_mstr">
  <!-- Declaring hidden form fields required for data validation.
  -->
     <INPUT Name="hidCatID" Size="2" Type="hidden" Value="">
     <INPUT Name="hidCatName" Size="2" Type="hidden" Value="">
     <INPUT Name="hidCatRmrk" Size="2" Type="hidden" Value="">
  <!-- Declaring a hidden form field, for determining the form
  mode. It will holds 'I' for Insert, 'U' for Update and 'D' for
  Delete. It will hold 'I' when the page is rendered for the 1st
  time. -->
     <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
  <!-- Declaring a hidden form field used for identifying records
  which have been selected for deletion. -->
     <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
  <TR Height="300" VAlign="top">
     <TD Align="center" Border="1" ColSpan="10"><BR>
     <!-- Form Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
              Name="TlbInner" Width="90%"><TR>
```

```
            <TD Align="left" ColSpan="2"><FONT Size="2"
                Color="#0000CC"><B>Master Setup -
                Category</B></FONT></TD>
        </TR><TR>
            <TD Align="center" ColSpan="2"> </TD>
        </TR><TR>
            <TD Align="right" Width="15%">Category:</TD>
            <TD Align="left"><INPUT maxLength="35" Name="txtCatName"
                onBlur="chngCase('txtCatName', 'frmMstrCat');"
                Type="text"></TD>
        </TR><TR>
            <TD Align="right">Remarks:</TD>
            <TD Align="left"><INPUT maxLength="255" Name="txtCatRmrk"
                onBlur="chngCase('txtCatRmrk', 'frmMstrCat');" Type="text"
                Size="50"></TD>
        </TR><TR>
            <TD> </TD>
            <TD Align="left">
                <INPUT Name="cmdSubmit" Type="submit" Value="Save">
                <IMG Height="1" Src="../Images/pixel.gif" Width="30">
                <INPUT Name="cmdReset" onClick="setNewMode();"
                      Type="button" Value="Clear">
            </TD>
        </TR></TABLE><BR>
    <!-- Form Table Code End. -->

<?
/* Building and executing a SQL query to retrieve records for the
CATEGORY table after sorting the category names in alphabetic
order. */
 */
  $query = "SELECT * FROM category ORDER BY category_name";
  $parsed = ociparse($rcq, $query);
  ociexecute($parsed);

/* Checking if records have been retrieved from the Category
table. */
  if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
?>

    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
              Width="90%">
        <TR BgColor="<?=$gl_mstr_top_bar_color;?>">
            <TD Width="15%"><INPUT Name="cmdDelete"
                onClick="setDelMode();" Type="button" Value="Delete"></TD>
            <TD><FONT Color="#FFFFFF"><B>Category</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Remarks</B></FONT></TD>
        </TR>
```

```
<?
  /* A variable named $rowBgColor hold the hexadecimal colour
  value. This variable is used decided the background colour of
  rows for data in the tabular format. */
     $rowBgcolor = $gl_mstr_scnd_bar_color;

  /* An iterative code performed as long as a single row of
  category data is available. */
     do {
     /* Incrementing the value of the variable used for assigning
     unique names to check boxes display of each row of data. */
        $count = $count + 1;

     /* If the colour code in for the current row matches the
     value for the second row. */
        if ($rowBgcolor == $gl_mstr_scnd_bar_color) {
        /* Colour code for the current row set to match the value
        for the colour code for the first bar. */
           $rowBgcolor = $gl_mstr_frst_bar_color;       }

     /* If the colour code in for the current row matches the
     value for the first row. */
        else {
        /* Colour code for the current row set to match the value
        for the colour code for the second bar. */
           $rowBgcolor = $gl_mstr_scnd_bar_color;      }
?>

        <TR BgColor="<?=$rowBgcolor;?>">
           <TD><INPUT Name="chk<?=($count);?>" Type="checkbox"
                 Value="<?=($line['CATEGORY_ID']);?>"></TD>
           <TD><A
                 HRef="JavaScript:setEditMode('<?=$line['CATEGORY_ID'];?>'
                 , '<?=$line['CATEGORY_NAME'];?>',
                 '<?=$line['CATEGORY_REMARKS'];?>')"><?
                 echo($line['CATEGORY_NAME']); ?></A></TD>
           <TD><? echo($line['CATEGORY_REMARKS']); ?></TD>
        </TR>

<?      } while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM))  ?>
        </TABLE><BR>
<?   }  ?>
     </TD>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends. -->
</BODY>
</HTML>
```

As seen in the above code, the JavaScript functions for the HTML page rendered by **ctgry_mstr.php** is stored in the **mstrscript.js** file. The file will be downloaded by the client's web browser and will contain the following:

```
/*
  Date :- 13/05/05
  Author :- Hansel Colaco
  Filename :- mstrscript.js
  Purpose :- Defines Generic JavaScript functions for trimming
  strings and changing string case.
*/
/* Generating the function used for trimming string values. */
  function strtrim() {
  /* Returns the string passed as a parameter after removing
  blank spaces, if any, at both the beginning and the end of the
  parameter. */
     return this.replace(/^\s+/,'').replace(/\s+$/,'');
  }
/* Generating a global alias name for the function used for
trimming string values. */
  String.prototype.trim = strtrim;

/* The chngCase() function changes the first alphabet to an upper
case character for the value held in the field passed as the
first parameter. The second parameter is a reference to the form
holding the field of the first parameter. This function is called
when a form cursor moves away from a field. */
  function chngCase(pFld, pFrm) {
  /* Using the eval() function to extract the value held in the
  field passed as a parameter. */
     var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
  /* Removing extra spaces for the value extracted above and
  restoring into the same variable. */
     mFldVal = mFldVal.trim();
  /* If the variable containing the extracted value is not empty.
  */
     if (mFldVal != "") {
     /* Changing the string Proper case. */
        mFldVal = mFldVal.charAt(0).toUpperCase() + mFldVal.substr(1,
                  mFldVal.length);
     }
  /* Using the eval() function to display the new value in the
field passed as a parameter. */
     eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
  }
```

```
/* The formDeleteValues() function generates a string of comma
separated identities of the records selected for deletion. If no
records are selected, a message indicates the same. The
parameters for this function are references to a hidden variable
(used for storing the generated string) and the form
respectively. It is called when the DELETE button is clicked. */
  function formDeleteValues(pHid, pFrm) {
  /* Initialising variables for later use. */
     var selValues = "";
     var firstSelBox = "";
  /* Using the eval() function to create a variable to hold a
  reference to the form object. */
     eval("frm = document." + pFrm);
  /* Iterating through every object on the form. */
     for (i=0; i<frm.elements.length; i++) {
     /* If the current object is a Checkbox. */
        if (frm.elements[i].type == "checkbox") {
        /* If the variable firstSelBox is empty. */
           if (firstSelBox == "") {
           /* Assigning the element number, of the current form
           object, to the variable firstSelBox. */
              firstSelBox = i; }
           /* If the current Checkbox has been selected, (i.e.
           checked). */
           if (frm.elements[i].checked == true) {
           /* Assigning the element number, of the current form
           object, to the variable selValues. */
              selValues = selValues + frm.elements[i].value + ",";      }
        }
     }
  /* If the variable selValues does not holds a value. */
     if (selValues.length < 1) {
     /* Displaying a message indicating to select a checkbox. */
        alert('Please choose records you wish to delete.');
     /* Using the eval() function to place the form cursor on the
     first checkbox. */
        eval("document." + pFrm + ".elements[" + firstSelBox + "].focus();");}
  /* If the variable selValues holds a value. */
     else {
     /* Removing the last character (i.e. a comma) for the value
     held in the variable selValues. */
        selValues = selValues.substring(0, selValues.length-1);
     /* Using eval() function to assign a value to the form's
     hidden variable. */
        eval("document." + pFrm + "." + pHid + ".value = '" + selValues + "';");
        frm.submit();
     }
  }
```

Hands On Exercises

1. Based on the Hands on exercise developed in Chapter 12 previously, modify the form to capture information for Book Dimensions. After modifications, the PHP program should be able to store captured data into a table named **DmsnMstr** held under the Oracle database.

- The contents held in the table is displayed in a tabular layout, placed immediately after the data entry form. Refer diagram 14.10
- Besides storing new entries, the program should allow modification of previously stored data
- Similarly, the PHP program should allow (single / multiple) deletion of entries held in the Oracle table
- The page (containing the data entry form and the tabular layout) generated, will display a message indicating the previous operation performed by the PHP program

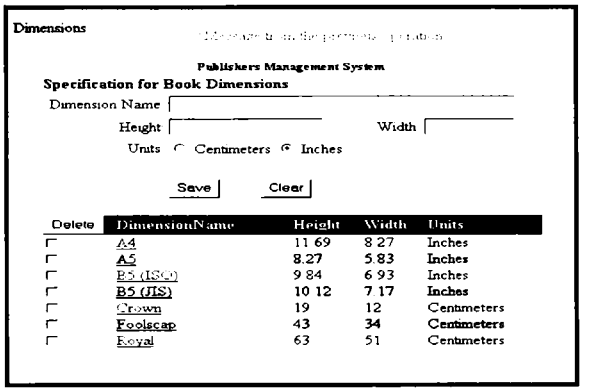


Diagram 14.10: Output for DmsnMstr.php.

SECTION III: WORKING WITH PHP

Regular Expression

Regular expressions are one of those peculiar features that popup in a variety of programming languages but because they seem to be a difficult concept to grasp, many developers push them away into the corner, forgetting that they even exist. Programmers tend to ignore them, which create a large gap in their programming skill. The use of regular expressions creates elegant, powerful PHP programs.

Understanding regular expressions?

Regular expressions started out as a feature of the Unix shell. They were designed to make it easier to find, replace and work with strings and since their invention, they've been in wide use in many different parts of Unix based Operating Systems. They were commonly used in PERL and since then have been implemented into PHP

A regular expression is a specially formatted pattern that can be used to find instances of one string in another. Several programming languages including Visual Basic, PERL, JavaScript and PHP support regular expressions. In other words Regular expressions are basically a pattern definition language used to make complex and flexible searches possible.

Regular expressions in PHP are similar to any modern, text editor's **find** / **replace** tools. This functionality makes editing a text file a whole lot easier. Think of PHP's, regular expressions as an extremely advanced, find-replace tool that saves programmers the pain of having to write custom data validation routines to check e-mail addresses, make sure phone numbers are in the correct format and so on.

Regular Expression Engine

A regular expression **engine** is a piece of software that can process regular expressions, trying to match the pattern to the given string. Usually, the engine is part of a larger application and the user does not have access to the engine directly. Rather, the application will invoke it for whenever needed, making sure the right regular expression is applied to the right file or data.

As usual in the software world, different regular expression engines are not fully compatible with each other. This chapter will focus on the Regular Expression flavor used by PHP 5, for the simple reason that this flavor is the most popular one.

Common Uses Of Regular Expressions

There are a few common uses for regular expressions. Perhaps the most useful is form validation. For example, regular expressions can be used to check that an email address entered into a form uses the correct syntax or making sure that there's a space where required and so on.

They can also be used to complete complex search and replace operations within a given body of text that would not be possible with PHP's standard **str_replace** function.

In a web application, regular expressions would work well for:

- Input validation
- Verifying input format, **e.g.** an email address
- Parsing data from pre-defined variables
- Searching and replacing data in a file or database
- If PHP code spec is being used to create user-defined functions to validate or manipulate portions of a string, then these user-defined functions could be scrapped and regular expressions used instead.
- If custom written functions are being written:
  - To make sure form data contains valid information such as **an @** and **a dot** in an e-mail address

  OR

  - To loop through each character in a string and replace it if it matches a certain criteria, such as if it's upper case, or if it's a space

Regular expressions are really simple to implement. Once a few regular expressions, which are specially formatted strings that are used to tell the PHP regular expression engine, which portion of a string to match, are understood, using regular expressions really becomes routine.

Types Of Regular Expressions In PHP

While the basics of regular expressions are pretty much the same across the board, there are different flavors. PHP supports two flavors of regular expressions. **P**erl **C**ompatible **R**egular **E**xpressions (PCRE) and POSIX Extended Regular Expressions have both been available since PHP 3.

PCRE is generally considered a bit faster, performance wise, than POSIX, but the syntax of POSIX is easier to grasp. The basic concepts of regular expressions **are the same** for both types.

Differences

Following is a set of code spec commonly used to validate e-mail addresses. The first function performs exactly the same as the second, while using regular expression.

Function One:

```
<?php
  function validateEmail($email) {
      $hasAtSymbol = strpos($email, "@");
      $hasDot = strpos($email, ".");

      if($hasAtSymbol && $hasDot)
          return true;

      else
          return false;
  }

  echo validateEmail("sharanams@analystdesigner.com");
?>
```

Function Two:

```
<?php
  function validateEmail($email) {
      return ereg("^[a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+$", $email);
  }

  echo validateEmail("sharanams@analystdesigner.com");
?>
```

The first function looks easy and well structured, but it would be easier to validate an e-mail address using a **one-line** version of the **validateEmail()** function shown in **function two**.

The second function shown above uses regular expressions only and contains one call to the **ereg** function. The **ereg** function always returns **true** or **false**, indicating whether its string argument matched the regular expression or not.

Regular Expression Functions In PHP

Before a string can be matched to a regular expression, the regular expression has to be created. The syntax of regular expressions is a little quirky at first and each phrase in the expression represents some sort of search criteria. A list of some of the most common regular expressions as well as an example of how to use each one has been provided.

PHP has six functions that work with regular expressions. They all take a regular expression as their first argument and are shown below:

ereg()	The most common regular expression function, ereg() allows searching a string for a match with a regular expression /s.
eregi()	Performs exactly the same as ereg(), but is case **insensitive**.
ereg_replace()	Allows searching a string for a regular expression and then replace any occurrence of that string with a new string.
eregi_replace()	Has exactly the same search-replace functionality as ereg_replace(), but is case **insensitive**.
split()	Allows searching a string for a regular expression and returns the matches as an array of strings.
spliti()	Performs exactly the same as spilt(), but is case **insensitive**.

The simplest form of regular expressions is to match a single character or word. This does not require any knowledge of regular expressions or how to create patterns. In fact, it is as simple as the **Find** option used in word processing software. All that needs to be done is to specify the character or word that must be matched.

Symbols Used In Regular Expressions

Using ^ and $

Let's take a look at two special symbols '**^**' and '**$**'. These symbols indicate the start and the end of a string, respectively, as follows:

- "**^Sharanam**": Matches any string that starts with **Sharanam**
- "**Shah$**": Matches a string that ends in the substring **Shah**
- "**^abc$**": Matches a string that starts and ends with **abc.** This could only be the string **abc** itself
- "**Analyst Designer**": a string that has the text **Analyst Designer** in it.

The last example shows that if either of the two characters are not used then it indicates that the pattern may occur anywhere inside the string. The string is not bound to any of the edges.

Using *, + and ?

There are also the symbols '*****', '**+**', and '**?**', which denote the number of times a character or a sequence of characters may occur. The '*' means **zero or more**, '+' means **one or more** and '**?**' means **zero or one**.

They can be used as:

- "**sh***": Matches a string that has a character **s** followed by zero or more **h**. The possibilities are "**s**", "**sh**", "**shhh**" and so on
- "**sh+**": Matches a string that has a character **s** followed by at least one or more **h**. The possibilities are "**sh**", "**shhh**" and so on
- "**sh?**": Matches a string that has a character **s** but may or may not be followed by a single character **h**. The possibilities are "**s**", "**sh**"
- "**s?h+$**": Matches a string that has may have a character **s** but compulsorily ends with at least one or more **h**. The possibilities are "**h**", "**sh**", "**shhh**", "**shah**" and so on

Using Bounds {}

Bounds can also be used, which come inside braces and indicate ranges in the terms of occurrences. They are used as:

- "**sh{2}**": Matches a string that has a character **s** followed by exactly two **h**. The possibility is "**shh**"
- "**sh{2,}**": Matches a string that has a character **s** followed by at least two **h**. The possibilities are "**shh**", "**shhhh**" and so on
- "**sh{3,5}**": Matches a string that has a character **s** followed by at least three but not more than five **h**. The possibilities are "**shhh**", "**shhhh**", "**shhhhh**"

REMINDER

The first number/parameter in a range should always be specified. This means **{0,2}** or **{,2}** are valid and not **{,2}**. Also, notice that the symbols used earlier such as '*', '+' and '?' have the same effect as using the bounds **{0,}**, **{1,}**, and **{0,1}** respectively.

Using quantifiers

To quantify a sequence of characters, put them inside round brackets as:

- "**s(ha)***": Matches a string that has a character **s** followed by zero or more copies of the sequence "**ha**"
- "**s(ha){1,5}**": Matches a string that has a character **s** followed by at least one but not more than five copies of "**ha**"

Using | The OR operator

The '**|**' symbol works similar to the OR operator and can be used as:

- "**Sharanam | Mr.Shah**": Matches a string that holds either "**Sharanam**" or "**Mr.Shah**"

- "**(Vaishali|Sharanam)Shah**": Matches a string that holds either "**VaishaliShah**" or "**SharanamShah**"
- "**(s|h)*a**": Matches a string that has a sequence of alternating characters i.e. **s** and **h** but ending with a character **a**. The possibilities are "**sa**", "**ssa**", "**sssa**" and so on or "**ha**", "**hha**", "**hhha**" and so on

Using . The Period

A period '.' stands for any single character and can be used as:

- "**s.[0-9]**": Matches a string that has a character **s** followed by any one character further followed by a digit. The possibilities are "sh2", "sa1" and so on
- "**^.{8}$**": Matches a string with exactly 8 characters. The possibilities are "Sharanam", "Vaishali" and so on

Using [ ] Bracket Expressions

Bracket expressions specify which characters are allowed in a single position of a string and can be used as:

- "**[sh]**": Matches a string that has either the character **s** or **h**. This is same as "**s|h**"
- "**[a-d]**": Matches a string that holds lowercase characters '**a**' through '**d**'. This is equal to "**a|b|c|d**" or "**[abcd]**")
- "**^[a-zA-Z]**": Matches a string that begins with a letter
- "**[0-9]%**": Matches a string that has a single digit before a percent sign
- "**,[a-zA-Z0-9]$**": Matches a string that ends in a comma followed by an alphanumeric character
- "**%[^a-zA-Z]%**": Matches a string with a character that is not an alphabet between two percent signs

Using Regular Expression Functions

REMINDER

With **ereg()**, matching is **case sensitive**. For **case insensitive** matching, use **eregi()**.

In this example the signature (**i.e.** codespec) of the function ereg(), will look for the word Analyst in the supplied string. If it locates the word, **true** is returned. Otherwise, false is returned. Obviously, in this example the result would be false.

Now consider the following:

```
eregi("Analyst", "Sharanam is an analyst Designer.");
```

In this example the signature (**i.e.** codespec) of the function **eregi()**, will look for the word Analyst in the supplied string. In this example the result would be true, as the match performed here is case insensitive.

Now, let's introduce a simple pattern using the (.) meta-character. A meta-character is a character that PHP expands to mean something else. The (.) meta-character, expands to represent any character **except** a new line. Here the original example is slightly modified, to show the use of this special character.

```
ereg("Mu...i", "Sharanam stays in Mumbai");
```

The regular expression of **Mu...i** will match any occurrence of the letter **Mu** that is followed by **any three** other characters and then by the letter **i**o. This will match the word Mumbai in the string, so the expression will return **true**.

The ^ and $ symbols can be specified in the ereg function. **^** denotes that a match should be found at the **beginning** of a string, while the **$** modifier denotes that the match should be found at the end of a string.

```
ereg("^Mumbai", "Sharanam stays in Mumbai");
```

The preceding line of code returns **false** because Mumbai is **not** at the **beginning** of the string in which pattern matching was attempted.

```
ereg("Mumbai$", "Sharanam stays in Mumbai");
```

The preceding line of code returns **true** because Mumbai is at the **end** of the string in which pattern matching was attempted.

Both modifiers can be used together. To find out if a string contained **only the word** Mumbai use the following**:**

```
ereg("^Mumbai$", "Sharanam stays in Mumbai");
```

The preceding line of code returns **false** because Mumbai is not the only word in the string in which pattern matching was attempted.

There are also meta-characters that specify the number of times a particular pattern can occur in a match. One is the **?** character. When this meta-character is used, the preceding pattern or character will be matched **zero** or **one** times.

```
ereg("MumbaiX?", "Sharanam stays in Mumbai");
```

This expression will evaluate to **true** because the letter X **can occur once, or not at all.**

REMINDER

Since the characters preceding the meta-character (**X**) are not placed in brackets, the **meta-character applies only to the letter X**.

The next meta-character is the (*) character. It will match the preceding pattern or character, zero or more times. This character is used as a **wildcard** character in many different applications. The difference in regular expressions is that **it must follow the pattern** to be matched, rather than just replace it.

```
ereg("M*bai", "Sharanam stays in Mumbai");
```

This expression matches the letters **M**, followed by zero or more letters, followed by the letters **bai**.

The **+** meta-character can also be used and provides **almost the same functionality** as the * character, except that the pattern **must occur at least once**.

```
ereg("Mu+bai", "Sharanam stays in Mumbai");
```

That would match the letters **Mu**, followed by one or more letters, then followed by the letters **bai**.

The previous modifiers are useful when matching varying occurrences of a pattern, what if an exact amount of occurrences must be matched? Regular expressions provide a simple means of doing so using **bounds**.

```
ereg("Mum{1}bai", "Sharanam stays in Mumbai");
```

This would match the word Mumbai. The **{1}** signifies that the preceding character or pattern (in this case **m**) should be matched exactly once. This would match the letters **Mu**, followed by exactly one occurrence of the letter **m**, then followed by the letters **bai**.

Now, let's take a look at specifying a range for the number of matches.

```
ereg("Mum{1,3}bai", "Sharanam stays in Mumbai");
```

This would match the word Mu**m**bai, Mu**mm**bai, or Mu**mmm**bai. The bound **{1,3}** allows matches of at least one but not more than three **m**.

Up to this point single characters have been dealt with. Take a look at grouping with parentheses, in order to use the different meta-characters on more than one character at a time.

```
ereg("Mu(mm)+bai", "Sharanam stays in Mummmmbai");
```

In this example, the regular expression will match the letters **Mu**, followed by one or more occurrences of **mm**, then followed by the letters **bai**. In this case, the expression would return **true**. It would have also returned true if the string being matched against had the word **Mummbai** in it or the word **Mummmmmmbai**.

Using the (|) character. This operator permits an **either or** situation.

```
ereg("(Sharanam|Mr. Shah)", "Mr. Shah stays in Mumbai");
```

This example will match either **Sharanam** or **Mr. Shah**. Because the string being matched contains the word **Mr. Shah**, this expression will return **true**.

Using **[ ]** (square brackets), a range of characters to be matched can be expressed. This can be especially useful when matching numbers (**i.e.** digits) or a specific range of letters. Multiple ranges can be placed within the square brackets as follows:

```
ereg("Mumba[0-9g-j]", "Sharanam stays in Mumbai");
```

The preceding example will match the word **Mumba**, followed by a **digit** or a **letter** in the range of **g** through **j**. In this example, the expression would evaluate as **true**, because the letter **i** falls within this range.

To match the space character in a search string, use the predefined POSIX class, [[:space:]]. The square brackets indicate a related set of sequential characters and ":space:" is the actual class to match (which, in this case, is any white space character).

White spaces include the tab character, the new line character and the space character. Alternatively, use **one space** character (" ") if the search string must contain **just** one space and **not** a tab or new line character.

In most circumstances use **:space:** because it signifies a programmers intention a bit better than a single space character, which can easily be overlooked. There are several POSIX standard predefined classes that can be used as part of a regular expression, including **[:alnum:]**, **[:digit:]**, **[:lower:]** and so on.

Match a single space character like this:

```
<?php echo ereg("Sharanam[[:space:]]Shah", "Sharanam Shah"); ?>
```

Match either **no** spaces or **one** space by using the **?** character after the expression, as follows:

```
<?php echo ereg("Sharanam[[:space:]]?Shah", "SharanamShah"); ?>
```

The following expression uses escape characters and involves placing a backslash **before any special symbols**. To include the or symbol (|) in the search:

```
<?php echo ereg("^[a-zA-z]+\|[a-zA-z]+$", "Sharanam|Mr.Shah"); ?>
```

There are only a handful of symbols that require escape characters placed before them, they are:

```
^, $, (, ), ., [, |, *, ?, +, \ and {
```

The following regular expression validates a standard URL (with **no** port number).

```
ereg("^(http|ftp)://(www\.)?.+\.(com|net|org)$", "www.rediff.com");
```

^(http|ftp) indicates matching the protocol part of the URL. This makes sure that only **http** or **ftp** can be used. The **^** character specifies the beginning of the string and by enclosing http and ftp in brackets and separating them with the **or** character (**i.e.** the **|** symbol), the PHP regular expression engine is told that either the characters http or ftp must be at the beginning of the string.

(www\.)?.+\.(com|net|org)$ indicates that a domain name usually consists of www.somewebsite.com, but can optionally be specified without the **www** part. This example only allows **.com**, **.net** and **.org** domain names to be considered valid.

Example:
The above illustration can be built as shown below:

```
<?php
  function isValidDomain($domainName) {
      return ereg("^(http|ftp)://(www\.)?.+\.(com|net|org)$", $domainName);
  }
  echo isValidDomain("http://www.rediffmail.com");
  echo isValidDomain("ftp://downloads.com");
  echo isValidDomain("ftp://www.google.co.in");
  echo isValidDomain("www.phptraining.com");
?>
```

In the above example, the first two calls to **isValidDomain()** will return **true**, while the last two calls will return **false**.

Similarly, the **eregi()** can be used to perform **case insensitive** matching as required while performing email address validation:

```
<?php
  function CheckValidEmail($cEmail) {
      if( eregi("^[a-z0-9\._-]+" . "@" . "([a-z0-9][a-z0-9-]*[a-z0-9]\.)+"
          . "([a-z]+\.)?". "([a-z]+)$", $cEmail) ) {
          return TRUE;   }
      return FALSE;
  }
?>
```

Examining the regular expression portion line by line:

^[a-z0-9\._-]+	This portion of the expression will match any letter, any digit, a period, an underscore or a hyphen. The match must occur at the beginning of the string and it will match one or more of those characters.
@	This next piece of the regular expression will match a single occurrence of an @ symbol. Because this is appended to the previous portion of the regular expression, it would occur after the first portion was matched.
([a-z0-9][a-z0-9-]*[a-z0-9]\.)+	This portion will match a single letter or digit. Then zero or more letters or digits are matched, followed by a single letter or digit again. At the end of this piece, a single period must be matched.
([a-z]+\.)? .	This part of the regular expression will match one or more occurrences of any letter, followed by a period. This entire part of the expression can occur once or not at all as defined by the ? character.
([a-z]+)$	This final piece of the regular expression will match any letter and that letter must occur at least once.

REMINDER

With **ereg_replace()**, matching is **case sensitive**. For **case insensitive** matching, use **eregi_replace()**.

```
ereg_replace("Analyst Designer", "Project Leader", "Sharanam is an analyst
                Designer.");
```

In this example the signature (**i.e.** codespec) of the function ereg_replace(), will look for the word Analyst Designer in the supplied string and replace it with Project Leader. If it locates the word, **the modified string** is returned. Otherwise, **original string** is returned. Obviously, in this example the result would be the original string as ereg_replace() is case sensitive.

Now consider the following:

```
eregi_replace("Analyst Designer", "Project Leader",  "Sharanam is an analyst
                Designer.");
```

In this example the signature (**i.e.** codespec) of the function eregi_replace(), will look for the word Analyst Designer in the supplied string and replace it with Project Leader. In this example the result would be the modified string, as the match performed here is case insensitive.

```
ereg_replace("Mumbai$", "Delhi", "Sharanam stays in Mumbai");
```

The preceding line of code returns **the modified string** because Mumbai is at the **end** of the string in which pattern matching was attempted.

Using the (|) character. This operator permits an **either or** situation.

```
ereg_replace("(Mumbai|Delhi)$", "Chennai", "Sharanam stays in Mumbai");
```

This example will match either **Mumbai** or **Delhi** at the end of the string. Because the string being matched ends with the word **Mumbai**, the string returned will be "**Sharanam stays in Chennai**".

The **ereg_replace()** function can be used to replace URLs with HTML links. Consider the contents of **cnvtURL.php**, as shown below:

```
<?php
  $url = "http://www.yahoo.com";
  $link = ereg_replace("[[:alpha:]]+://[^<>[:space:]]+[[:alnum:]/]", "<A
                        HRef=\"\\0\">\\0</a>", $url);
?>
<HTML>
<HEAD><TITLE>Link To A URL</TITLE>
<BODY>The converted URL is:<BR><?=$link;?></BODY>
</HTML>
```

The converted URL is:
http://www.yahoo.com

Diagram 15.1: Output for cnvtURL.php.

REMINDER

If an integer value is used as the replacement parameter, the expected result may not be achieved. This is because **ereg_replace()** will interpret the number as the ordinal value of a character and apply that.

Example: replaceNumber.php

```
<?php
  $num = 12;
  $str = "There are twelve months in a
          year.";
  $str = ereg_replace("twelve",
                      $num, $str);
  echo $str."\n";

  $num = '12';
  $str = "There are twelve months in a
year.";
  $str = ereg_replace("twelve", $num, $str);
  echo $str."\n";
?>
```

```
[root@rose Chap15_Cds]# php replaceNumber.php
There are
          months in a year.
There are 12 months in a year.
[root@rose Chap15_Cds]#
```

Diagram 15.2: Output for replaceNumber.php.

Output: (Refer diagram 15.2)

In the above example, the first block of code fails in the replace operation because the variable **$num** holds a numeric value. The second block of code is able to perform the replacement as the variable **$num** holds a string.

The **split()** can be used to parse a date which may be delimited with slashes, dots or hyphens:

```
list($month, $day, $year) = split('[/.-]', $date);
```

The variable $date may hold the date in the formats such as mm/dd/yyyy, mm.dd.yyyy or mm-dd-yyyy.

The above code spec, after splitting the value held by the variable $date in three parts, will transfer them into variables $month, $day and $year.

REMINDER

With **split()**, matching is **case sensitive**. For **case insensitive** matching, use **spliti()**.

Example:
Now consider the following example, (stored in **splitiTest.php**) which demonstrates the technique that can be used in **spliti()**:

```
<?php
  $str = "http://www.yourdob.com?txtName=Sharanam&TxtSurname=Shah
         &TXTDOB=03/01/1981";
  $columns = spliti (".txt", $str, 5);
  echo "The parameters and their corresponding values passed are:";
  print_r($columns);
?>
```

```
[root@rose Chap15_Cds]# php splitiTest.php
The parameters and their corresponding values
passed are:Array
(
    [0] => http://www.yourdob.com
    [1] => Name=Sharanam
    [2] => Surname=Shah
    [3] => DOB=03/01/1981
)
[root@rose Chap15_Cds]#
```

Diagram 15.3: Output for splitiTest.php.

Output: (Refer diagram 15.3)

In the above example, the **spliti()** function is used to divide string contained in the variable **$str**, into five segments. The string is split for every occurrence of a character followed by 'txt'. The case used for 'txt' is irrelevant as spliti() is case insensitive.

In the above example, if the third parameter for **spliti()** was not specified, the value contained in **$str** would have been segmented into four parts (as the combination of a character followed by 'txt' appears three times).

Example:
Consider the following illustration, wherein a data entry form, within a PHP generated page, captures data such as name, address and date of birth in three individual textboxes. The captured values are submitted to a PHP page, which uses regular expressions to validate and format the data before rendering it on the client's browser.

Create the file **RegForm.php** with the following contents:

```
<?
/*
  Date :- 14/06/05
  Author :- Hansel Colaco.
  Filename :- RegForm.php
  Purpose :- Formats Visitor's data like name, birth date and
             address using regular expressions.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco';
?>
<HTML>
<HEAD><TITLE>Capturing Data From Visitor</TITLE></HEAD>
<BODY>
  <SCRIPT Language="JavaScript">
  /* Generating the function used for trimming string values. */
     function strtrim() {
     /* Returns the string passed as a parameter after removing
     blank spaces, if any, at both the beginning and the end of
     the parameter. */
        return this.replace(/^\s+/, '').replace(/\s+$/, '');
     }

  /* Generating a global alias name for the function used for
  trimming string values. */
     String.prototype.trim = strtrim;

  /* The function rmvSpace() trim extra spaces at either ends of
  the value entered in the form field pass as parameter. The
  function accepts two parameters; first is the form name and the
  second is the field name. */
     function rmvSpace(pFld, pFrm) {
     /* Using the eval() function to extract the value held in
     the field passed as a parameter. */
        var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
     /* Removing extra spaces for the value extracted above and
     restoring into the same variable. */
        mFldVal = mFldVal.trim();
     /* Using the eval() function to display the new value in the
     field passed as a parameter. */
        eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
     }
```

```
  /* The function clrFrmFlds() clear the contents of the form
  fields when the Clear button is clicked. */
     function clrFrmFlds() {
     /* A variable holding a reference to the form object
     initialised in this page. */
        frm = document.frmVisitorInfo;
     /* Clearing form fields. */
        frm.txtName.value = "";
        frm.txtDOB.value = "";
        frm.txtAddr.value = "";
     }
  </SCRIPT>

  <TABLE Border="0" BgColor="#FFFFFF" CellPadding="0" CellSpacing="0"
         Width="600"><TR>
     <TD Colspan="2" Height="15" Width="400"></TD>
     <TD Align="center" RowSpan="2" VAlign="top"><IMG Height="50"
          Src="../images/Sctonly2.gif" Width="200"></TD>
  </TR><TR>
     <TD BgColor="#FFFFFF" Width="20"> </TD>
     <TD BgColor="#FFFFFF" Width="380"><H3>Visitor's Information</H3></TD>
  </TR><TR>
     <TD ColSpan="3">
        <TABLE Border="0" CellPadding="0" CellSpacing="0"
               Width="100%"><TR>
           <TD Width="14"> </TD>
           <TD><?=$msg;?>
<!-- Initial code corresponding to the template ends. -->

<!-- Designing the layout of the form, which will capture the
visitor's name, birth date and address. The data captured will be
submitted to the RegForm.php page. A JavaScript based validation
is performed on the captured information before it is submitted.
-->
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Height="225"
Width="100%"><TR>
  <TD Width="20"></TD>
  <TD VAlign="top" Width="560">
<FORM Action="UseReg.php" Method="post" Name="frmVisitorInfo">
  <CENTER>Please enter the information to get access to the resources
          available<BR></CENTER>

  <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
         Width="100%"><TR>
     <TD ColSpan="3"><B>Personal Information</B></TD>
  </TR><TR>
     <TD Align="right" >Name: </TD>
```

```
      <TD Align="left" ColSpan="2"><INPUT Size="70" Name="txtName"
            onBlur="rmvSpace('txtName', 'frmVisitorInfo');" Type="text"
            Value="<?=$txtName;?>"></TD>
  </TR><TR>
      <TD Align="center" ColSpan="3"><FONT Color="blue">(Format:
            <B>Surname</B>[:SPACE:]<B>Name</B>[:SPACE:]<B>Second/Middl
            e Name</B>)</FONT></TD>
  </TR><TR>
      <TD Align="right">Birthdate: </TD>
      <TD Align="left" ColSpan="2"><INPUT Size="20" Name="txtDOB"
            onBlur="rmvSpace('txtDOB', 'frmVisitorInfo');" Type="text"
            Value="<?=$txtDOB;?>"></TD>
  </TR><TR>
      <TD Align="center" ColSpan="3"><FONT Color="blue">(Format:
            <B>dd/mm/yyyy</B> or <B>dd.mm.yyyy</B> or <B>dd-mm-
            yyyy</B>)</FONT></TD>
  </TR><TR>
      <TD Align="right">Address: </TD>
      <TD Align="left" ColSpan="2"><INPUT Size="70" Name="txtAddr"
            onBlur="rmvSpace('txtAddr', 'frmVisitorInfo');" Type="text"
            Value="<?=$txtAddr;?>"></TD>
  </TR><TR>
      <TD Align="center" ColSpan="3"><FONT Color="blue">(Use commas for
            multi-lines addresses)</FONT></TD>
  </TR><TR>
      <TD ColSpan="3"><IMG Height="10" Src="../images/pixel.gif"
            Width="1"></TD>
  </TR><TR>
      <TD Align="right"><INPUT Name="cmdSubmit" Type="submit"
            Value="Submit"></TD>
      <TD><IMG Height="10" Src="../images/pixel.gif" Width="1"></TD>
      <TD Align="left"><INPUT Name="cmdClear" onClick="clrFrmFlds();"
            Type="Button" Value="Clear"></TD>
  </TR></TABLE>
</FORM>
  </TD><TD VAlign="top" Width="20"><BR></TD>
</TR></TABLE>

<!-- Final code corresponding to the template begins. -->
            </TD>
            <TD Width="14">  </TD>
          </TR></TABLE>
      </TD>
  </TR></TABLE>
</BODY>
</HTML>
```

Save the changes made to **RegForm.php** and run it via a web browser. The resulting page rendered appears as shown in diagram 15.4.

Enter the information into the form fields and click Submit to pass the data to **UseReg.php**.

On receiving the data, **UseReg.php** processes the code snippet mentioned below:

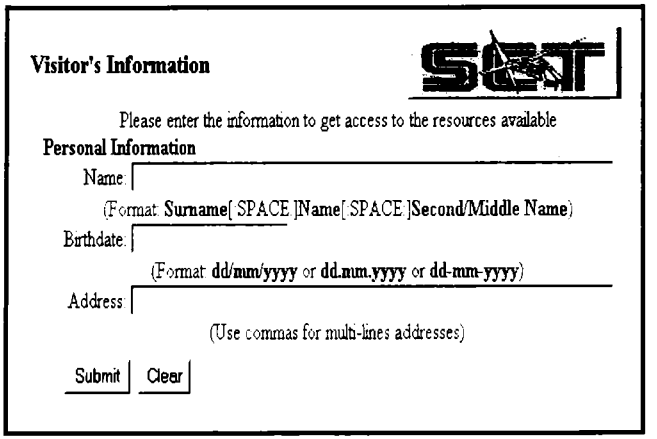


Diagram 15.4: Default page for RegForm.php.

```
<?
/*
  Date :- 14/06/05
  Author :- Hansel Colaco.
  Filename :- UseReg.php
  Purpose :- Validates and displays Visitor's data like name,
             birth date and address using regular expressions.
*/

/* A variable holding the developers name. */
  $author = 'Hansel Colaco';
/* Storing the system date in the format 1st Jan, 2000. */
  $sysdt = date("jS M, Y");
/* Storing the system time in the format 00:00. */
  $systm = date("H:i");

/* Checking if the variable for visitor's name is empty, (i.e. a
blank name field was submitted). */
  if(empty($txtName)) {
  /* Returning to the data entry form if the visitor name is
  empty. */
     $msg = '<FONT Color="red"><B>Field for visitor\'s Name is
              empty!</B></FONT>';
     include "RegForm.php";
     exit;
  }

/* When the visitor's name is submitted. */
  else {
  /* Using regular expressions with the ereg() function, to
  identify an occurrence of double spaces in the visitor's name.
  */
     if(ereg("([[:space:]]){2}", $txtName)) {
```

```
    /* Returning to the data entry form, if the visitor's name
    contains double spaces. */
        $msg = '<FONT Color="red"><B>Double spaces found in visitor\'s
                Name!</B></FONT>';
        include "RegForm.php";
        exit;
    }
  }

/* Checking if the variable for visitor's birth date is empty,
(i.e. a blank date of birth field was submitted). */
  if(empty($txtDOB)) {
  /* Returning to the data entry form, if the visitor's birth
  date is empty. */
      $msg = '<FONT Color="red"><B>Field for visitor\'s birth date is
              empty!</B></FONT>';
      include "RegForm.php";
      exit;
  }
/* When the visitor's birth date is submitted. */
  else {
  /* Returning to the data entry form, if the format for the
  visitor's birth data is not as specified in the data entry
  form. */
      if(!ereg("^([0-9]){2}[/.-]([0-9]){2}[/.-]([0-9]){4}", $txtDOB)) {
          $msg = '<FONT Color="red"><B>Invalid format for visitor\'s birth
                  date!</B></FONT>';
          include "RegForm.php";
          exit;   }
  }
/* Checking if the variable for visitor's postal address is
empty, (i.e. a blank address field was submitted). */
  if(empty($txtAddr)) {
  /* Returning to the data entry form, if the visitor's postal
  address is empty. */
      $msg = '<FONT Color="red"><B>Field for visitor\'s postal address is
              empty!</B></FONT>';
      include "RegForm.php";
      exit;
  }
?>

<HTML>
<HEAD><TITLE>Formatting Using Regular Expression</TITLE></HEAD>
<BODY><H4>Details Submitted As On:</H4>
  <TABLE Border="1" CellSpacing="1" Width="200"><TR>
      <TD Align="right" Width="75">Server Date :</TD>
      <TD Align="left" Width="100"><?="$sysdt";?></TD>
  </TR><TR>
```

```
      <TD Align="right" Width="75">Server Time :</TD>
      <TD Align="left" Width="100"><?="$systm";?></TD>
  </TR></TABLE><HR>
<?
/* Using the split() function to extract visitor's last name,
first name and second name into variables $lname, $fname and
$mname respectively. */
  list($lname, $fname, $mname) = split(" ", $txtName, 3);
?>

  <H4>Visitor's Name:</H4>
  <TABLE Border="1" CellSpacing="1" Width="500"><TR>
      <TD Align="left" Width="150"><?=$lname;?></TD>
      <TD Align="left" Width="150"><?=$fname;?></TD>
      <TD Align="left" Width="200"><?=$mname;?></TD>
  </TR><TR>
      <TD Align="center" Width="150">Surname</TD>
      <TD Align="center" Width="150">First Name</TD>
      <TD Align="center" Width="150">Second/Middle Name</TD>
  </TR></TABLE><HR>
<?
/* Using the ereg_replace() function to separates the day, month
and year value with spaces. */
  $txtDOB = ereg_replace('[/.-]', ' ', $txtDOB);
?>
  <H4>Date Of Birth:</H4>

  <TABLE Border="0" CellSpacing="1" Width="300"><TR>
      <TD Align="left" Width="150"><?=$txtDOB;?></TD>
  </TR></TABLE><HR>
  <H4>Postal Address:</H4>

  <TABLE Border="0" CellSpacing="1" Width="400">
<?
/* Using the split() function to extract the segments held in the
multi-line address. */
  $PostAddr = split(",", $txtAddr);
/* Using an iteration to display the visitor's address. */
  foreach($PostAddr as $fld) {
?>
      <TR><TD Align="left" Width="150"><?=$fld;?></TD></TR>
  </TABLE><HR>
<?   }  ?>
</BODY>
</HTML>
```

Save the changes made to **UseReg.php** and execute the file by passing a call through **RegForm.php**.

Following validations are performed by the **UseReg.php** file.

- Name cannot be left empty
- No double spacing in the name captured
- Birth date cannot be left empty
- Birth date should be in a specific format
- Address cannot be left empty

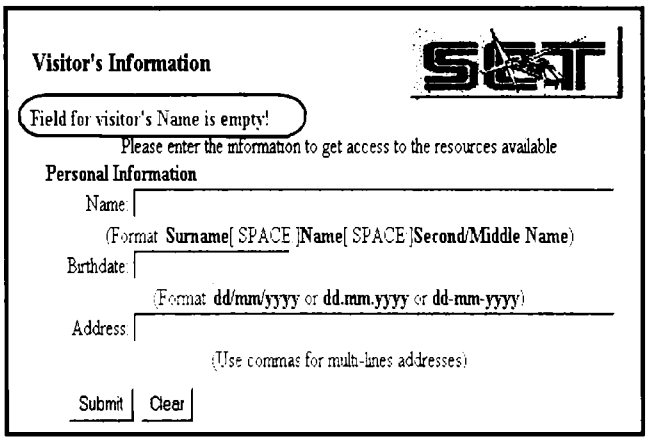


Diagram 15.5.1: Output generated by UseReg.php when an empty visitor's name field is submitted.

Verify the validations as:

- Name cannot be left empty

Submit the form without passing any data by pressing Submit button. The **UseReg.php** file checks the contents passed by the visitor's name field. As it is empty, an error message is generated and stored into the variable **$msg**. The entire contents of the **RegForm.php** file is included in the output before terminating the processing for **UseReg.php** file. This means the control is passed back to the **RegForm.php** file with a message indicating the error. Refer diagram 15.5.1.

As seen in diagram 15.5.1, the error message is displayed after the form title.

The above validation is done using the following code spec:

```
/* Checking if the variable for visitor's name is empty, (i.e. a
blank name field was submitted). */
  if(empty($txtName)) {
  /* Returning to the data entry form if the visitor name is
  empty. */
      $msg = '<FONT Color="red"><B>Field for visitor\'s Name is
              empty!</B></FONT>';
      include "RegForm.php";
      exit;   }
```

- No double spacing in the name captured

Enter the visitor's name. To test the double spacing validation, insert double spaces in between the words entered. Click Submit to submit information held in the form fields. This passes the control back to the **UseReg.php** file.

To validate the name for double spaces, the **ereg()** function is used which will scan the contents of visitor's name for the occurrence of double spaces:

```
ereg("([[:space:]]){2}", $txtName)
```

Since double space are encountered, the **ereg()** function will return true and thus generate an error message and passes the control back to the **RegForm.php** file. Refer diagram 15.5.2

The above validation is done using the following code spec:

```
/*    Using    regular
expressions    with    the
ereg()    function,    to
identify    an    occurrence
of double spaces in the
visitor's name. */
   if(ereg("([[:space:]]){2}",
         $txtName)) {
   /* Returning to the
   data entry form, if
   the visitor's name
   contains double
   spaces. */
      $msg = '<FONT Color="red"><B>Double spaces found in visitor\'s
            Name!</B></FONT>';
      include "RegForm.php";
      exit;
   }
```

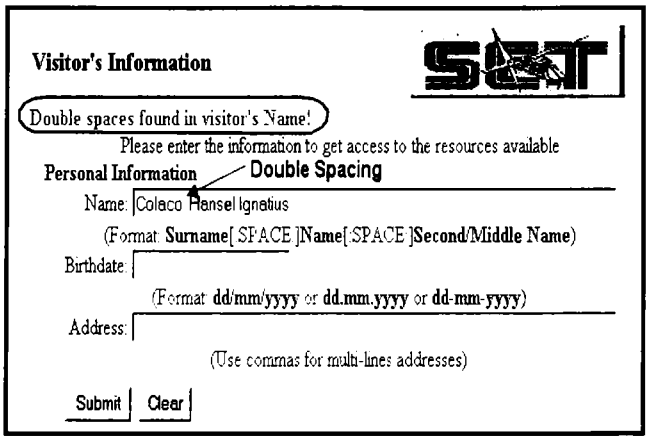


Diagram 15.5.2: Output generated by UseReg.php when double spaces are passed in the name field.

The above regular expression uses the [[:space:]] to identify space. Since double spacing is to be identified bound {2} is used to search two continuous spaces.

- Birthdate cannot be left empty

Now, Submit the form by passing a valid name but without a birth date. Since the date is empty, an error message is generated and stored into the variable **$msg**. The entire contents of the **RegForm.php** file is included in the output before terminating the processing for **UseReg.php** file. This means the control is passed back to the **RegForm.php** file with a message indicating the error. Refer diagram 15.5.3.

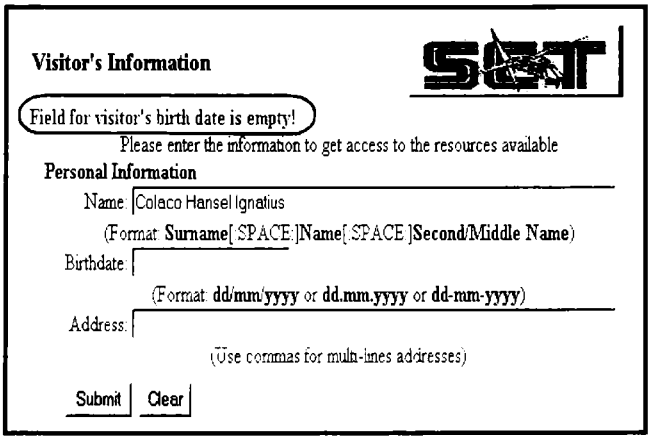


Diagram 15.5.3: Output generated by UseReg.php when an empty birth date field is submitted.

As seen in diagram 15.5.3, the error message is displayed after the form title.

The above validation is done using the following code spec:

```
/* Checking if the variable for visitor's birth date is empty,
(i.e. a blank date of birth field was submitted). */
  if(empty($txtDOB)) {
  /* Returning to the data entry form, if the visitor's birth
  date is empty. */
      $msg = '<FONT Color="red"><B>Field for visitor\'s birth date is empty!</B>
              </FONT>';
      include "RegForm.php";
      exit;  }
```

- Birthdate should be in a specific format

To test the date format validation Submit the form using valid name and a date in an invalid format such as **01 1 1981**. Now since the date format is invalid, an error message is generated and stored into the variable **$msg**. The entire contents of the **RegForm.php** file is included in the output before terminating the processing for **UseReg.php** file. This means the control is passed back to the **RegForm.php** file with a message indicating the error. Refer diagram 15.5.4.

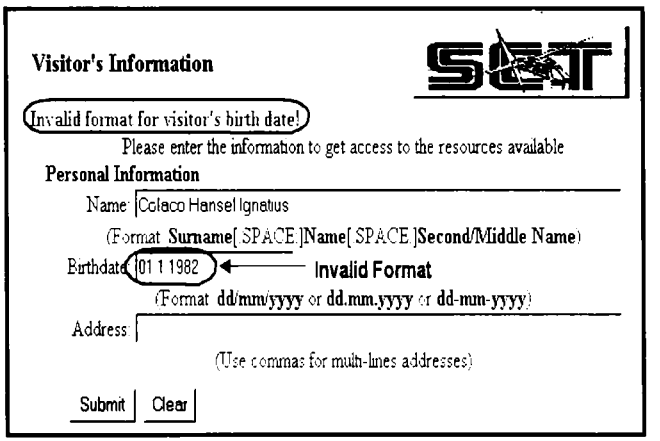


Diagram 15.5.4: Output generated by UseReg.php when invalid format for birth date is passed.

As seen in diagram 15.5.4, the error message is displayed after the form title.

The above validation is done using the following code spec:

```
/* Returning to the data entry form, if the format for the
visitor's birth data is not as specified in the data entry
form. */
    if(!ereg("^([0-9]){2}[/.-]([0-9]){2}[/.-]([0-9]){4}", $txtDOB)) {
        $msg = '<FONT Color="red"><B>Invalid format for visitor\'s birth
                date!</B></FONT>';
        include "RegForm.php";
        exit;  }
```

In the above regular expression ^ indicates that the date should begin with a two digit number (DD - Date) indicated by ([0-9]){2}). Here [0-9] means digits and since two digits are desired a bound {2} is specified. Thereafter the date should have either a slash (/) or a period (.) or a hyphen (-). These are placed in box brackets [] which will accept either of them. Further the date should have two-digit number (MM - Month). This is again done using ([0-9]){2} expression as explained earlier. Then followed by either a slash (/) or a period (.) or a hyphen (-) done using box brackets []. Finally a two or a four-digit number (YY or YYYY) is required and it is expressed as ([0-9]){4} indicating a four-digit number will appear here.

- Address cannot be left empty

To test the final validation, Submit the form with a valid name and a birth date, but leave the address field empty. Since the postal address field is blank, an error message is generated and stored into the variable **$msg**. The entire contents of the **RegForm.php** file is included in the output before terminating the processing for **UseReg.php** file. This means the control is passed back to the **RegForm.php** file with a message indicating the error. Refer diagram 15.5.5.

As seen in diagram 15.5.5, the error message is displayed after the form title.

The above validation is done using the following code spec:

```
/*   Checking   if   the
visitor's  postal  address
is empty. */
  if(empty($txtAddr)) {
  /*   Returning   to   the
  data  entry  form,  if  the
  visitor's          postal
  address is empty. */
      $msg = '<FONT Color="red"><B>Field for visitor\'s postal address is
              empty!</B></FONT>';
      include "RegForm.php";
      exit;
  }
```

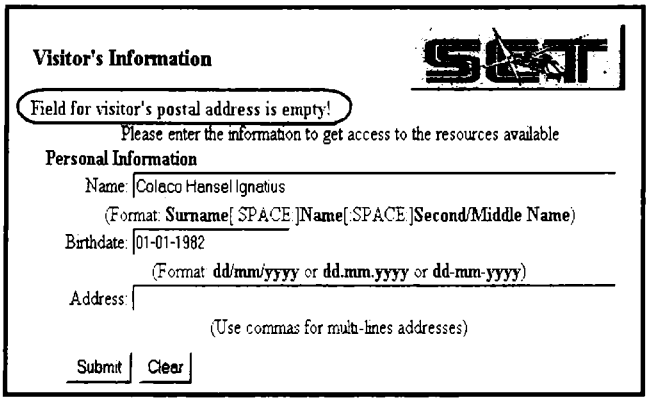


Diagram 15.5.5: Output generated by UseReg.php when an empty postal address field is submitted.

To view the output generated completely by **UseReg.php**, ensure that valid data is submitted via the form in **RegForm.php**. Refer diagram 15.6.1.

When valid data is received by **UserReg.php**, Regular expression functions are used to decide the layout for display, which includedes:

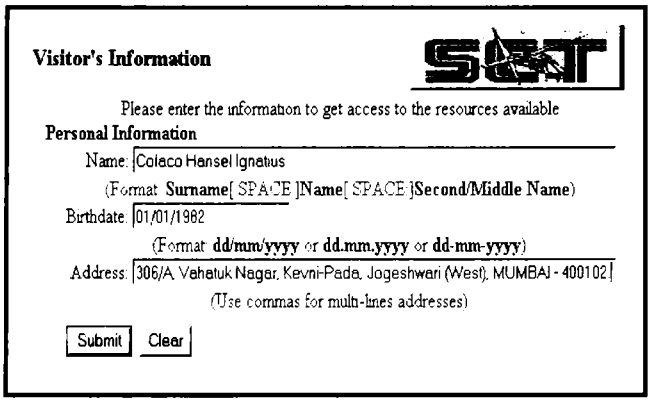


Diagram 15.6.1: Submitting a valid set of visitor's data via RegForm.php.

- Separating elements for the name on the bases of spaces. This is done using the **split()** function:

```
/* Using the split() function to extract visitor's last name,
first name and second name into variables $lname, $fname and
$mname respectively. */
  list($lname, $fname, $mname)    = split(" ", $txtName, 3);
```

The above code spec extracts the last name, first name and the middle name from the form field **txtName** and stores it in the variables **$lname**, **$fname** and **$mname** respectively.

- Formatting the birth date. This is done using the **ereg_replace()** function:

```
/* Using the ereg_replace() to separates the day, month and year
value with spaces. */
  $txtDOB = ereg_replace('[/.-]', ' ', $txtDOB);
```

The above code spec replaces a foreward slash (**/**), a period (**.**) or a hyphen (**-**) with a space.

- Separating elements for the postal address into an array, using the **split()** function:

```
/* Using the split() function to extract the segments held in the
multi-line address. */
  $PostAddr = split(",", $txtAddr);
```

The above code spec extracts the elements of the postal address form field **txtAddr** and stores it in the array $PostAddr. The number of commas determine number of elements in the array $PostAddr.

When the **UseReg.php** file generates an output from valid data a page as shown in diagram 15.6.2 is displayed .

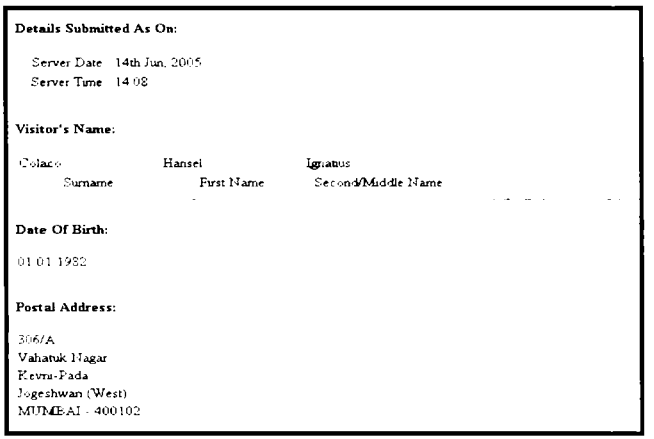


Diagram 15.6. 2: Output generated by UseReg.php when a valid set of visitor's data is submitted.

Hands On Exercises

1. Develop a PHP program (**AuthRegForm.php**), which will generate an Author's Registration form as shown in diagram 15.7.1. The HTML based form should perform the following:
   - Captures data such as Author's name, date of birth and address, (include separate fields for city and state)
   - Accept and validate captured data as all form fields are mandatory
   - Clears the form fields when the Clear button is clicked
   - Submit captured information to **UseReg.php**, when the Submit button is clicked

2. Develop a PHP program (**UseReg.php**), which will retrieve data captured by the HTML form (generate by **AuthRegForm.php**). This program uses regular expressions to validate and format data as shown in diagram 14.7.2.

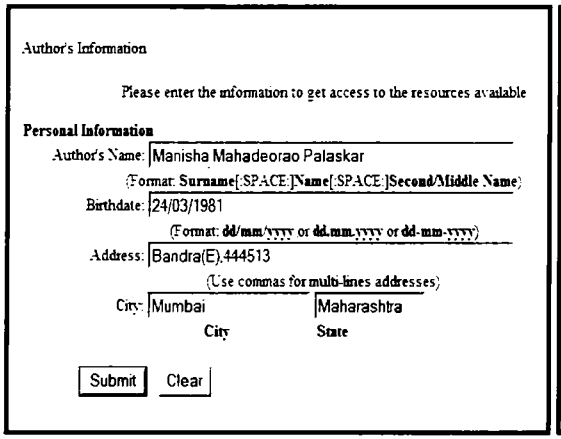


Diagram 14.7.1: Output for AuthRegForm.php.

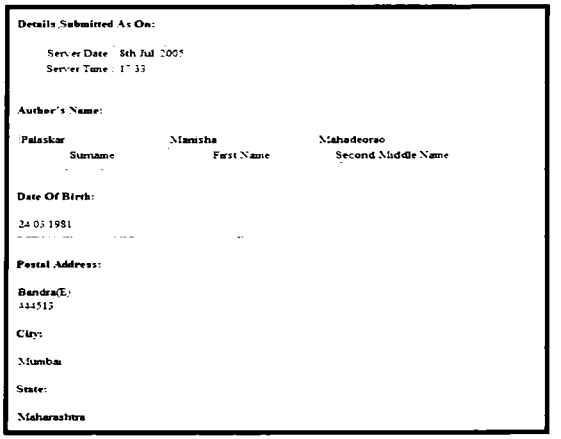


Diagram 14.7.2: Output for UseReg.php.

SECTION III: WORKING WITH PHP

Solutions To Hands On Exercises

10. PHP CONDITIONAL STATEMENTS AND ITERATIONS

1. **Generating a PHP script that will display the multiples of the numbers 1 to 10, upto 10 times: (MultiplicationTab.php)**

```
<?php
  $ctr = 0;
  echo "The multiples of the numbers 1 to 10 upto 10 times are:\n";
  for($i=1; $i<3; $i++) {
      for($j=1; $j<11; $j++) {
          $k = $ctr+1;
          while($k <= ($i*5)) {
              echo "| $k x $j = " . ($k*$j) . " |";
              $k++;
          }
          echo "\n";
      }
      echo "\n";
      $ctr = $ctr + 5;
  }
?>
```

2. **Generating a PHP script that will display the grade on the bases of marks obtained: (TestCondStat.php)**

```
<?php
  $Marks = array(67, 82, 23, 37, 57, 97);
  foreach( $Marks As $mark) {
      if ($mark >= 80) {
          echo "The Result for $mark marks is: Distinction.\n";      }
      else {
          if ($mark >= 60) {
              echo "The Result for $mark marks is: First Class.\n"; }
          else {
              if ($mark >= 45) {
                  echo "The Result for $mark marks is: Second Class.\n";   }
              else {
                  if ($mark >= 35) {
                      echo "The Result for $mark marks is: Pass Class.\n"; }
```

```
                else { echo "The Result for $mark marks is: Fail.\n";     }
            }
        }
    }
  }
?>
```

3. Generating a list of possibilities for two digit numbers, consisting of numbers 0 to 4: (TestForLoop.php)

```
<?php
  for($i=0; $i<5; $i++) {
      for($j=0; $j<5; $j++) {
          echo $i.$j." "; }
  echo "\n";
  }
?>
```

12. WORKING WITH WEB PAGES

1. Generating a data entry form to capture and manipulate information for Book Dimensions:

a. Contents of **DmsnMstr.php:**

```
<?
/*
  Date :- 13/05/05
  Author :- Hansel Colaco.
  Filename :- DmsnMstr.php
  Purpose :- Allows display, insert, updates and delete of
             dimension master table.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Dimension Master Form";
/* If the form mode is set to Delete. */
  if ($hidMode == 'D') {
      /* A variable $Message is initialised to hold a message,
      which has occurred during the previous (delete) operation.
      */
          $Message = '<FONT Color="RED">* Details for '.$hidDelRcrdLst.' has been
                      deleted.</FONT>';   }
/* A variable named $hidMode is initialised to 'I'. This variable
hold the form status in terms of I - Insert, U - Update and D -
Delete. */
  $hidMode = 'I';
?>
```

```
<HTML>
<HEAD>
  <TITLE><?=$title;?></TITLE>
  <META Content="<?=$author;?>" Name="Dimension">
  <META Content="" Name="Keywords">
  <META Content="" Name="Description">
  <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- Java script code-spec to trim a string and change string
case.  -->
  <SCRIPT Language="JavaScript" Src="mstrscript.js" Type="text/javascript">
  </SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
  <BR><BR>
<!-- JavaScript code-spec unique to the dimension master form.-->
  <SCRIPT Language="JavaScript">
  /* The chkBlanks() function verifies that the form field of
  dimension name is not left empty. If the field is empty or
  contains blank spaces, a message indicates to enter a valid
  value. If the field is not empty, the data is submitted. */
     function chkBlanks() {
     /* A variable holding a reference to the form object
     initialised in this page. */
        frm = document.frmMstrDmsn;
     /* Extracting the contents of the dimension field into a
     variable named str. */
        var str = frm.txtDmsnName.value;
     /* If the dimension name field is empty or contains blank
     spaces. */
        if(str.trim() == "") {
        /* Message prompting to enter a valid value. */
           alert('Please enter Dimension name.');
        /* Placing the form cursor on the book name field. */
           frm.txtDmsnName.focus();
        /* Preventing the data in the form field from being
        submitted. */
           return false;
        }
     /* If data in the field Height is numeric or not */
        else if (chkNum('txtDmsnHeight', 'frmMstrDmsn')) {
        /* Preventing the data in the form field from being
        submitted. */
           return false; }
     /* If data in the field Width is numeric or not */
        else if (chkNum('txtDmsnWidth',' frmMstrDmsn')) {
        /* Preventing the data in the form field from being
        submitted. */
           return false; }
```

```
    /* If the Dimension name field holds a value. */
        else {
        /* Calling the function vldtFrmFlds() to validate data
        captured by the form. If the function returns TRUE. */
            if (vldtFrmFlds() == true) {
            /* Allowing the data in the form field to be
            submitted. */
                return true; }
        /* If the function vldtFrmFlds() returns FALSE during
        validation of data captured by the form. */
            else {
            /* Preventing the data in the form field from being
            submitted. */
                return false; }
        }
    }

/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
    function vldtFrmFlds() {
    /* A variable holding a reference to the form object
    initialised in this page. */
        frm = document.frmMstrDmsn;
    /* If the form is in the update mode. */
        if (frm.hidMode.value == 'U') {
        /* Initialising a variable named mNoChng to hold TRUE.
        This value of this variable determines whether data in
        the form field has been modified or not. */
            var mNoChng = true;
        /* Checking the form objects individually for modified
        information. If any changes have been made, the value of
        the variable named mNoChng is set to hold FALSE. */
            if (frm.hidDmsnName.value != frm.txtDmsnName.value) {
                mNcChng = false; }
            if (frm.hidDmsnHeight.value != frm.txtDmsnHeight.value) {
                mNoChng = false; }
            if (frm.hidDmsnWidth.value != frm.txtDmsnWidth.value) {
                mNoChng = false; }
            if (frm.hidMsrgUnt.value != frm.rdoMsrgUnt.value) {
                mNoChng = false; }
        /* If data in the form fields have remained unchanged. */
            if (mNoChng) {
                alert('Update failed.\n No changes have been made during
                    modification.');
```

```
            /* Calling the JavaScript function to reset values in
            the form fields. */
               setNewMode();
            /* Placing the form cursor on the dimension name
            field. */
               frm.txtDmsnName.focus();
            /* Preventing the data in the form field from being
            submitted. */
               return false;
            }
         /* If data in the form fields have been changed. */
            else {
            /* Allowing the data in the form field to be
            submitted. */
               return true;  }
         }
      /* If the form is not in the update mode. */
         else {
         /* Allowing the data in the form field to be submitted.
         */
            return true;  }
      }
   /* The function setNewMode() is called to prepare the form to
   except a new set for data. This function is called directly
   when the Clear button is clicked. */
      function setNewMode() {
      /* A variable holding a reference to the form object
      initialised in this page. */
         frm = document.frmMstrDmsn;
      /* Clearing form fields. */
         frm.txtDmsnName.value = "";
         frm.txtDmsnHeight.value = "";
         frm.txtDmsnWidth.value = "";
         frm.rdoMsrgUnt.value="";
      /* Clearing hidden variables used for comparison during in
      the update operation. */
         frm.hidDmsnID.value = "";
         frm.hidDmsnName.value = "";
         frm.hidDmsnHeight.value = "";
         frm.hidDmsnWidth.value="";
         frm.rdoMsrgUnt[1].checked=true;
         frm.rdoMsrgUnt.value="I";
      /* Setting the form mode to insert. */
         frm.hidMode.value = "I";
      /* If the form contains a Delete button. */
         if (frm.cmdDelete) {
         /* Enabling the Delete button. */
            frm.cmdDelete.disabled = false;      }
      }
```

```
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
   function setEditMode(id, name, hgth, wdth, unt) {
   /* A variable holding a reference to the form object
   initialised in this page. */
      frm = document.frmMstrDmsn;
   /* If the form is not in the update mode. */
      if(frm.hidMode.value != 'U') {
      /* Assigning values passed as parameters to hidden
      variables. These are used for comparison during in the
      update operation. */
         frm.hidDmsnID.value = id;
         frm.hidDmsnName.value = name;
         frm.hidDmsnHeight.value = hgth;
         frm.hidDmsnWidth.value = wdth;
         frm.hidMsrgUnt.value = unt;
      /* Populating the form fields. */
         frm.txtDmsnName.value = name;
         frm.txtDmsnHeight.value = hgth;
         frm.txtDmsnWidth.value = wdth;
         if (unt == "C") {
            frm.rdoMsrgUnt[0].checked=true;
            frm.rdoMsrgUnt.value="C";
         }
         if (unt == "I") {
            frm.rdoMsrgUnt[1].checked=true;
            frm.rdoMsrgUnt.value="I";
         }
      /* Setting the form mode to update. */
         frm.hidMode.value = 'U';
      /* Disabling the Delete button. */
         frm.cmdDelete.disabled = true;
      }
   /* If the form is in the update mode. */
      else {
         alert('Update in progress.\n An entry has already been selected for
               modification.'); }
   }
/* The function setDelMode() is called to prepare the form to
remove data from the tabular layout. This function is called
directly when the Delete button is clicked. */
   function setDelMode() {
   /* Calling the JavaScript function to reset values in the
   form fields. */
      setNewMode();
```

```
      /* Setting the form mode for deleting record(s). */
         document.frmMstrDmsn.hidMode.value = 'D';
      /* Calling the JavaScript function to populate the variable
      holding a list of deleted records. The function
      formDeleteValues() has been defined in the mstrscript.js
      file. */
         formDeleteValues('hidDelRcrdLst', 'frmMstrDmsn');
      }
   </SCRIPT>
<!-- Outer Table Code Begins. -->
   <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0"
         CellSpacing="0" Name="TlbOuter" Width="70%"><TR>
      <TD Align="center" Border="1" BgColor="#C0C0C0"
         Width="10%">Dimensions</TD>
      <TD Align="center" BgColor="#FFFFFF" ColSpan="9"
         Width="90%"><?=$Message;?></TD>
   </TR>
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
   <FORM Action="DmsnMstr.php" Method="post" Name="frmMstrDmsn"
         onSubmit="return chkBlanks();">
   <!-- Declaring a hidden form field used to identify the current
   (i.e. Dimension master) page. -->
      <INPUT Name="hidPage" Type="hidden" Value="DmsnMstr">
   <!-- Declaring hidden form fields required for data validation.
   -->
      <INPUT Name="hidDmsnID" Size="2" Type="hidden" Value="">
      <INPUT Name="hidDmsnName" Size="2" Type="hidden" Value="">
      <INPUT Name="hidDmsnHeight" Size="2" Type="hidden" Value="">
      <INPUT Name="hidDmsnWidth" Size="2" Type="hidden" Value="">
      <INPUT Name="hidMsrgUnt" Size="2" Type="hidden" Value="">
   <!-- Declaring a hidden form field used for determining the
   form mode. It will hold 'I' for Insert, 'U' for Update and 'D'
   for Delete. It will hold 'I' when the page is rendered for the
   1st time. -->
      <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
   <!-- Declaring a hidden form field used for identifying records
   which have been selected for deletion. -->
      <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
   <TR Height="300" VAlign="top">
      <TD Align="center" Border="1" ColSpan="10"><BR>
      <!-- Form Table Code Begins. -->
         <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
               Name="TlbInner" Width="90%"><TR>
            <TD Align="center" ColSpan="4"><FONT Size="2"
               Color="#0000CC"><B>Publishers Management
               System</B></FONT></TD>
         </TR><TR>
```

```
            <TD Align="left" ColSpan="4"><B>Specification for Book
                Dimensions</B></TD>
        </TR><TR>
            <TD Align="right">Dimension Name:</TD>
            <TD Align="left" ColSpan="3"><INPUT maxLength="55"
                Name="txtDmsnName" onBlur="chngCase('txtDmsnName',
                'frmMstrDmsn');" Type="text" Size="50"></TD>
        </TR><TR>
            <TD Align="right">Height:</TD>
            <TD Align="left"><INPUT maxLength="10" Name="txtDmsnHeight"
                Type="text" Size="20"></TD>
            <TD Align="right" Width="12%">Width:</TD>
            <TD Align="left"><INPUT maxLength="10" Name="txtDmsnWidth"
                Type="text" Size="10"></TD>
        </TR><TR title="Measurement Units">
            <TD Align="right" Width="25%">Units:</TD>
            <TD Align="left" ColSpan="3"><INPUT Name="rdoMsrgUnt"
                type="radio" value="C"> Centimeters <INPUT Checked
                Name="rdoMsrgUnt" type="radio" value="I"> Inches</TD>
        </TR><TR>
            <TD Align="center" ColSpan="4"> </TD>
        </TR><TR>
                <TD> </TD>
                <TD Align="left" ColSpan="3">
                <INPUT Name="cmdSubmit" Type="submit" Value="Save">
                <IMG Height="1" Src="../Images/pixel.gif" Width="30">
                <INPUT Name="cmdReset" onClick="setNewMode();"
                        Type="button" Value="Clear">
            </TD>
        </TR></TABLE><BR>
    <!-- Form Table Code End. -->
<?
/* Checking if data have been retrieved from the previous
instance of the dimension form. */
  if ($txtDmsnName != '') {
?>
    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
                Width="90%"><TR BgColor="#400040">
            <TD Width="15%"><INPUT Name="cmdDelete"
                onClick="setDelMode();" Type="button" Value="Delete"></TD>
            <TD><FONT
                Color="#FFFFFF"><B>DimensionName</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Height</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Width</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Units</B></FONT></TD>
        </TR>
```

```
        <TR BgColor="#E4E4E4">
            <TD><INPUT Name="chk1" Type="checkbox" Value="1"></TD>
            <TD><A HRef="JavaScript:setEditMode('1',
                '<?=$txtDmsnName;?>', '<?=$txtDmsnHeight;?>',
                '<?=$txtDmsnWidth;?>',
                '<?=$rdoMsrgUnt;?>')"><?echo($txtDmsnName);?></A></TD>
            <TD><? echo($txtDmsnHeight);?></TD>
            <TD><? echo($txtDmsnWidth);?></TD>
<?      if ($rdoMsrgUnt == "I") { ?>
            <TD>Inches</TD>
<?      }
        else { ?>
            <TD>Centimeters</TD>
<?      } ?>
        </TR>
        </TABLE><BR>
<?  } ?>
    </TD>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends. -->
</BODY>
</HTML>
```

b. Contents of **mstrscript.js:**

```
/*Date :- 13/05/05
  Author :- Hansel Colaco
  Filename :- mstrscript.js
  Purpose :- Defines Generic JavaScript functions for trimming
             strings and changing string case.
*/
/* Generating the function used for trimming string values. */
  function strtrim() {
  /* Returns the string passed as a parameter after removing
  blank spaces, if any, at both the beginning and the end of the
  parameter. */
    return this.replace(/^\s+/,'').replace(/\s+$/,'');
  }

/* Generating a global alias name for the function used for
trimming string values. */
  String.prototype.trim = strtrim;

/* The chngCase() function changes the first alphabet to an upper
case character for the value held in the field passed as the
first parameter. The second parameter is a reference to the form
holding the field of the first parameter. This function is called
when a form cursor moves away from a field. */
  function chngCase(pFld, pFrm) {
```

```
  /* Using the eval() function to extract the value held in the
  field passed as a parameter. */
     var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
  /* Removing extra spaces for the value extracted above and
  restoring into the same variable. */
     mFldVal = mFldVal.trim();
  /* If the variable containing the extracted value is not empty.
  */
     if (mFldVal != "") {
     /* Changing the string Proper case. */
        mFldVal = mFldVal.charAt(0).toUpperCase()
                  + mFldVal.substr(1, mFldVal.length); }
  /* Using the eval() function to display the new value in the
  field passed as a parameter. */
     eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
  }

/* The formDeleteValues() function generates a string of comma-
separated identities of the records selected for deletion. If no
records are selected, a message indicates the same. The
parameters for this function are references to a hidden variable
(used for storing the generated string) and the form
respectively. It is called when the DELETE button is clicked. */
  function formDeleteValues(pHid, pFrm) {
  /* Initialising variables for later use. */
     var selValues = "";
     var firstSelBox = "";
  /* Using the eval() function to create a variable to hold a
  reference to the form object. */
     eval("frm = document." + pFrm);
  /* Iterating through every object on the form. */
     for (i=0; i<frm.elements.length; i++) {
     /* If the current object is a Checkbox. */
        if (frm.elements[i].type == "checkbox") {
        /* If the variable firstSelBox is empty. */
           if (firstSelBox == "") {
           /* Assigning the element number, of the current form
           object, to the variable firstSelBox. */
              firstSelBox = i; }
           /* If the current Checkbox has been selected, (i.e.
           checked). */
           if (frm.elements[i].checked == true) {
           /* Assigning the element number, of the current form
           object, to the variable selValues. */
              selValues = selValues + frm.elements[i].value + ",";      }
        }
     }
  /* If the variable selValues does not holds a value. */
     if (selValues.length < 1) {
```

```
        /* Displaying a message indicating to select a checkbox. */
            alert('Please choose records you wish to delete.');
        /* Using the eval() function to place the form cursor on the
        first checkbox. */
            eval("document." + pFrm + ".elements[" + firstSelBox + "].focus();");
        }
    /* If the variable selValues holds a value. */
        else {
        /* Removing the last character (i.e. a comma) for the value
        held in the variable selValues. */
            selValues = selValues.substring(0, selValues.length-1);
        /* Using eval() function to assign a value to the form's
        hidden variable. */
            eval("document." + pFrm + "." + pHid + ".value = '" + selValues +
                 "';");
            frm.submit();
        }
    }

/* The chkNum() function validates whether the content of the
form field passed as parameter is a number. */
    function chkNum(fldName, pFrm) {
        var mFldVal = eval("document." + pFrm + "." + fldName + ".value;");
        mFldVal = mFldVal.trim();
        if(mFldVal == "") {
            alert('Enter a numeric value');
            eval("document." + pFrm + "." + fldName + ".focus();");
            return true;
        }
        if(isNaN(mFldVal) == true) {
            alert('Characters not allowed');
            eval("document." + pFrm + "." + fldName + ".focus();");
            return true;
        }
        else { return false; }
    }
```

13. BASICS OF FILE HANDLING

1. Generating a data entry form to capture and manipulate information for Book Dimensions. This form uses flat file (dmsnmstr.txt) to store inform passed via the form:

a. Contents of **dmsnmstr.txt:**

```
1:LetterA:11:8.5:I
2:A4:11.69:8.27:I
3:A5:8.27:5.83:I
4:Foolscap:43:34:C
5:B5 (JIS):10.12:7.17:I
```

```
6:B5 (ISO):9.84:6.93:I
7:Crown:19:12:I
8:Royal:63:51:C
9:Fgsdfgh:34:34:C
```

b. Contents of **DmsnMstr.php:**

```
<?
/*
  Date :- 13/05/05
  Author :- Hansel Colaco.
  Filename :- DmsnMstr.php
  Purpose :- Allows display, insert, updates and delete of
             dimension master table.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Dimension Master Form";
/* Storing the name of the file to be read. */
  $filename = "dmsnmstr.txt";
/* A variable holding the message to be displayed on the page. */
  $Message = "";
/* Storing the maximum record number assigned to an entry in the
text file. */
  $maxRcrd = 0;
/* Initialising a variable named $insert to determine whether to
continue the insert operation or not. If it holds 0, the insert
operation is cancelled. */
  $insert = 1;
/* Initialising a variable named $update to determine whether to
continue the update operation or not. If it holds 0, the update
operation is cancelled. */
  $update = 1;
/* If Dimension name has been passed through the variable
$txtDmsnName, (i.e. the $txtDmsnName is not empty). */
  if (!empty($txtDmsnName)) {
  /* Using the function file_exists() to check whether PHP is
  able to locate the dmsnmstr.txt file. */
     if (file_exists($filename)) {
     /* If the dmsnmstr.txt file is found. Use the function
     file() to extract the contents of the dmsnmstr.txt file into
     a variable named $fileContents. */
        $fileContents = file($filename);
     /* Using an iteration to extract and transfer each line held
     in $fileContents into the variable $line. */
        foreach ($fileContents as $line_num => $line) {
```

```
        /* Using the function explode() to separate the segments
        of the current line and store them as individual elements
        for the array $segment. */
           $segment = explode(":", $line);
        /* Checking if the form is in the insert mode. */
           if ($hidMode == 'I') {
           /* Checking for duplication in dimension, i.e.
           dimension name already exists in the text file. */
              if ($segment[1] == $txtDmsnName) {
              /* Assigning $insert with a 0 (zero), to cancel the
              insert operation. */
                 $insert = 0;
              /* Terminating the iteration used for traversing
              through the each line in the text file. */
                 break;
              }
           /* If a match for the new dimension name was not found
           in the text file. */
              else {
              /* Checking if the current entry number is greater
              than the value held in $maxRcrd. */
                 if ($segment[0] > $maxRcrd) {
                 /* Storing the highest entry number into the
                 variable $maxRcrd. */
                    $maxRcrd = $segment[0];
                 }
              }
           }
        /* Checking if the form is in the update mode. */
           if ($hidMode == 'U') {
           /* Checking if modified dimension name already exists
           as a different entry in the text file. */
              if ($segment[1] == $txtDmsnName
                 and $segment[0] != $hidDmsnID) {
              /* Assigning $update with a 0 (zero), to cancel the
              update operation. */
                 $update = 0;
              /* Terminating the iteration used for traversing
              through the each line in the text file. */
                 break;
              }
           }
        }
     }
  }

/* Checking if the form is in the insert mode. */
  if ($hidMode == 'I') {
```

```
/* Checking the value held by $insert. If zero the insert
operation is cancelled. */
   if ($insert == 0) {
   /* Generating and storing an error message which indicates a
   duplication of dimension. */
      $Message = '<FONT Color="red">Entry for <B>'.$txtDmsnName.'</B>
                 already exists. </FONT>';    }
/* If the insert operation is permitted. */
   else {
   /* If the dmsnmstr.txt file is editable, a file handle is
   created using the fopen() function. The handle is created
   using the 'a' switch to allow data to be appended to the
   text file. */
      if (is_writable($filename) and $fileHandle = fopen($filename, 'a')) {
      /* Storing the next highest entry number into the
      variable $newRcrd. */
         $newRcrd = $maxRcrd + 1;
      /* Concatenating a new line character to a string that
      will be appended into the text file and storing it into
      the variable $append. */
         $append =
                 "$newRcrd:$txtDmsnName:$txtDmsnHeight:$txtDmsnWidth:
                 $rdoMsrgUnt" .chr(13).chr(10);
      /* Attempting to append a line into the text file and
      checking if the write operation was successful. */
         if(!fwrite($fileHandle, $append) === FALSE) {
         /* Generating and storing a message that indicates the
         insert operation was successful. */
            $Message = '<FONT Color="blue">Entry for
                       <B>'.$txtDmsnName.'</B>added
                       successfully.</FONT>';      }
      /* If the write operation on the dmsnmstr.txt file fails.
      */
         else {
         /* Generating and storing an error message, which
         indicates a failure in insert the new dimension. */
            $Message = '<FONT Color="red">Failure in adding entry for
                       <B>'.$txtDmsnName.'</B>.</FONT>';      }
      /* Closing the file handle which is used to access and
      append the ctgrymstr.txt file. */
         fclose($fileHandle);
      }
   /* If data cannot be written to the file ctgrymstr.txt or an
   error is encountered while creating a file handle for the
   file. */
      else {
```

```
        /* Generating and storing an error message that indicates
        failure, while accessing the text file. */
            $Message = '<FONT Color="red">Unable to access text file.</FONT>';
        }
    }
  }
/* Checking if the form is in the update mode. */
  if ($hidMode == 'U') {
  /* Checking the value held by $update. If zero the update
  operation is cancelled. */
    if ($update == 0) {
    /* Generating and storing an error message which indicates a
    duplication of dimension. */
        $Message = '<FONT Color="red">The modified dimension
                    <B>'.$txtDmsnName.'</B> already exists.</FONT>';  }
  /* If the update operation is permitted to continue. */
    else {
    /* If the dmsnmstr.txt file is found. Use the function
    file() to extract the contents of the dmsnmstr.txt file into
    a variable named $fileContents. */
        $fileContents = file($filename);
        echo($filecontents);
    /* If the ctgrymstr.txt file is editable, a file handle is
    created using the fopen() function. The handle is created
    using the 'w' switch to allow file data to be truncated,
    (i.e. the file holds nothing). */
        if (is_writable($filename) and $fileHandle = fopen($filename, 'w')) {
        /* Using an iteration to extract and transfer each line
        held in $fileContents into the variable $line. */
            foreach ($fileContents as $line_num => $line) {
            /* Using the function explode() to separate the
            segments of the current line and store them as
            individual elements for the array $segment. */
                $segment = explode(":", $line);
            /* If the first element in $segment is not empty. */
                if (!empty($segment[0])) {
                /* Checking whether the first element for $segment
                hold a value identical to record number for the
                entry selected for modification. */
                    if ($segment[0] == $hidDmsnID) {
                    /* Transferring values from the form fields into
                    the second and third segments for the array
                    object $segment. */
                        $segment[1] = $txtDmsnName;
                        $segment[2] = $txtDmsnHeight;
                        $segment[3] = $txtDmsnWidth;
                        $segment[4] = $rdoMsrgUnt;
                    }
```

```
            /* Ensuring that the last element for $segment ends
            with a new line character. */
               if (ord(substr($segment[4], -1)) == 10) {
               /* Generating the string that will be appended
               into the text file and storing it into the
               variable $append. */
                  $append =
                     "$segment[0]:$segment[1]:$segment[2]:$segment[3]:
                     $segment[4]"; }

            /* When the last element for $segment does not end
            with a new line character. */
               else {
               /* Concatenating a new line character to the
               string that will be appended into the text file
               and storing it into the variable $append. */
                  $append =
                     "$segment[0]:$segment[1]:$segment[2]:$segment[3]:
                     $segment[4]".chr(13).chr(10);      }

            /* Attempting to append a line into the text file
            and checking if the write operation was successful.
            */
               if(!fwrite($fileHandle, $append) === FALSE) {
               /* Generating and storing a message that
               indicates the update operation was successful.
               */
                  $Message = '<FONT Color="blue">Entry for
                              <B>'.$txtDmsnName.'</B> modified
                              successfully.</FONT>';      }

            /* If the write operation on the dmsnmstr.txt file
            fails. */
               else {
               /* Generating and storing an error message,
               which indicates a failure in update the new
               dimension. */
                  $Message = '<FONT Color="red">Failure in changing entry
                              for <B>'.$txtDmsnName.'</B>.</FONT>';}
            }
         }
      /* Closing the file handle which is used to access and
      append the dmsnmstr.txt file. */
         fclose($fileHandle);
      }
```

```
    /* If data cannot be written to the file dmsnmstr.txt or an
    error is encountered while creating a file handle for the
    file. */
       else {
       /* Generating and storing an error message that indicates
       failure while accessing the text file. */
          $Message = '<FONT Color="red">Unable to access text file.</FONT>';
       }
    }
  }
/* Checking if the form is in the delete mode. */
  if ($hidMode == 'D') {
  /* Using the function explode() to separate the entry number(s)
  stored in $hidDelRcrdLst and store them as individual elements
  of the array $delRcrd. */
     $delRcrd = explode(",", $hidDelRcrdLst);
  /* Use the function file() to extract the contents of the
  dmsnmstr.txt file into a variable named $fileContents. */
     $fileContents = file($filename);
  /* If the dmsnmstr.txt file is editable, a file handle is
  create using the fopen() function. The handle is created using
  the 'w' switch to allow file data to be truncated, (i.e. the
  file holds nothing). */
     if (is_writable($filename) and $fileHandle = fopen($filename, 'w')) {
     /* Initialising the variable $Rcrd to act as a counter for
     the entry number. */
        $Rcrd = 1;
     /* Using an iteration to extract and transfer each line held
     in $fileContents into the variable $line. */
        foreach ($fileContents as $line_num => $line) {
        /* Using the function explode() to separate the segments
        of the current line and store them as individual elements
        for the array $segment. */
           $segment = explode(":", $line);
        /* If the first element in $segment is not empty. */
           if (!empty($segment[0])) {
           /* A variable to act as a flag to determine whether
           the current entry is delete or not. If 1 the record is
           not deleted. */
              $writeRcrd = 1;
           /* An iteration through the element in the array
           holding the list of entry number(s) marked for
           deletion. */
              for($i=0; $i<=count($delRcrd); $i++) {
              /* Check if the current element in $delRcrd is
              empty. */
                 if (!empty($delRcrd[$i])) {
```

```
            /* Comparing the current record number held in
            $delRcrd and first element in $segment. */
               if ($segment[0] == $delRcrd[$i]) {
               /* If a match is found, set the variable
               $writeRcrd to hold 0. */
                  $writeRcrd = 0;
               /* Checking if the variable $delEntry is
               empty, to determine the method for storing
               the current dimension into the variable
               $delEntry. */
                  if(empty($delEntry)) {
                     $delEntry = $segment[1]; }
                  else {
                     $delEntry = $delEntry.", ".$segment[1];   }
               }
            }
         }
      /* Checking the value held by $writeRcrd, to determine
      whether the record is retained or not. */
         if ($writeRcrd == 1) {
         /* Ensuring that the last element for $segment ends
         with a new line character. */
            if (ord(substr($segment[4], -1)) == 10) {
            /* Generating the string that will be appended
            into the text file and storing it into the
            variable $append. */
               $append =
                        "$Rcrd:$segment[1]:$segment[2]:$segment[3]
                        :$segment[4]";        }

         /* When the last element for $segment does not end
         with a new line character. */
            else {
            /* Concatenating a new line character to the
            string that will be appended into the text file
            and storing it into the variable $append. */
               $append =
                        "$Rcrd:$segment[1]:$segment[2]:$segment[3]
                        :$segment[4]".chr(13).chr(10);      }
         /* Attempting to append a line into the text file
         and checking if the write operation fails. */
            if(fwrite($fileHandle, $append) === FALSE) {
            /* Generating and storing an error message,
            which indicates a failure in insert the new
            dimension. */
               $Message = '<FONT Color="red">Failure in re-creating the
                           file.</FONT>';        }
```

```
                /* Incrementing the entry number and storing it
                into the variable $Rcrd. */
                   $Rcrd = $Rcrd + 1;
                }
             }
          }
       /* Closing the file handle which is used to access and
       append the dmsnmstr.txt file. */
          fclose($fileHandle);
       /* Checking if the variable $delEntry is not empty. */
          if(!empty($delEntry)) {
          /* A variable $Message is initialised to hold a message,
which list the record(s) deleted. */
             $Message = '<BR><FONT Color="RED">* Details for '.$delEntry.' has
                         been deleted.</FONT>'; }
       }
    /* If data cannot be written to the file dmsnmstr.txt or an
    error is encountered while creating a file handle for the file.
    */
       else {
       /* Generating and storing an error message that indicates
       failure while accessing the text file. */
          $Message = '<FONT Color="red">Unable to access text file.</FONT>';
       }
    }
/* A variable named $hidMode is initialised to 'I'. This variable
holds the form status in terms of I - Insert, U - Update and D -
Delete. */
    $hidMode = 'I';
?>
<HTML>
<HEAD>
    <TITLE><?=$title;?></TITLE>
    <META Content="<?=$author;?>" Name="Dimension">
    <META Content="" Name="Keywords">
    <META Content="" Name="Description">
    <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- JavaScript code-spec to trim a string and change string
case. -->
    <SCRIPT Language="JavaScript" Src="mstrscript.js" Type="text/javascript">
    </SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
    <BR><BR>
<!-- JavaScript code-spec unique to the dimension master form.-->
    <SCRIPT Language="JavaScript">
```

```
/* The chkBlanks() function verifies that the form field of
dimension name is not left empty. If the field is empty or
contains blank spaces, a message indicates to enter a valid
value. If the field is not empty, the data is submitted. */
   function chkBlanks() {
   /* A variable holding a reference to the form object
   initialised in this page. */
      frm = document.frmMstrDmsn;
   /* Extracting the contents of the dimension field into a
   variable named str. */
      var str = frm.txtDmsnName.value;
   /* If the dimension name field is empty or contains blank
   spaces. */
      if(str.trim() == "") {
      /* Message prompting to enter a valid value. */
         alert('Please enter Dimension name.');
      /* Placing the form cursor on the dimension name field.
      */
         frm.txtDmsnName.focus();
      /* Preventing the data in the form field from being
      submitted. */
         return false;
      }
   /* If data in the field Height is numeric or not */
      else if (chkNum('txtDmsnHeight', 'frmMstrDmsn')) {
      /* Preventing the data in the form field from being
      submitted. */
         return false; }
   /* If data in the field Width is numeric or not */
      else if (chkNum('txtDmsnWidth', 'frmMstrDmsn')) {
      /* Preventing the data in the form field from being
      submitted. */
         return false; }
   /* If the dimension name field holds a value. */
      else {
      /* Calling the function vldtFrmFlds() to validate data
      captured by the form. If the function returns TRUE. */
         if (vldtFrmFlds() == true) {
         /* Allowing the data in the form field to be
         submitted. */
            return true;      }
      /* If the function vldtFrmFlds() returns FALSE during
      validation of data captured by the form. */
         else {
         /* Preventing the data in the form field from being
         submitted. */
            return false; }
      }
   }
```

```
/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
   function vldtFrmFlds() {
   /* A variable holding a reference to the form object
   initialised in this page. */
      frm = document.frmMstrDmsn;
   /* If the form is in the update mode. */
      if (frm.hidMode.value == 'U') {
      /* Initialising a variable named mNoChng to hold TRUE.
      This value of this variable determines whether data in
      the form field has been modified or not. */
         var mNoChng = true;
      /* Checking the form objects individually for modified
      information. If any changes have been made, the value of
      the variable named mNoChng is set to hold FALSE. */
         if (frm.hidDmsnName.value != frm.txtDmsnName.value) {
            mNoChng = false; }
         if (frm.hidDmsnHeight.value != frm.txtDmsnHeight.value) {
            mNoChng = false; }
         if (frm.hidDmsnWidth.value != frm.txtDmsnWidth.value) {
            mNoChng = false; }
         if (frm.hidMsrgUnt.value != frm.rdoMsrgUnt.value) {
            mNoChng = false; }
      /* If data in the form fields have remained unchanged. */
         if (mNoChng) {
            alert('Update failed.\n No changes have been made during
                  modification.');
         /* Calling the JavaScript function to reset values in
         the form fields. */
            setNewMode();
         /* Placing the form cursor on the dimension name
         field. */
            frm.txtDmsnName.focus();
         /* Preventing the data in the form field from being
         submitted. */
            return false;
         }
      /* If data in the form fields have been changed. */
         else {
         /* Allowing the data in the form field to be
         submitted. */
            return true;  }
      }
   /* If the form is not in the update mode. */
      else {
```

```
        /* Allowing the data in the form field to be submitted.
        */
            return true;   }
    }
/* The function setNewMode() is called to prepare the form to
except a new set for data. This function is called directly
when the Clear button is clicked. */
    function setNewMode() {
    /* A variable holding a reference to the form object
    initialised in this page. */
        frm = document.frmMstrDmsn;
    /* Clearing form fields. */
        frm.txtDmsnName.value = "";
        frm.txtDmsnHeight.value = "";
        frm.txtDmsnWidth.value = "";
        frm.rdoMsrgUnt.value="";
    /* Clearing hidden variables used for comparison during in
    the update operation. */
        frm.hidDmsnID.value = "";
        frm.hidDmsnName.value = "";
        frm.hidDmsnHeight.value = "";
        frm.hidDmsnWidth.value="";
        frm.rdoMsrgUnt[1].checked=true;
        frm.rdoMsrgUnt.value="I";
    /* Setting the form mode to insert. */
        frm.hidMode.value = "I";
    /* If the form contains a Delete button. */
        if (frm.cmdDelete) {
        /* Enabling the Delete button. */
            frm.cmdDelete.disabled = false;       }
    }
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
    function setEditMode(id, name, hgth, wdth, unt) {
    /* A variable holding a reference to the form object
    initialised in this page. */
        frm = document.frmMstrDmsn;
    /* If the form is not in the update mode. */
        if(frm.hidMode.value != 'U') {
        /* Assigning values passed as parameters to hidden
        variables. These are used for comparison during in the
        update operation. */
            frm.hidDmsnID.value = id;
            frm.hidDmsnName.value = name;
            frm.hidDmsnHeight.value = hgth;
            frm.hidDmsnWidth.value = wdth;
```

```
        /* Populating the form fields. */
            frm.txtDmsnName.value = name;
            frm.txtDmsnHeight.value = hgth;
            frm.txtDmsnWidth.value = wdth;
            if (unt == "C") {
                frm.rdoMsrgUnt[0].checked=true;
                frm.rdoMsrgUnt.value="C";
                frm.hidMsrgUnt.value = "C";
            }
            if (unt == "I") {
                frm.rdoMsrgUnt[1].checked=true;
                frm.rdoMsrgUnt.value="I";
                frm.hidMsrgUnt.value = "I";
            }
        /* Setting the form mode to update. */
            frm.hidMode.value = 'U';
        /* Disabling the Delete button. */
            frm.cmdDelete.disabled = true;
        }
    /* If the form is in the update mode. */
        else {
            alert('Update in progress.\n An entry has already been selected for
                modification.');  }
    }
  /* The function setDelMode() is called to prepare the form to
  remove data from the tabular layout. This function is called
  directly when the DELETE button is clicked. */
    function setDelMode() {
    /* Calling the JavaScript function to reset values in the
    form fields. */
        setNewMode();
    /* Setting the form mode for deleting record(s). */
        document.frmMstrDmsn.hidMode.value = 'D';
    /* Calling the JavaScript function to populate the variable
    holding a list of deleted records. The function
    formDeleteValues() has been defined in the mstrscript.js
    file. */
        formDeleteValues('hidDelRcrdLst', 'frmMstrDmsn');
    }
  </SCRIPT>
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="#E4E4E4" Border="0" CellPadding="0"
        CellSpacing="0" Name="TlbOuter" Width="70%"><TR>
    <TD Align="center" Border="1" BgColor="#C0C0C0"
        Width="10%">Dimensions</TD>
    <TD Align="center" BgColor="#FFFFFF" ColSpan="9"
        Width="90%"><?=$Message;?></TD>
  </TR>
```

```
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="DmsnMstr.php" Method="post" Name="frmMstrDmsn"
        onSubmit="return chkBlanks();">
  <!-- Declaring a hidden form field used to identify the current
  (i.e. dimension master) page. -->
     <INPUT Name="hidPage" Type="hidden" Value="DmsnMstr">
  <!-- Declaring hidden form fields required for data validation.
  -->
     <INPUT Name="hidDmsnID" Size="2" Type="hidden" Value="">
     <INPUT Name="hidDmsnName" Size="2" Type="hidden" Value="">
     <INPUT Name="hidDmsnHeight" Size="2" Type="hidden" Value="">
     <INPUT Name="hidDmsnWidth" Size="2" Type="hidden" Value="">
     <INPUT Name="hidMsrgUnt" Size="2" Type="hidden" Value="">
  <!-- Declaring a hidden form field used for determining the
  form mode. It will holds 'I' for Insert, 'U' for Update and 'D'
  for Delete. It will hold 'I' when the page is rendered for the
  1st time. -->
     <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
  <!-- Declaring a hidden form field used for identifying records
  which have been selected for deletion. -->
     <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
  <TR Height="300" VAlign="top">
     <TD Align="center" Border="1" ColSpan="10"><BR>
     <!-- Form Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
               Name="TlbInner" Width="90%"><TR>
           <TD Align="center" ColSpan="4"><FONT Size="2"
              Color="#0000CC"><B>Publishers Management
              System</B></FONT></TD>
        </TR><TR>
           <TD Align="left" ColSpan="4"><B>Specification for Book
              Dimensions</B></TD>
        </TR><TR>
           <TD Align="right">Dimension Name:</TD>
           <TD Align="left" ColSpan="4"><INPUT maxLength="55"
              Name="txtDmsnName" onBlur="chngCase('txtDmsnName',
              'frmMstrDmsn');" Type="text" Size="50"></TD>
        </TR><TR>
           <TD Align="right">Height:</TD>
           <TD Align="left"><INPUT maxLength="10" Name="txtDmsnHeight"
              Type="text" Size="20"></TD>
           <TD Align="right" Width="12%">Width:</TD>
           <TD Align="left"><INPUT maxLength="10" Name="txtDmsnWidth"
              Type="text" Size="10"></TD>
        </TR><TR title="Measurement Units">
           <TD Align="right" Width="25%">Units:</TD>
```

```
            <TD Align="left" ColSpan="3"
                <INPUT Name="rdoMsrgUnt" type="radio" value="C">
                    Centimeters
                <INPUT Checked Name="rdoMsrgUnt" type="radio" value="I">
                    Inches
            </TD>
        </TR><TR>
            <TD Align="center" ColSpan="4"> </TD>
        </TR><TR>
                <TD> </TD>
                <TD Align="left" ColSpan="3">
                <INPUT Name="cmdSubmit" Type="submit" Value="Save">
                <IMG Height="1" Src="../Images/pixel.gif" Width="30">
                <INPUT Name="cmdReset" onClick="setNewMode();"
                    Type="button" Value="Clear">
            </TD>
        </TR></TABLE><BR>
    <!-- Form Table Code End. -->
<?
/* Using the function file_exists() to check whether PHP is able
to locate the dmsnmstr.txt file. */
  if (file_exists($filename)) {
  /* If the dmsnmstr.txt file is found. */
?>
    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
            Width="90%">
        <TR BgColor="#400040">
            <TD Width="15%"><INPUT Name="cmdDelete"
                onClick="setDelMode();" Type="button" Value="Delete"></TD>
            <TD><FONT Color="#FFFFFF"><B>Dimension
                Name</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Height</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Width</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Units</B></FONT></TD>
        </TR>
<?
  /* Use the function file() to extract the contents of the
  dmsnmstr.txt file into a variable named $fileContents. */
      $fileContents = file($filename);
  /* Using an iteration to extract and transfer each line held in
  $fileContents into the variable $line. */
      foreach ($fileContents as $line_num => $line) {
      /* Using the function explode() to separate the segments of
      the current line and store them as individual elements for
      the array $segment. */
          $segment = explode(":", $line);
?>
      <TR BgColor="#E4E4E4">
```

```
     <!-- Creating a check box using the value held by the first
     element of the array $segment. -->
        <TD><INPUT Name="chk<?=($segment[0]);?>" Type="checkbox"
            Value="<?=($segment[0]);?>"></TD>
     <!-- Creating a hyperlink using the value held by the second
     element of the array $segment. -->
        <TD><A HRef="JavaScript:setEditMode('<?=($segment[0]);?>',
            '<?=($segment[1]);?>', '<?=($segment[2]);?>',
            '<?=($segment[3]);?>', '<?=(substr($segment[4],0,1));?>')">
            <?echo($segment[1]);?></A></TD>
     <!-- Displaying the value held by the third element of the
     array $segment. -->
        <TD><?=($segment[2]);?></TD>
        <TD><?=($segment[3]);?></TD>
<?      if (substr($segment[4],0,1) == "I") {    ?>
        <TD>Inches</TD>
<?      }
        else { ?>
        <TD>Centimeters</TD>
<?      } ?>
     </TR>
<?   }   ?>
     </TABLE><BR>
<?
  }
/* If PHP cannot locate the dmsnmstr.txt file. */
  else {
?>
  <FONT Color="red"><B>Unable to open file!</B></FONT>
<?
  /* Terminating the any further processes on the page. */
     exit();
  }
?>
     </TD>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends. -->
</BODY>
</HTML>
```

c. Contents of **mstrscript.js** is same as mentioned in the solution for Chapter 12.

14. INTEGRATION BETWEEN PHP AND ORACLE

1. Generating a data entry form to capture and manipulate information for Book Dimensions. This form uses table given below to store inform passed via the form:

a. SQL queries for creation and population of the **DmsnMstr** table:

```
CREATE TABLE DmsnMstr(DmsnID Number(3) PRIMARY KEY,
  Name VarChar2(15) DEFAULT NULL, BkHght Number(4, 2) NOT NULL,
  BkWdth Number(4, 2) NOT NULL,
  MsrgUnt VarChar2(1) CHECK(MsrgUnt IN('C', 'I')));

INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(1, 'Letter', 11, 8.5, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(2, 'A4', 11.69, 8.27, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(3, 'A5', 8.27, 5.83, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(4, 'Foolscap', 43, 34, 'C');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(5, 'B5 (JIS)', 10.12, 7.17, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(6, 'B5 (ISO)', 9.84, 6.93, 'I');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(7, 'Crown', 19, 12, 'C');
INSERT INTO DmsnMstr (DmsnID, Name, BkHght, BkWdth, MsrgUnt)
  VALUES(8, 'Royal', 63, 51, 'C');
```

c. Contents of **DmsnMstr.php:**

```
<?
/*Date :- 13/05/05
  Author :- Hansel Colaco.
  Filename :- DmsnMstr.php
  Purpose :- Allows display, insert, updates and delete of
             dimension master table.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco.';
/* A variable holding the title for the page. */
  $title = "Dimension Master Form";
/* Variables used to store colours for the master page. */
  $gl_mstr_top_bar_color = '#400040';
  $gl_mstr_frst_bar_color = '#E4E4E4';
  $gl_mstr_scnd_bar_color = '#C0C0C0';
/* Variables used to store Database access information. */
  $dbuser = "DBA_PUB";
  $dbpass = "sct2306";
```

```
/* The function ocilogon() accepts the oracle login information
to create the actual connection to the database. This connection
made available whenever required by referencing $rcq, which the
connection object created by the ocilogon() function. */
  $rcq = ocilogon($dbuser, $dbpass);

/* Checking whether the creation of the connection object failed
or not. If failed, an error message is generated. */
  if(!$rcq){
     $Message = '<FONT Color="red"><B>Database connection
                 error!</B><BR>Please contact the Oracle Database
                 Administrator.</FONT><BR>';  }
/* If the creation of the connection object succeeded. */
  else {
  /* If Category name has been passed through the variable
  $txtCatName, (i.e. the $txtCatName is not empty). */
     if (!empty($txtDmsnName)) {
     /* Verifying that the form is currently in the INSERT mode.
     */
        if ($hidMode == "I") {
        /* Building and executing a SQL query to retrieve records
        for the Category table such that the contents of the
        Category_Name matches the value entered in the category
        name field in the form. */
           $query = "SELECT * FROM DmsnMstr WHERE TRIM(LOWER(Name))
                    = TRIM(LOWER('".$txtDmsnName."'))";
           $parsed = ociparse($rcq, $query);
           ociexecute($parsed);
        /* Checking if any record was retrieved by the above SQL
        query. If the same category name exists in the database,
        an error message is displayed to the user, which
        indicates that the Insert operation has failed. */
           if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
           /* Generating and storing an error message which
           indicates a duplication of category. */
              $Message = '<FONT Color="red">Entry for
                          <B>'.$txtDmsnName.'</B> already exists.
                          </FONT>';
           }
        /* When the same category name does not exist in the
        databases, the new category is stored into the database.
        */
           else {
```

```
        /* Building and executing a SQL query to INSERT the
        new category into the category table. The new category
        is assigned a unique ID, which is the highest ID
        assigned to a category incremented by one. */
            $query = "INSERT INTO DmsnMstr (DmsnID, Name,
                    BkHght,BkWdth,MsrgUnt) VALUES((SELECT
                    COALESCE(MAX(DmsnID), 0)+1 FROM
                    DmsnMstr),'$txtDmsnName', $txtDmsnHeight,
                    $txtDmsnWidth, '$rdoMsrgUnt')";
            $parsed = ociparse($rcq, $query);
            ociexecute($parsed);
        /* Generating and storing a message that indicates the
        insert operation was successful. */
            $Message = '<FONT Color="blue">Entry for
                        <B>'.$txtDmsnName.'</B> added
                        successfully.</FONT>';
        }
    }
/* Verifying that the form is currently in the UPDATE mode.
*/
    if ($hidMode == "U") {
    /* Building and executing a SQL query to retrieve records
    for the category table such that the contents of the
    Category_Name matches the value entered in the category
    name field in the form, while the unique record number
    does not match the unique record number held in the
    hidden form field $hidCatID. */
        $query = "SELECT * FROM DmsnMstr WHERE TRIM(LOWER(Name))
                = TRIM(LOWER('$txtDmsnName')) AND NOT
                (DmsnID=$hidDmsnID)";
        $parsed = ociparse($rcq, $query);
        ociexecute($parsed);
    /* Checking if any record was retrieved by the above SQL
    query. If the same category name exists in another
    record, an error message is displayed to the user, which
    indicates that the Update operation has failed. */
        if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
        /* Generating and storing an error message which
        indicates a duplication of category. */
            $Message = '<FONT Color="red">The modified dimension
                        <B>'.$txtDmsnName.'</B> already exists.</FONT>';
        }
    /* When the same category name does not exist in another
    record, the modified category data is stored into the
    table. */
        else {
```

```
            /* Building and executing a SQL query to Update the
            category table with the change in the category data.
            The record to be changed is determined on the bases of
            the unique ID assigned to the category data. */
                $query = "UPDATE DmsnMstr SET Name='$txtDmsnName',
                        BkHght=$txtDmsnHeight, BkWdth=$txtDmsnWidth,
                        MsrgUnt='$rdoMsrgUnt' WHERE DmsnID=$hidDmsnID";
                $parsed = ociparse($rcq, $query);
                ociexecute($parsed);
            /* Generating and storing a message that indicates the
            update operation was successful. */
                $Message = '<FONT Color="blue">Entry for
                            <B>'.$txtDmsnName.'</B> modified
                            successfully.</FONT>';
            }
        }
    }
/* Checking if the form is in the delete mode. */
    if ($hidMode == 'D') {
    /* Building and executing a SQL query to retrieve category
    names from the category table such that the unique record
    number matches the entries selected for deletion, (i.e.
    value held in $hidDelRcrdLst). */
        $query = "SELECT Name FROM DmsnMstr WHERE DmsnID
                  IN($hidDelRcrdLst)";
        $parsed = ociparse($rcq, $query);
        ociexecute($parsed);
    /* An iterative code performed as long as a single row of
    category data is available. */
        while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM)) {
        /* Checking if the variable $delEntry is empty, to
        determine the method for storing the current category
        into the variable $delEntry. */
            if(empty($delEntry)) {   $delEntry = $line['NAME']; }
            else {  $delEntry = $delEntry.", ".$line['NAME'];      }
        }
    /* Checking if the variable $delEntry is not empty. */
        if(!empty($delEntry)) {
        /* Building and executing a SQL query to delete record(s)
        for the category table. The condition of deleting
        record(s) is that the unique record number should exists
        in the list of entries selected for deletion, (i.e. value
        held in $hidDelRcrdLst). */
            $query = "DELETE FROM DmsnMstr WHERE DmsnID
                      IN($hidDelRcrdLst)";
            $parsed = ociparse($rcq, $query);
            ociexecute($parsed);
```

```
        /* A variable $Message is initialised to hold a message,
        which list the record(s) deleted. */
            $Message = '<BR><FONT Color="RED">Details for
                        <B>'.$delEntry.'</B> has been deleted.</FONT>';
        }
      /* If the variable $delEntry is empty. */
        else {
        /* A variable $Message is initialised to hold an error
        message, which indicates that records for deletion do not
        exists in the database. */
            $Message = '<BR><FONT Color="RED">Records selected for deletion
                        do not exists. </FONT>';
        }
      }
    }
/* A variable named $hidMode is initialised to 'I'. This variable
hold the form status in terms of I - Insert, U - Update and D -
Delete. */
    $hidMode = 'I';
/* A variable $count is initialised to hold zero. This is used in
while loops to generate unique checkbox names for the tabular
layout. */
    $count = 0;
?>
<HTML>
<HEAD>
    <TITLE><?=$title;?></TITLE>
    <META Content="<?=$author;?>" Name="Dimension">
    <META Content="" Name="Keywords">
    <META Content="" Name="Description">
    <META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!-- JavaScript code-spec to trim a string and change string case. -->
    <SCRIPT Language="JavaScript" Src="mstrscript.js" Type="text/javascript">
    </SCRIPT>
</HEAD>
<BODY LeftMargin="0" MarginHeight="0" MarginWidth="0" TopMargin="0">
    <BR><BR>
<!-- Java script code-spec unique to the dimension master form. -->
    <SCRIPT Language="JavaScript">
    /* The chkBlanks() function verifies that the form field of
    dimension name is not left empty. If the field is empty or
    contains blank spaces, a message indicates to enter a valid
    value. If the field is not empty, the data is submitted. */
        function chkBlanks() {
        /* A variable holding a reference to the form object
        initialised in this page. */
            frm = document.frmMstrDmsn;
```

```
    /* Extracting the contents of the dimension field into a
    variable named str. */
       var str = frm.txtDmsnName.value;
    /* If the dimension name field is empty or contains blank
    spaces. */
       if(str.trim() == "") {
       /* Displaying a message indicates to enter a valid value.
       */
          alert('Please enter Dimension name.');
       /* Placing the form cursor on the book name field. */
          frm.txtDmsnName.focus();
       /* Preventing the data in the form field from being
       submitted. */
          return false;
       }
    /* If data in the field Height is numeric or not */
       else if (chkNum('txtDmsnHeight', 'frmMstrDmsn')) {
       /* Preventing the data in the form field from being
       submitted. */
          return false; }
    /* If data in the field Width is numeric or not */
       else if (chkNum('txtDmsnWidth', 'frmMstrDmsn')) {
       /* Preventing the data in the form field from being
       submitted. */
          return false; }
    /* If the category name field holds a value. */
       else {
       /* Calling the function vldtFrmFlds() to validate data
       captured by the form. If the function returns TRUE. */
          if (vldtFrmFlds() == true) {
          /* Allowing the data in the form field to be
          submitted. */
             return true; }
       /* If the function vldtFrmFlds() returns FALSE during
       validation of data captured by the form. */
          else {
          /* Preventing the data in the form field from being
          submitted. */
             return false; }
       }
    }

/* The function vldtFrmFlds() is called when the form data is
about to be submitted. This function verifies the presence of
changes in the form fields during the update mode. If changes
are not encountered, then a indicating the same is displayed
and any further processing gets terminated. */
   function vldtFrmFlds() {
```

```
/* A variable holding a reference to the form object
initialised in this page. */
   frm = document.frmMstrDmsn;
/* If the form is in the update mode. */
   if (frm.hidMode.value == 'U') {
   /* Initialising a variable named mNoChng to hold TRUE.
   This value of this variable determines whether data in
   the form field has been modified or not. */
      var mNoChng = true;
   /* Checking the form objects individually for modified
   information. If any changes have been made, the value of
   the variable named mNoChng is set to hold FALSE. */
      if (frm.hidDmsnName.value != frm.txtDmsnName.value) {
         mNoChng = false; }
      if (frm.hidDmsnHeight.value != frm.txtDmsnHeight.value) {
         mNoChng = false; }
      if (frm.hidDmsnWidth.value != frm.txtDmsnWidth.value) {
         mNoChng = false; }
      if (frm.hidMsrgUnt.value != frm.rdoMsrgUnt.value) {
         mNoChng = false; }
   /* If data in the form fields have remained unchanged. */
      if (mNoChng) {
         alert('Update failed.\n No changes have been made during
               modification.');
      /* Calling the JavaScript function to reset values in
      the form fields. */
         setNewMode();
      /* Placing the form cursor on the dimension name
      field. */
         frm.txtDmsnName.focus();
      /* Preventing the data in the form field from being
      submitted. */
         return false;
      }
   /* If data in the form fields have been changed. */
      else {
      /* Allowing the data in the form field to be
      submitted. */
         return true;  }
   }
/* If the form is not in the update mode. */
   else {
/* Allowing the data in the form field to be submitted. */
      return true;  }
}
```

```
/* The function setNewMode() is called to prepare the form to
except a new set for data. This function is called directly
when the Clear button is clicked. */
   function setNewMode() {
   /* A variable holding a reference to the form object
   initialised in this page. */
      frm = document.frmMstrDmsn;
   /* Clearing form fields. */
      frm.txtDmsnName.value = "";
      frm.txtDmsnHeight.value = "";
      frm.txtDmsnWidth.value = "";
      frm.rdoMsrgUnt.value="";
   /* Clearing hidden variables used for comparison during in
   the update operation. */
      frm.hidDmsnID.value = "";
      frm.hidDmsnName.value = "";
      frm.hidDmsnHeight.value = "";
      frm.hidDmsnWidth.value="";
      frm.rdoMsrgUnt[1].checked=true;
      frm.rdoMsrgUnt.value="I";
   /* Setting the form mode to insert. */
      frm.hidMode.value = "I";
   /* If the form contains a Delete button. */
      if (frm.cmdDelete) {
      /* Enabling the Delete button. */
         frm.cmdDelete.disabled = false;      }
   }
/* The function setEditMode() is called to prepare the form to
update data held in the tabular layout. This function is called
directly when a link in the tabular layout is clicked. The
function also sets the hidden field form mode to update and
populates the text fields. */
   function setEditMode(id, name, hgth, wdth, unt) {
   /* A variable holding a reference to the form object
   initialised in this page. */
      frm = document.frmMstrDmsn;
   /* If the form is not in the update mode. */
      if(frm.hidMode.value != 'U') {
      /* Assigning values passed as parameters to hidden
      variables. These are used for comparison during in the
      update operation. */
         frm.hidDmsnID.value = id;
         frm.hidDmsnName.value = name;
         frm.hidDmsnHeight.value = hgth;
         frm.hidDmsnWidth.value = wdth;
         frm.hidMsrgUnt.value = unt;
```

```
        /* Populating the form fields. */
            frm.txtDmsnName.value = name;
            frm.txtDmsnHeight.value = hgth;
            frm.txtDmsnWidth.value = wdth;

            if (unt == "C") {
                frm.rdoMsrgUnt[0].checked=true;
                frm.rdoMsrgUnt.value="C";
            }

            if (unt == "I") {
                frm.rdoMsrgUnt[1].checked=true;
                frm.rdoMsrgUnt.value="I";
            }

        /* Setting the form mode to update. */
            frm.hidMode.value = 'U';
        /* Disabling the Delete button. */
            frm.cmdDelete.disabled = true;
        }
    /* If the form is in the update mode. */
        else {
            alert('Update in progress.\n An entry has already been selected for
                modification.');  }
    }
  /* The function setDelMode() is called to prepare the form to
  remove data from the tabular layout. This function is called
  directly when the Delete button is clicked. */
    function setDelMode() {
    /* Calling the JavaScript function to reset values in the
    form fields. */
        setNewMode();
    /* Setting the form mode for deleting record(s). */
        document.frmMstrDmsn.hidMode.value = 'D';
    /* Calling the JavaScript function to populate the variable
    holding a list of deleted records. The function
    formDeleteValues() has been defined in the mstrscript.js
    file. */
        formDeleteValues('hidDelRcrdLst', 'frmMstrDmsn');
    }
  </SCRIPT>
<!-- Outer Table Code Begins. -->
  <TABLE Align="center" BgColor="<?=$gl_mstr_frst_bar_color;?>"Border="0"
        CellPadding="0" CellSpacing="0" Name="TlbOuter" Width="70%"><TR>
    <TD Align="center" Border="1" BgColor="#C0C0C0"
        Width="10%">Dimensions</TD>
    <TD Align="center" BgColor="#FFFFFF" ColSpan="9"
        Width="90%"><?=$Message;?></TD>
  </TR>
```

```
<!-- Initialising a form object, which will submit data captured
on the form to the processing file. -->
  <FORM Action="DmsnMstr.php" Method="post" Name="frmMstrDmsn"
        onSubmit="return chkBlanks();">
  <!-- Declaring a hidden form field used to identify the current
  (i.e. category master) page. -->
     <INPUT Name="hidPage" Type="hidden" Value="DmsnMstr">
  <!-- Declaring hidden form fields required for data validation.
  -->
     <INPUT Name="hidDmsnID" Size="2" Type="hidden" Value="">
     <INPUT Name="hidDmsnName" Size="2" Type="hidden" Value="">
     <INPUT Name="hidDmsnHeight" Size="2" Type="hidden" Value="">
     <INPUT Name="hidDmsnWidth" Size="2" Type="hidden" Value="">
     <INPUT Name="hidMsrgUnt" Size="2" Type="hidden" Value="">
  <!-- Declaring a hidden form field used for determining the
  form mode. It will holds 'I' for Insert, 'U' for Update and 'D'
  for Delete. It will hold 'I' when the page is rendered for the
  1st time. -->
     <INPUT Name="hidMode" Type="hidden" Value="<?=$hidMode;?>">
  <!-- Declaring a hidden form field used for identifying records
  which have been selected for deletion. -->
     <INPUT Name="hidDelRcrdLst" Type="hidden" Value="">
  <TR Height="300" VAlign="top">
     <TD Align="center" Border="1" ColSpan="10"><BR>
     <!-- Form Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="2" CellSpacing="0"
               Name="TlbInner" Width="90%"><TR>
           <TD Align="center" ColSpan="4"><FONT Size="2"
               Color="#0000CC"><B>Publishers Management
               System</B></FONT></TD>
        </TR><TR>
           <TD Align="left" ColSpan="4"><B>Specification for Book
               Dimensions</B></TD>
        </TR><TR>
           <TD Align="right">Dimension Name:</TD>
           <TD Align="left" ColSpan="3"><INPUT maxLength="55"
               Name="txtDmsnName" onBlur="chngCase('txtDmsnName',
               'frmMstrDmsn');" Type="text" Size="50"></TD>
        </TR><TR>
           <TD Align="right">Height:</TD>
           <TD Align="left"><INPUT maxLength="10" Name="txtDmsnHeight"
               Type="text" Size="20"></TD>
           <TD Align="right" Width="12%">Width:</TD>
           <TD  Align="left"><INPUT maxLength="10" Name="txtDmsnWidth"
               Type="text" Size="10"></TD>
        </TR><TR title="Measurement Units">
           <TD Align="right" Width="25%">Units:</TD>
           <TD Align="left" ColSpan="3"><INPUT Name="rdoMsrgUnt"
                Type="radio" Value="C"> Centimeters
```

```
                <INPUT Checked Name="rdoMsrgUnt" type="radio" value="I">
                        Inches
            </TD>
        </TR><TR>
            <TD Align="center" ColSpan="4"> </TD>
        </TR><TR>
                <TD> </TD>
                <TD Align="left" ColSpan="4">
                <INPUT Name="cmdSubmit" Type="submit" Value="Save">
                <IMG Height="1" Src="../Images/pixel.gif" Width="30">
                <INPUT Name="cmdReset" onClick="setNewMode();"
                        Type="button" Value="Clear">
            </TD>
        </TR></TABLE><BR>
    <!-- Form Table Code End. -->
<?
/* Building and executing a SQL query to retrieve records from
the CATEGORY table after sorting the category names in alphabetic
order. */
  $query = "SELECT * FROM DmsnMstr ORDER BY Name";
  $parsed = ociparse($rcq, $query);
  ociexecute($parsed);
/* Checking if records have been retrieved from the Category
table. */
  if (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM) > 0) {
?>
    <!-- Data Table Code Begins. -->
        <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
                Width="90%">
        <TR BgColor="<?=$gl_mstr_top_bar_color;?>">
            <TD Width="15%"><INPUT Name="cmdDelete"
                onClick="setDelMode();" Type="button" Value="Delete"></TD>
            <TD><FONT
                Color="#FFFFFF"><B>DimensionName</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Height</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Width</B></FONT></TD>
            <TD><FONT Color="#FFFFFF"><B>Units</B></FONT></TD>
        </TR>
<?
  /* A variable named $rowBgColor hold the hexadecimal colour
  value. This variable is used decided the background colour for
  rows of data in the tabular format. */
    $rowBgcolor = $gl_mstr_scnd_bar_color;
  /* An iterative code performed as long as a single row of
  category data is available. */
    do {
    /* Incrementing the value of the variable used for assigning
    unique names to check boxes display of each row of data. */
        $count = $count + 1;
```

```
    /* If the colour code in for the current row matches the
    value for the second row. */
       if ($rowBgcolor == $gl_mstr_scnd_bar_color) {
       /* Colour code for the current row set to match the value
       for the colour code for the first bar. */
          $rowBgcolor = $gl_mstr_frst_bar_color;      }
    /* If the colour code in for the current row matches the
    value for the first row. */
       else {
       /* Colour code for the current row set to match the value
       for the colour code for the second bar. */
          $rowBgcolor = $gl_mstr_scnd_bar_color;      }
?>
       <TR BgColor="<?=$rowBgcolor;?>">
          <TD><INPUT Name="chk<?=($count);?>" Type="checkbox"
               Value="<?=($line['DMSNID']);?>"></TD>
          <TD><A HRef="JavaScript:setEditMode('<?=$line['DMSNID'];?>',
               '<?=$line['NAME'];?>', '<?=$line['BKHGHT']?>',
               '<?=$line['BKWDTH']?>', '<?=$line['MSRGUNT']?>')">
               <?echo($line['NAME']); ?></A></TD>
          <TD><? echo($line['BKHGHT']); ?></TD>
          <TD><? echo($line['BKWDTH']); ?></TD>
<?        if ($line['MSRGUNT'] == "I") {    ?>
          <TD>Inches</TD>
<?        }
          else { ?>
          <TD>Centimeters</TD>
<?        } ?>
          </TR>
<?   } while (ocifetchinto($parsed, $line, OCI_ASSOC+OCI_NUM))      ?>
          </TABLE><BR>
<?} ?>
  </FORM>
  </TR></TABLE>
  <!-- Outer Table Code Ends. -->
</BODY>
</HTML>
```

d. Contents of **mstrscript.js** is same as mentioned in the solution for Chapter 12.

15. REGULAR EXPRESSION

1. Generating a PHP program (**AuthRegForm.php**), which will generate an Author's Registration form:

```
<?
/*
  Date :- 14/06/05
  Author :- Hansel Colaco.
  Filename :- AuthRegForm.php
  Purpose :- Formats Author's data like name, birth date and
             address using regular expressions.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco';
?>
<HTML>
<HEAD><TITLE>Capturing Data From Author</TITLE></HEAD>
<BODY>
  <SCRIPT Language="JavaScript">
  /* Generating the function used for trimming string values. */
     function strtrim() {
     /* Returns the string passed as a parameter after removing
     blank spaces, if any, at both the beginning and the end of
     the parameter. */
        return this.replace(/^\s+/,'').replace(/\s+$/,'');
     }

  /* Generating a global alias name for the function used for
  trimming string values. */
     String.prototype.trim = strtrim;

  /* The function rmvSpace() trim extra spaces at either ends of
  the value entered in the form field pass as parameter. The
  function accepts two parameters; first is the form name and the
  second is the field name. */
     function rmvSpace(pFld, pFrm) {
     /* Using the eval() function to extract the value held in
     the field passed as a parameter. */
        var mFldVal = eval("document." + pFrm + "." + pFld + ".value;");
     /* Removing extra spaces for the value extracted above and
     restoring into the same variable. */
        mFldVal = mFldVal.trim();
     /* Using the eval() function to display the new value in the
     field passed as a parameter. */
        eval("document." + pFrm + "." + pFld + ".value = mFldVal;");
     }
```

```
  /* The function clrFrmFlds() clear the contents of the form
  fields when the Clear button is clicked. */
     function clrFrmFlds() {
     /* A variable holding a reference to the form object
     initialised in this page. */
        frm = document.frmAuthorInfo;
     /* Clearing form fields. */
        frm.txtName.value = "";
        frm.txtDOB.value = "";
        frm.txtAddr.value = "";
        frm.txtCity.value="";
        frm.txtState.value="";
     }
  </SCRIPT>
  <TABLE Border="0" BgColor="#FFFFFF" CellPadding="0" CellSpacing="0"
        Width="600"><TR>
     <TD Colspan="2" Height="15" Width="400"></TD>
  </TR><TR>
     <TD BgColor="#FFFFFF" Width="20"> </TD>
     <TD BgColor="#FFFFFF" Width="380"<H4>Author's Information</H4></TD>
  </TR><TR>
     <TD ColSpan="3">
        <TABLE Border="0" CellPadding="0" CellSpacing="0"
              Width="100%"><TR>
           <TD Width="14"> </TD>
           <TD><?=$msg;?>
<!-- Initial code corresponding to the template ends. -->

<!-- Designing the layout of the form, which will capture the
Author's name, birthdate and address. The data captured will be
submitted to the AuthRegForm.php page. A JavaScript based
validation is performed on the captured information before it is
submitted. -->
<TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0" Height="225"
       Width="100%"><TR>
  <TD Width="20"></TD>
  <TD VAlign="top" Width="560">
<FORM Action="UseReg.php" Method="post" Name="frmAuthorInfo">
     <CENTER>Please enter the information to get access to the resources
           available<BR></CENTER>
     <TABLE Align="center" Border="0" CellPadding="0" CellSpacing="0"
           Width="100%"><TR>
        <TD ColSpan="3"><B>Personal Information</B>
        </TD>
     </TR><TR>
        <IMG Height="1" Src="../Images/pixel.gif" Width="30">
        <TD Align="right" >Author's Name: </TD>
```

```
    <TD Align="left" ColSpan="2"><INPUT Size="50" Name="txtName"
        onBlur="rmvSpace('txtName', 'frmAuthorInfo');" Type="text"
        Value="<?=$txtName;?>"></TD>
</TR><TR>
    <TD Align="center" ColSpan="3"><FONT Color="blue">(Format:
        <B>Surname</B>[:SPACE:]<B>Name</B>[:SPACE:]<B>Second /
        Middle Name</B>)</FONT></TD>
</TR><TR>
    <TD Align="right">Birthdate: </TD>
    <TD Align="left" ColSpan="2"><INPUT Size="50" Name="txtDOB"
        onBlur="rmvSpace('txtDOB', 'frmAuthorInfo');" Type="text"
        Value="<?=$txtDOB;?>"></TD>
</TR><TR>
    <TD Align="center" ColSpan="3"><FONT Color="blue">(Format:
        <B>dd/mm/yyyy</B> or <B>dd.mm.yyyy</B> or <B>dd-mm-
        yyyy</B>)</FONT></TD>
</TR><TR>
    <TD Align="right">Address: </TD>
    <TD Align="left" ColSpan="2"><INPUT Size="50" Name="txtAddr"
        onBlur="rmvSpace('txtAddr', 'frmAuthorInfo');" Type="text"
        Value="<?=$txtAddr;?>"></TD>
</TR><TR>
    <TD Align="center" ColSpan="3"><FONT Color="blue">(Use commas for
        multi-lines addresses)</FONT></TD>
</TR><TR>
<TD Align="right">City: </TD>
    <TD Align="left" ColSpan="5">
        <INPUT class="text" Size="20" maxLength="20" Name="txtCity"
               onBlur="rmvSpace('txtCity', 'frmAthrInfo');" Type="text"
               Value="<?=$txtCity;?>">
        <INPUT class="text" Size="12" maxLength="20" Name="txtState"
               onBlur="rmvSpace('txtState', 'frmAthrInfo');" Type="text"
               Value="<?=$txtState;?>">
    </TD>
</TR><TR>
    <TD Align="right"> </TD>
    <TD Align="left" ColSpan="5">
        <IMG Height="1" Src="../images/pixel.gif" Width="50"><B>City</B>
        <IMG Height="1" Src="../images/pixel.gif" Width="75">
             <B>State</B>
    </TD>
</TR><TR>
    <TD ColSpan="6"><IMG Height="10" Src="../images/pixel.gif"
        Width="1"></TD>
</TR><TR>
    <TD ColSpan="3"><IMG Height="10" Src="../images/pixel.gif"
        Width="1"></TD>
</TR><TR>
    <TD Align="right"><INPUT Name="cmdSubmit" Type="submit"
        Value="Submit"></TD>
```

```
        <TD><IMG Height="10" Src="../images/pixel.gif" Width="1"></TD>
        <TD Align="left"><INPUT Name="cmdClear" onClick="clrFrmFlds();"
            Type="Button" Value="Clear"></TD>
    </TR></TABLE>
  </FORM>
  </TD><TD VAlign="top" Width="20"><BR></TD>
</TR></TABLE>

<!-- Final code corresponding to the template begins. -->
            </TD>
            <TD Width="14"> </TD>
        </TR></TABLE>
    </TD>
  </TR></TABLE>
</BODY>
</HTML>
```

2. Generating a PHP program (**UseReg.php**), which will retrieve data captured by the HTML form (generate by **AuthRegForm.php**):

```
<?
/*
  Date :- 08/07/05
  Author :- Hansel Colaco.
  Filename :- UseReg.php
  Purpose :- Validates and displays Author's data like name,
birth date and address using regular expressions.
*/
/* A variable holding the developers name. */
  $author = 'Hansel Colaco';

/* Storing the system date in the format 1st Jan, 2000. */
  $sysdt = date("jS M, Y");
/* Storing the system time in the format 00:00. */
  $systm = date("H:i");

/* Checking if the variable for visitor's name is empty, (i.e. a
blank name field was submitted). */
  if(empty($txtName)) {
  /* Returning to the data entry form, if the Author's name is
  empty. */
      $msg = '<FONT Color="red"><B>Field for Author\'s Name is
               empty!</B></FONT>';
      include "AuthRegForm.php";
      exit;
  }
/* When the visitor's name is submitted. */
  else {
```

```
  /* Using regular expressions with the ereg() function, to
  identify an occurrence of double spaces in the Author's name.
  */
     if(ereg("([[:space:]]){2}", $txtName)) {
     /* Returning to the data entry form, if the Author's name
     contains double spaces. */
        $msg = '<FONT Color="red"><B>Double spaces found in Author\'s
                Name!</B></FONT>';
        include "AuthRegForm.php";
        exit;
     }
  }

/* Checking if the variable for Author's birth date is empty,
(i.e. a blank date of birth field was submitted). */
  if(empty($txtDOB)) {
  /* Returning to the data entry form, if the visitor's birth
  date is empty. */
     $msg = '<FONT Color="red"><B>Field for Author\'s birth date is
             empty!</B></FONT>';
     include "AuthRegForm.php";
     exit;
  }
/* When the Author's birth date is submitted. */
  else {
  /* Returning to the data entry form, if the format for the
  Author's birth data is not as specified in the data entry form.
  */
     if(!ereg("^([0-9]){2}[/.-]([0-9]){2}[/.-]([0-9]){4}", $txtDOB)) {
        $msg = '<FONT Color="red"><B>Invalid format for Author\'s birth
               date!</B></FONT>';
        include "AuthRegForm.php";
        exit;
     }
  }

/* Checking if the variable for Author's postal address is empty,
(i.e. a blank address field was submitted). */
  if(empty($txtAddr)) {
  /* Returning to the data entry form, if the Author's postal
  address is empty. */
     $msg = '<FONT Color="red"><B>Field for Author\'s postal address is
             empty!</B></FONT>';
     include "AuthRegForm.php";
     exit;
  }
/* Checking if the variable for Author's postal address is empty,
(i.e. a blank address field was submitted). */
  if(empty($txtCity)) {
```

```
  /* Returning to the data entry form, if the Author's postal
  address is empty. */
      $msg = '<FONT Color="red"><B>Field for Atuhor\'s city is
             empty!</B></FONT>';
      include "AuthRegForm.php";
      exit;
  }
/* Checking if the variable for Author's postal address is empty,
(i.e. a blank address field was submitted). */
  if(empty($txtState)) {
  /* Returning to the data entry form, if the Author's postal
  address is empty. */
      $msg = '<FONT Color="red"><B>Field for Atuhor\'s State is
             empty!</B></FONT>';
      include "AuthRegForm.php";
      exit;
  }
?>
<HTML><HEAD><TITLE>Formatting Using Regular Expression</TITLE></HEAD>
<BODY>
  <H4>Details Submitted As On:</H4>
  <TABLE Border="1" CellSpacing="1" Width="200"><TR>
      <TD Align="right" Width="100">Server Date :</TD>
      <TD Align="left" Width="100"><?=$sysdt;?></TD>
  </TR><TR>
      <TD Align="right" Width="100">Server Time :</TD>
      <TD Align="left" Width="100"><?=$systm;?></TD>
  </TR></TABLE><HR>
<?
/* Using the split() function to extract Author's last name,
first name and second name into variables $lname, $fname and
$mname respectively. */
  list($lname, $fname, $mname) = split(" ", $txtName, 3);
?>
  <H4>Author's Name:</H4>
  <TABLE Border="1" CellSpacing="1" Width="500"><TR>
      <TD Align="left" Width="150"><?=$lname;?></TD>
      <TD Align="left" Width="150"><?=$fname;?></TD>
      <TD Align="left" Width="200"><?=$mname;?></TD>
  </TR><TR>
      <TD Align="center" Width="150">Surname</TD>
      <TD Align="center" Width="150">First Name</TD>
      <TD Align="center" Width="150">Second/Middle Name</TD>
  </TR></TABLE><HR>
<?
/* Using the ereg_replace() function to separates the day, month
and year value with spaces. */
  $txtDOB = ereg_replace('[/.-]', ' ', $txtDOB);
?>
```

```
  <H4>Date Of Birth:</H4>
  <TABLE Border="0" CellSpacing="1" Width="300"><TR>
      <TD Align="left" Width="150"><?=$txtDOB;?></TD>
  </TR></TABLE><HR>
  <H4>Postal Address:</H4>
  <TABLE Border="0" CellSpacing="1" Width="400">
<?
/* Using the split() function to extract the segments held in the
multi-line address. */
  $PostAddr = split(",", $txtAddr);
/* Using an iteration to display the Author's address. */
  foreach($PostAddr as $fld) {
?>
      <TR><TD Align="left" Width="150"><?=$fld;?></TD></TR>
<?   }  ?>
  </TABLE>
  <H4>City:</H4>
  <TABLE Border="0" CellSpacing="1" Width="400">
  <TR><TD Align="left" Width="150"><?=$txtCity;?></TD></TR>
  </TABLE>
  <H4>State:</H4>
  <TABLE Border="0" CellSpacing="1" Width="400">
  <TR><TD Align="left" Width="150"><?=$txtState;?></TD></TR>
  </TABLE>
</BODY>
</HTML>
```

SECTION IV: PROJECT DEVELOPMENT USING ORACLE AND PHP

Personnel Management System

One of our clients is a top of the line, recruitment agency, for the Persian Gulf region. They are in a specialized niche where they cater to the needs of a select few clients who are big players in the hospitality business. The term hospitality business covers four and five star hotels at one end of the spectrum and work camp, catering at the other end of the spectrum.

Our client sources educated, certified, trained and experienced personnel from across the globe for delivery to their clients in the Persian Gulf. India, Nepal, Sri Lanka, Bangladesh, UK, Philippines and so on are just some of the countries that our client does business with.

To achieve their goals our client has appointed agents in all these countries who source appropriate personnel that suit the hospitality industry across the board, from housekeeping staff to pastry chefs to VP Marketing and so on.

The Current Business Model

Our client's principals, send them a fax specifying:

- The levels of candidates required
- Specific attributes associated with each level
- The total number of candidates required at each level

and so on.

This marks the start of a specific recruitment drive carried out by our clients on behalf of their principals.

Once the levels and numbers are known, our client in turn activates a specific agent in a specific country by sending them a fax indicating the level /s, of candidate /s, together with the specific attributes required.

The agents in turn use their tried and tested techniques to acquire the resumes of personnel with attributes that match (or marginally exceed) the attributes specified.

Once the resumes stream in, they go through one level of stringent filtering at our client's place before being dispatched to the principals offices for further processing.

REMINDER

All resumes are tagged with each agent's identity, saved in our client's database and then forwarded to their principals by courier.

Once, these resumes are received at the principal's end they go through another level of stringent checking. Further filtering (**i.e.** elimination of candidates) can happen. Once this is done the principal sends our client a list of all the names of the short listed candidates who will be interviewed in their country of origin.

After setting up a mutually acceptable interview date, the principal's representative (generally from their HR section) then travels to the country required to interview each of the short listed candidates. This results lead to further filtering of candidates (**i.e.** candidates failing the interview).

Subsequently, a final list is prepared of selected candidates whose visas will be processed by the principal so that they can take up their contractual obligations.

At this time the passports and other travel documents required for visa processing are collected from each candidate and held by our client. This is carefully documented.

Within about a 20 day period the visas are processed at the principals end and dispatched to our client via fax. Our client then takes:

- The fax of the visa document received
- The candidate travel documents
- The candidate

and approaches **specific medical centers** approved by the embassy from where the entry/work visa will be stamped in a candidates passport to complete their medical formalities.

WARNING

There is a very small percent of candidates who reach this level and are rejected on medical grounds.

Once this formality is complete and the candidate has been declared medically fit to work at the country of destination our client takes:

- The fax of the visa document received
- The candidate travel documents
- The candidate's medical certificate

- The candidate's airline ticket to the country of destination

and approaches the appropriate embassy in Mumbai for stamping of their entry/work visa in the candidate's passport, prior their departure to the country of destination.

Once this formality is complete, candidates are flown to their work destinations, by the earliest available flight, to take up their contractual obligations.

The above defined business model has been decomposed to develop a Personnel Management System using the following table structure.

Diagram 16.0.0 provides a visual representation of **E**ntity **R**elationships **i.e.** the relationships between the tables used to store business data of the Personnel Management System.

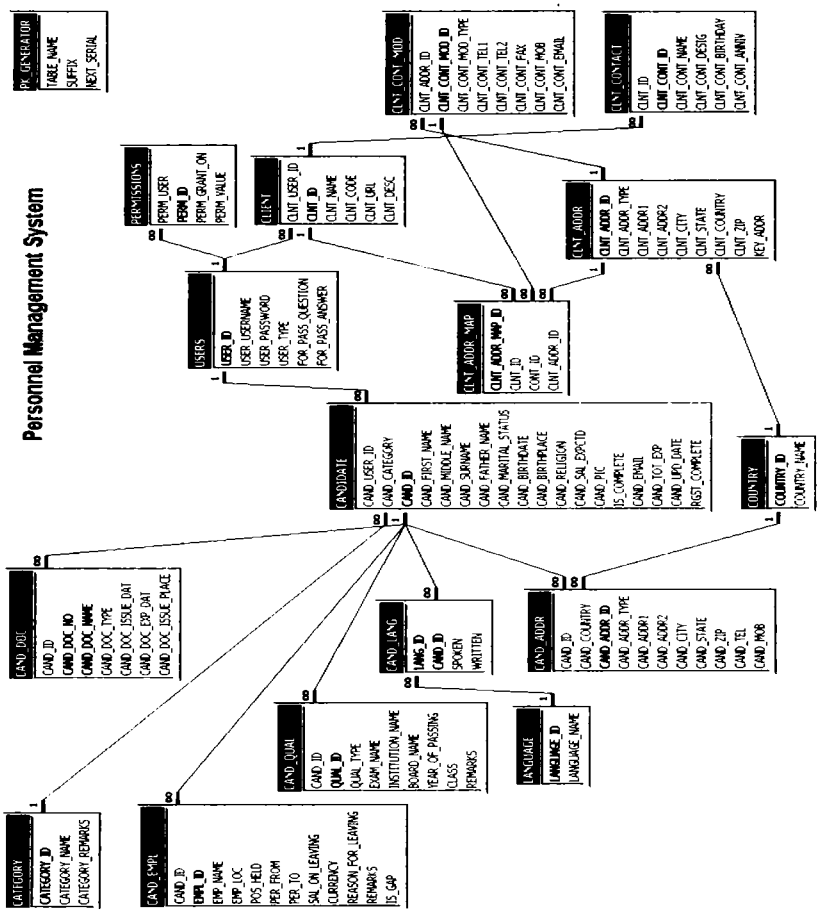


Diagram 16.0.0

REMINDER

The symbols **1** and **1** indicate a one-to-one relationship, while the symbols **1** and ∞ indicate a one-to-many relationship.

Fully normalized table structures together with a detailed description of each table column used in the personnel management system follows.

System And Referential Information

Table Definition:

Table Name : **PK_GENERATOR**
Primary Key : TABLE_NAME
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
TABLE_NAME	VARCHAR2	35	NOT NULL	
SUFFIX	VARCHAR2	5	NOT NULL	
NEXT_SERIAL	NUMBER	4	NOT NULL	

Table Description:

Column Name	Description
TABLE_NAME	Table name for generating ID
SUFFIX	Suffix for the table
NEXT_SERIAL	The unique number for the respective table

Explanation:
The PK_GENERATOR table stores the unique number for forming primary key for each of the tables.

Table Definition:

Table Name : **CATEGORY**
Primary Key : CATEGORY_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CATEGORY_ID	NUMBER	2	NOT NULL	
CATEGORY_NAME	VARCHAR2	35	NOT NULL	
CATEGORY_REMARKS	VARCHAR2	255		NULL

Table Description:

Column Name	Description
CATEGORY_ID	Internal ID. Auto Generated
CATEGORY_NAME	Name of the category
CATEGORY_REMARKS	Remarks for the category if any

Explanation:
The CATEGORY table is a lookup table, which records the categories available.

Table Definition:

Table Name : **COUNTRY**
Primary Key : COUNTRY_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
COUNTRY_ID	NUMBER	2	NOT NULL	
COUNTRY_NAME	VARCHAR2	35	NOT NULL	

Table Description:

Column Name	Description
COUNTRY_ID	Internal ID. Auto Generated
COUNTRY_NAME	Name of the country

Explanation:
The COUNTRY table is a lookup table, which records the names of countries.

Table Definition:

Table Name : **LANGUAGE**
Primary Key : LANGUAGE_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
LANGUAGE_ID	NUMBER	2	NOT NULL	
LANGUAGE_NAME	VARCHAR2	35	NOT NULL	

Table Description:

Column Name	Description
LANGUAGE_ID	Internal ID. Auto Generated
LANGUAGE_NAME	Name of the language

Explanation:
The LANGUAGE table is a lookup table, which records the names of languages (known to a candidate).

Authentication Information

Table Definition:

Table Name : **USERS**
Primary Key : USER_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
USER_ID	VARCHAR2	7	NOT NULL	
USER_USERNAME	VARCHAR2	15	NOT NULL	
USER_PASSWORD	VARCHAR2	20	NOT NULL	
USER_TYPE	VARCHAR2	5	NOT NULL	admn / cand / clnt
FOR_PASS_QUESTION	VARCHAR2	100		NULL
FOR_PASS_ANSWER	VARCHAR2	50		NULL

Table Description:

Column Name	Description
USER_ID	Auto Generated unique user number beginning with **US**, followed by 5 digits
USER_USERNAME	Username for the user to login
USER_PASSWORD	Password for the user
USER_TYPE	User types: candidate (cand), client (clnt), administrator (admn)
FOR_PASS_QUESTION	The question for Forgot Password Option
FOR_PASS_ANSWER	The answer for forgot password option

Explanation:
The USERS table stores authentication information of system users (clients and candidates).

Table Definition:
Table Name : **PERMISSIONS**
Primary Key : PERM_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
PERM_USER	VARCHAR2	7	NOT NULL	
PERM_ID	VARCHAR2	7	NOT NULL	
PERM_GRANT_ON	VARCHAR2	35	NOT NULL	
PERM_VALUE	NUMBER	2	NOT NULL	

Table Description:

Column Name	Description
PERM_USER	User's login name. Reference existing in USERS.USER_ID
PERM_ID	Internal ID. Auto generated
PERM_GRANT_ON	Module name
PERM_VALUE	Permission value assigned

Explanation:
The PERMISSIONS table stores the permissions assigned to users of the system (clients and candidates).

Client Information

Table Definition:

Table Name : **CLIENT**
Primary Key : CLNT_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_USER_ID	VARCHAR2	7		
CLNT_ID	NUMBER	4		
CLNT_NAME	VARCHAR2	35		
CLNT_CODE	VARCHAR2	5		
CLNT_URL	VARCHAR2	50		
CLNT_DESC	VARCHAR2	255		

Table Description:

Column Name	Description
CLNT_USER_ID	Client's login name. Reference existing in USERS.USER_ID
CLNT_ID	Internal ID number. Auto generated
CLNT_NAME	Name of the client
CLNT_CODE	Code Used by the company
CLNT_URL	URL of company website
CLNT_DESC	Any information the client wishes to furnish

Explanation:
The CLIENT table stores information of companies/clients.

Table Definition:

Table Name : **CLNT_CONTACT**
Primary Key : CLNT_CONT_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ID	NUMBER	4		
CLNT_CONT_ID	NUMBER	4		
CLNT_CONT_NAME	VARCHAR2	35		
CLNT_CONT_DESIG	VARCHAR2	35		
CLNT_CONT_BIRTHDAY	VARCHAR2	12		
CLNT_CONT_ANNIV	VARCHAR2	12		

Table Description:

Column Name	Description
CLNT_ID	Client's number. Reference existing in CLIENT.CLNT_ID
CLNT_CONT_ID	Internal ID number for the contact person. Auto generated
CLNT_CONT_NAME	Name of the contact person associated with the client.
CLNT_CONT_DESIG	Designation
CLNT_CONT_BIRTHDAY	Birthday of the contact person
CLNT_CONT_ANNIV	Anniversary of the contact person

Explanation:
The CLNT_CONTACT table stores information of the key contacts in a particular company.

Table Definition:
Table Name : **CLNT_ADDR**
Primary Key : CLNT_ADDR_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ADDR_ID	NUMBER	4	NOT NULL	
CLNT_ADDR_TYPE	VARCHAR2	20	NOT NULL	
CLNT_ADDR1	VARCHAR2	50	NOT NULL	
CLNT_ADDR2	VARCHAR2	50		NULL
CLNT_CITY	VARCHAR2	35	NOT NULL	
CLNT_STATE	VARCHAR2	35	NOT NULL	
CLNT_COUNTRY	NUMBER	2	NOT NULL	
CLNT_ZIP	VARCHAR2	10		NULL
KEY_ADDR	VARCHAR2	3		NULL

Table Description:

Column Name	Description
CLNT_ADDR_ID	Internal ID number for the client address information. Auto generated
CLNT_ADDR_TYPE	Type of address. Whether official/personal
CLNT_ADDR1	Address Line 1
CLNT_ADDR2	Address Line 2
CLNT_CITY	City of the contact
CLNT_STATE	State
CLNT_COUNTRY	Country reference
CLNT_ZIP	Pincode of the client address
KEY_ADDR	Identifier for the key address of the client

Explanation:
The CLNT_ADDR table stores multiple addresses of the clients.

Table Definition:

Table Name : **CLNT_CONT_MOD**
Primary Key : CLNT_CONT_MOD_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ADDR_ID	NUMBER	4		
CLNT_CONT_MOD_ID	NUMBER	4		
CLNT_CONT_MOD_TYPE	VARCHAR2	20		
CLNT_CONT_TEL1	VARCHAR2	35		
CLNT_CONT_TEL2	VARCHAR2	35		
CLNT_CONT_FAX	VARCHAR2	35		
CLNT_CONT_MOB	VARCHAR2	35		
CLNT_CONT_EMAIL	VARCHAR2	50		

Table Description:

Column Name	Description
CLNT_ADDR_ID	Reference for Address information for a client existing in CLNT_ADDR.CLNT_ADDR_ID
CLNT_CONT_MOD_ID	Internal ID number for the client contact information. Auto generated
CLNT_CONT_MOD_TYPE	Contact Mode Type. Whether official/personal
CLNT_CONT_TEL1	Telephone Number 1
CLNT_CONT_TEL2	Telephone Number 2
CLNT_CONT_FAX	Fax number of the contact
CLNT_CONT_MOB	Mobile number for contact
CLNT_CONT_EMAIL	Email address for contact

Explanation:
The CLNT_CONT_MOD table stores the different contact numbers/modes for a client.

Table Definition:

Table Name : **CLNT_ADDR_MAP**
Primary Key : CLNT_ADDR_MAP_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CLNT_ADDR_MAP_ID	NUMBER	4		
CLNT_ID	NUMBER	4		
CONT_ID	NUMBER	4		
CLNT_ADDR_ID	NUMBER	4		

Table Description:

Column Name	Description
CLNT_ADDR_MAP_ID	Internal ID number for the link between clients and addresses. Auto generated
CLNT_ID	Client number. Reference existing in CLIENT.CLNT_ID
CONT_ID	Reference for Contact information for a client existing in CLNT_CONT_MOD.CLNT_CONT_MOD_ID
CLNT_ADDR_ID	Reference for Address information for a client existing in CLNT_ADDR.CLNT_ADDR_ID

Explanation:
The CLNT_ADDR_MAP table stores the mapping of addresses and contact information to clients.

Candidate Information

Table Definition:
Table Name : **CANDIDATE**
Primary Key : CAND_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_USER_ID	VARCHAR2	7	NOT NULL	
CAND_CATEGORY	NUMBER	2		
CAND_ID	VARCHAR2	7		
CAND_FIRST_NAME	VARCHAR2	35		
CAND_MIDDLE_NAME	VARCHAR2	35		
CAND_SURNAME	VARCHAR2	35		
CAND_FATHER_NAME	VARCHAR2	35		
CAND_MARITAL_STATUS	VARCHAR2	4		
CAND_BIRTHDATE	VARCHAR2	12		
CAND_BIRTHPLACE	VARCHAR2	35		
CAND_RELIGION	VARCHAR2	35		
CAND_SAL_EXPCTD	VARCHAR2	20		
CAND_PIC	VARCHAR2	50		
IS_COMPLETE	VARCHAR2	2		
CAND_EMAIL	VARCHAR2	50		
CAND_TOT_EXP	VARCHAR2	8		
CAND_UPD_DATE	VARCHAR2	12		
RGST_COMPLETE	VARCHAR2	3		

Table Description:

Column Name	Description
CAND_USER_ID	Candidate's login name. Reference existing in USERS.USER_ID
CAND_CATEGORY	Category number. Reference existing in CATEGORY.CATEGORY_ID
CAND_ID	A unique address number beginning with CD, followed by 5 digits
CAND_FIRST_NAME	First Name of candidate
CAND_MIDDLE_NAME	Middle Name of candidate
CAND_SURNAME	Surname of candidate
CAND_FATHER_NAME	Father's Name of Candidate
CAND_MARITAL_STATUS	Marital Status of candidate: Yes (Yes) / No (No)
CAND_BIRTHDATE	Date of Birth
CAND_BIRTHPLACE	Place of Birth
CAND_RELIGION	Religion
CAND_SAL_EXPCTD	Salary Expected
CAND_PIC	Image Path of the Candidate's photograph
IS_COMPLETE	Status of the Record. Whether complete
CAND_EMAIL	E-mail
CAND_TOT_EXP	Total Job Experience
CAND_UPD_DATE	Last Updated Date For Experience
RGST_COMPLETE	Registration Status

Explanation:
The CANDIDATE table records personal details of the candidates.

Table Definition:

Table Name : **CAND_ADDR**
Primary Key : CAND_ADDR_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7		
CAND_COUNTRY	NUMBER	2		
CAND_ADDR_ID	VARCHAR2	7		
CAND_ADDR_TYPE	VARCHAR2	20		
CAND_ADDR1	VARCHAR2	50		
CAND_ADDR2	VARCHAR2	50		
CAND_CITY	VARCHAR2	35		
CAND_STATE	VARCHAR2	35		
CAND_ZIP	VARCHAR2	20		
CAND_TEL	VARCHAR2	35		
CAND_MOB	VARCHAR2	35		

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
CAND_COUNTRY	Country number. Reference existing in COUNTRY.COUNTRY_ID
CAND_ADDR_ID	A unique address number beginning with **CA**, followed by 5 digits
CAND_ADDR_TYPE	Address Type: Official / Permanent / Present / Native
CAND_ADDR1	Address Line1
CAND_ADDR2	Address Line2
CAND_CITY	City of the candidate
CAND_STATE	State
CAND_ZIP	Zip code of the candidate
CAND_TEL	Telephone Number of the candidate
CAND_MOB	Mobile Number of the candidate

Explanation:
The CAND_ADDR table records multiple contact information of the candidates.

Table Definition:

Table Name : **CAND_DOC**
Primary Key : CAND_DOC_NAME, CAND_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7		
CAND_DOC_NO	VARCHAR2	20		
CAND_DOC_NAME	VARCHAR2	16	NOT NULL	
CAND_DOC_TYPE	VARCHAR2	20		
CAND_DOC_ISSUE_DAT	VARCHAR2	12		
CAND_DOC_EXP_DAT	VARCHAR2	12		
CAND_DOC_ISSUE_PLACE	VARCHAR2	35		

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
CAND_DOC_NO	A unique passport / drivers license number
CAND_DOC_NAME	Document type: Passport (Passport) / Drivers License (Driving License)
CAND_DOC_TYPE	The type of the document, if any
CAND_DOC_ISSUE_DAT	Issue date for the document
CAND_DOC_EXP_DAT	Expiry date for the document
CAND_DOC_ISSUE_PLACE	Place of Issue for the document

Explanation:
The CAND_DOC table records the details of the documents possessed by the candidate.

Table Definition:

Table Name : **CAND_QUAL**
Primary Key : QUAL_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7		
QUAL_ID	VARCHAR2	7		
QUAL_TYPE	VARCHAR2	15	NOT NULL	
EXAM_NAME	VARCHAR2	50	NOT NULL	
INSTITUTION_NAME	VARCHAR2	50		
BOARD_NAME	VARCHAR2	50		
YEAR_OF_PASSING	VARCHAR2	10		
CLASS	VARCHAR2	20		
REMARKS	VARCHAR2	255		

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
QUAL_ID	A number for the qualification beginning with **CQ**, followed by 5 digits
QUAL_TYPE	Records the type of qualification (Academic / Professional / Technical)
EXAM_NAME	Name of the examination
INSTITUTION_NAME	Name of the institution
BOARD_NAME	Board/University attached to
YEAR_OF_PASSING	Year of Passing
CLASS	Class obtained in the examination
REMARKS	Any details user may wish to furnish

Explanation:
The CAND_QUAL table records the qualification details of the candidate.

Table Definition:

Table Name : **CAND_LANG**
Primary Key : LANG_ID, CAND_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
LANG_ID	NUMBER	2		
CAND_ID	VARCHAR2	7		
SPOKEN	VARCHAR2	5	NOT NULL	
WRITTEN	VARCHAR2	5	NOT NULL	

Table Description:

Column Name	Description
LANG_ID	Language number. Reference existing in LANGUAGE.LANGUAGE_ID
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
SPOKEN	Spoken ability of the candidate for a language (Good / Fair / Poor)
WRITTEN	Written ability of the candidate for a language (Good / Fair / Poor)

Explanation:
The CAND_LANG table records the languages know to a candidate.

Table Definition:
Table Name : **CAND_EMPL**
Primary Key : EMPL_ID
Foreign Key : - -

Column Definition:

Column Name	Data Type	Width	Allow Null	Default
CAND_ID	VARCHAR2	7		
EMPL_ID	VARCHAR2	7		
EMP_NAME	VARCHAR2	35		
EMP_LOC	VARCHAR2	35		
POS_HELD	VARCHAR2	35		
PER_FROM	VARCHAR2	12		
PER_TO	VARCHAR2	12		
SAL_ON_LEAVING	VARCHAR2	20		
CURRENCY	VARCHAR2	25		
REASON_FOR_LEAVING	VARCHAR2	255		
REMARKS	VARCHAR2	255		
IS_GAP	VARCHAR2	3		

Table Description:

Column Name	Description
CAND_ID	Candidate number. Reference existing in CANDIDATE.CAND_ID
EMPL_ID	A unique employment number beginning with CE, followed by 5 digits
EMP_NAME	Name of the employer
EMP_LOC	Employer Location

Table Description: (Continued)

Column Name	Description
POS_HELD	Position Held
PER_FROM	Period From
PER_TO	Period To
SAL_ON_LEAVING	Salary on Leaving
CURRENCY	Currency in which salary is received
REASON_FOR_LEAVING	Reason for Leaving
REMARKS	Any details user wishes to furnish
IS_GAP	Indicator to identify candidates with incomplete employment information

Explanation:
The CAND_EMPL table records the employment details of the candidates.

REMINDER

The above table description is similar to the one mentioned in Chapter 6: **Basic Interaction With SQL** under the topic **Choosing A Business Model to follow** with the exception that the column level constraints have been dropped. This is because the same has been handled at form level by the PMS application.

Environment Configuration For Personnel Management System

The **Personnel Management System** is developed using **HTML,** delivered to a client Browser, via **PHP** programs. The PHP programs access business data in real time from an **Oracle Database** run on a Linux machine.

When being created and tested, prior upload to the Internet, the system runs as an Intranet composed of:

- Apache web server which will serve the php pages
- A browser acting as the client which accesses these pages

Once tested and validated, this setup can be connected to any Internet pipeline for business data access without geographical boundaries.

The following steps will help configuring the environment:

- Copy the **PMS** folder (available in the book's accompanying CDROM) to **/var/www/sct**. Refer diagram 16.0.1

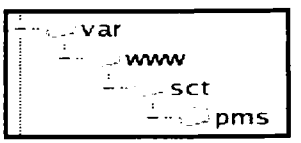


Diagram 16.0.1

- Make the following entries in the **httpd.conf** file available under **/usr/local/apache2 /conf** directory: Refer diagram 16.0.2

```
ServerName 192.168.0.3

NameVirtualHost 192.168.0.3

<VirtualHost 192.168.0.3>
   DocumentRoot /var/www/sct/pms
   ServerName sct.pms.com
</VirtualHost>
```

REMINDER

For the **ServerName**, **NameVirtualHost** And **VirtualHost** directives, enter the actual IP Address of the host computer .

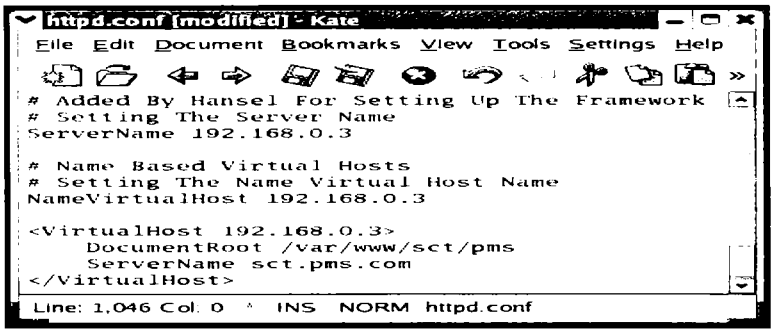


Diagram 16.0.2

- To allow a Web Browser to resolve the URL entered into its address bar make the following entry in the **hosts** file available under **/etc/hosts:** Refer diagram 16.0.3

```
<IP of the Linux machine Hosting the Site> [TAB] <Name of the Web Site>
```

Consider that the IP address of the Linux machine hosting the Site is **192.168.0.3** and the Web Site's URL is **sct.pms.com**, then the entry appended to the **hosts** file will be as follows:

```
192.168.0.3        sct.pms.com
```

The same entry should exist in the **hosts** file of each and every client machine desiring to access it via the Intranet (else its necessary to setup a DNS Server on the Intranet for Domain Name Resolution)

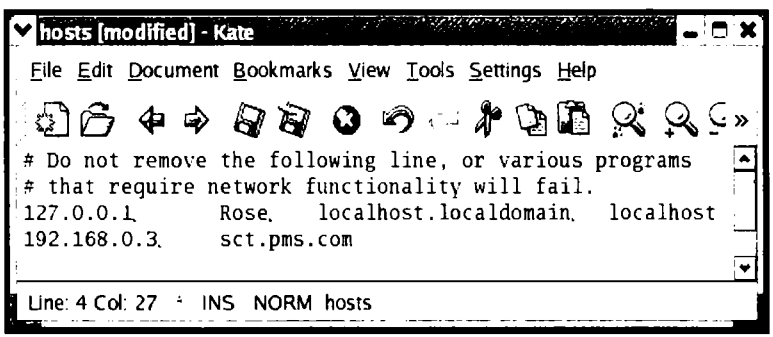


Diagram 16.0.3

- Edit the **php.ini** file available under **/usr/local/lib/** to allow registration of global variables. Global variables are extensively used by the **Personnel Management System**.

Registering Globals is a feature, which is usually **disabled** and hence needs to be activated by switching the **register_globals** to **ON**. Refer diagram 16.0.4

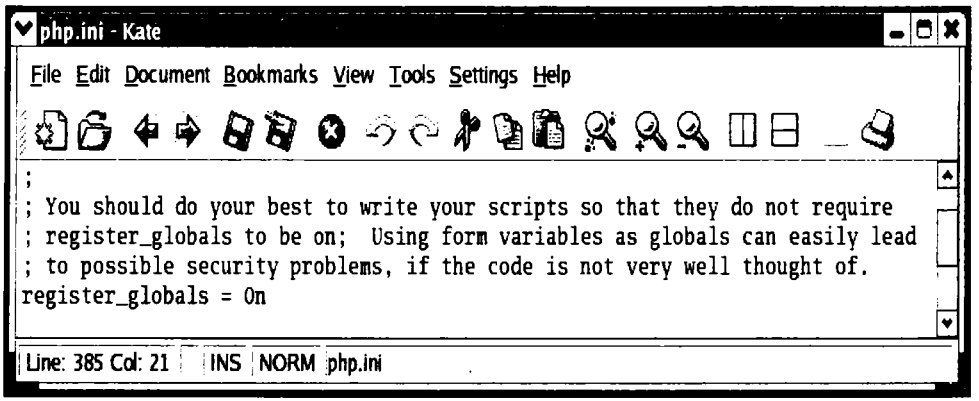


Diagram 16.0.4

- Since this is a web-based application, the following services should be active on the Linux machine hosting the Personnel management system:

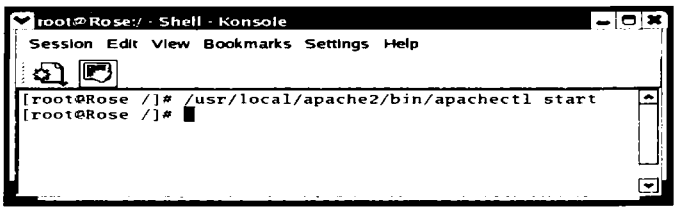


Diagram 16.1: Starting Apache

1. Apache **HTTP Server**, which hosts the web application. Refer diagram 16.1
2. Oracle 10g **Database Engine**, which handles management of data stored

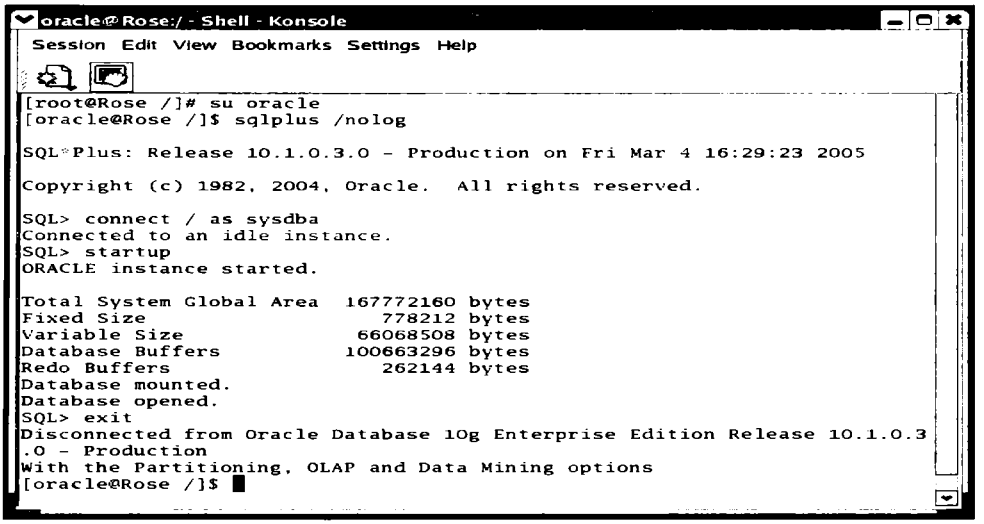


Diagram 16.2: The Oracle database starting up

3. Oracle 10g **Database Listener**, which responds to the calls made by the PHP pages at a pre-defined port (**e.g.** 1521) Refer diagram 16.3.1 and diagram 16.3.2

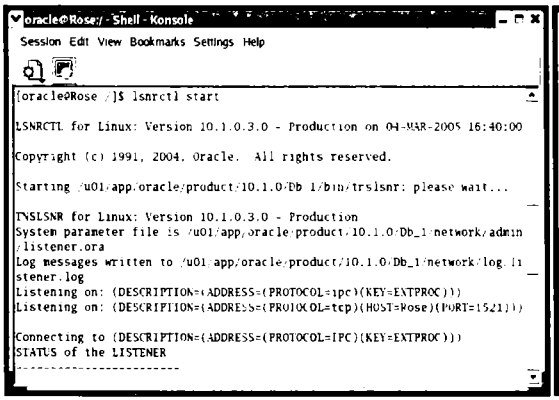


Diagram 16.3.1: Output of the **lsnrctl** command.

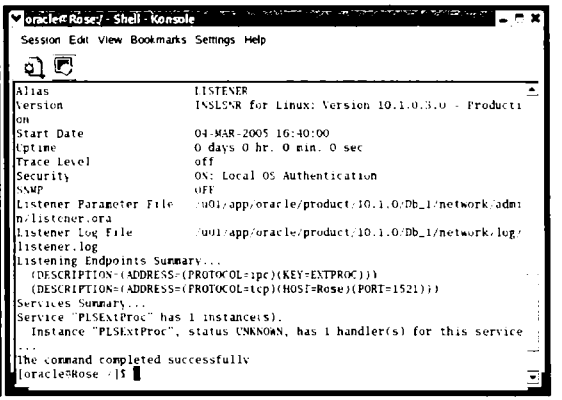


Diagram 16.3.2: The Listener started successfully.

- Login to the Oracle database using the SYS login via SQL *PLUS and create a tablespace named **PM_SYS** as:

```
<System Prompt> sqlplus sys/<password>@<Connection String> as SYSDBA
```

```
SQL> CREATE TABLESPACE PM_SYS
  DATAFILE 'PM_SYS.dat' SIZE 50M
  DEFAULT STORAGE(INITIAL 10K Next 50K
     MINEXTENTS 1 MAXEXTENTS 999 PCTINCREASE 10)
  ONLINE;
```

Create a user named DBA_PM as:

```
SQL> CREATE USER "DBA_PM"
  PROFILE "DEFAULT"
  IDENTIFIED BY "sct2306"
  DEFAULT TABLESPACE "PM_SYS"
  TEMPORARY TABLESPACE "TEMP"
  ACCOUNT UNLOCK;
```

Grant Permissions to the DBA_PM user as:

```
SQL> GRANT "DBA" TO "DBA_PM" WITH ADMIN OPTION;
```

Login as the DBA_PM user as:

```
SQL> sqlplus DBA_PM/<password>@<Connection String>
```

Execute the following file at the SQL prompt as:

```
SQL> @<Path_To_This_File>\PMS_Data.sql
```

This file will create the tables belonging to the Personnel Management System (Refer Diagram 16.0.0) and also populate the master tables such as **category**, **country**, **language**, **pk_generator** and **users** with test data for the Personnel Management System. The users table will only hold the admin user.

Alternatively, the **PMS_TestData.sql** can be executed at the SQL prompt. This will populate all the tables under the Personnel Management System with test data.

This **completes** the environment configuration for the Personnel Management System, which can now be started using a web browser and entering **http://sct.pms.com** into its address bar.

SECTION IV: PROJECT DEVELOPMENT USING ORACLE AND PHP

Personnel Management System Manual

Personnel Management System

This Personnel Management System is a web based, candidate data acquisition and placement application that essentially has three distinct sections.

The first section, called the **Candidate section**, collects specific candidate information via several HTML forms. After this information is collected, it is processed appropriately at the Server and the information keyed in by a candidate is segregated according to the job categories that a candidate has indicated they have skills in. Finally, this appropriately segregated data is stored in Oracle tables.

What's interesting here, is that if a candidate binds their skill set and resume to multiple job categories, then, multiple copies of the very same candidate data is **not stored** in various tables. Instead, only one copy of candidate information is stored in a table and flags are set in various other tables, which point to the same block of information that must be referenced. This approach, results in appreciable savings in hard disk space.

The second section of this application is the **Client section**. In this section, the candidate captured and stored earlier is made available to employers (**i.e. Clients** who have registered with the web site earlier). Occasionally, candidate information is also made available to other recruitment agencies.

Clients (and/or recruitment houses) that wish to have convenient access to a huge number of appropriately educated and experienced, prospective, employees have to pay a very nominal fee to access this service.

The third (and final) section of this application is an **Administrative section**. This section permits the maintenance of this application, such as the backup of table data, on occasion the updation, deletion and insertion of data, which can be done directly into the base tables via this section.

These three sections broadly cover the functionality of this candidate acquisition and placement application. Refer to diagram 17.0 for the framework on which the Personnel Management System will run. Having broadly, specified what the scope of this application, only the **Candidate section** and the **Administrative section** have been profiled in detail in this material.

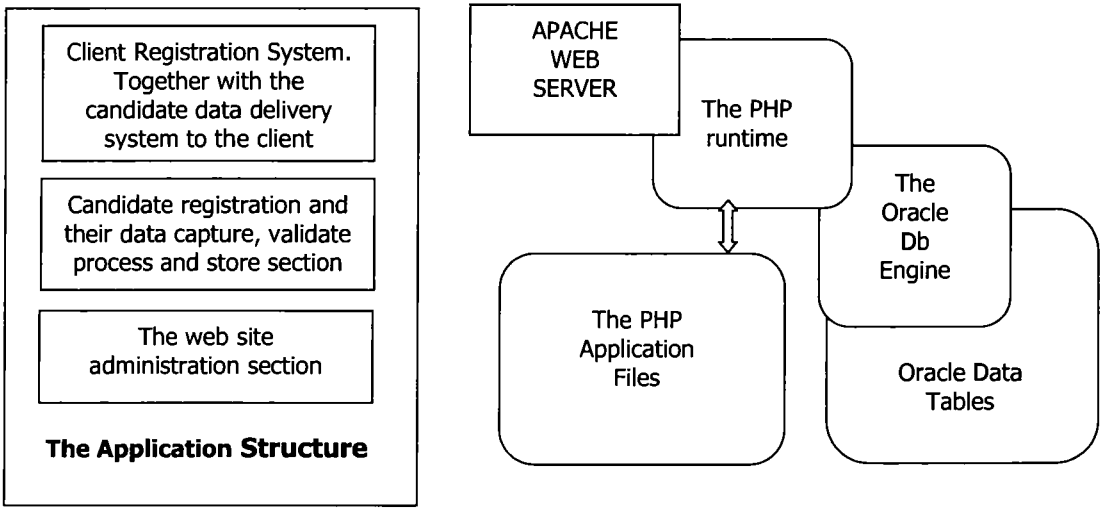


Diagram 17.0: Personnel Management System Framework

Getting Started

The Home Page

The application can be accessed using the following URL: **http://sct.pms.com**. as shown in diagram 17.1

This accesses the site's index page via which all its modules are accessed.

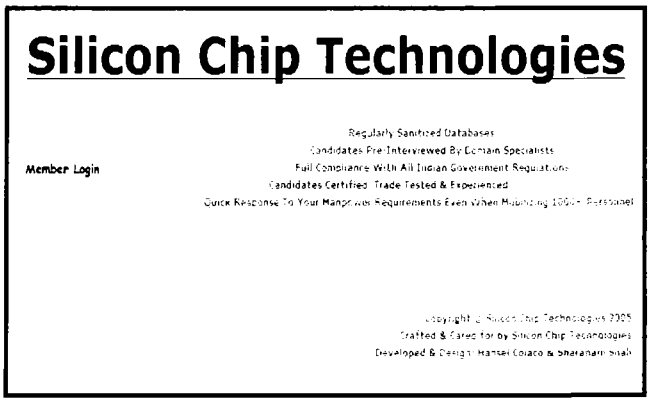


Diagram 17.1: The Home Page for the Web based Personnel Management System.

Given below are the Applications:

Requirements

- Design and host a website which provides information about the services provided by the company
- Any client of the company should be able to browse the website and gather information of the services offered, organizational structure and contact information
- Clients should be able to login to the site and view profiles of the candidates, which suit their requirement and setup interview dates with prospective candidates
- Candidates should be able to post their resumes and assign their resume content to more than one job category
- The site Administrator should be able to make all necessary changes to the database / tables and other Oracle resources in use
- The site Administrator should be able take backups and restore the Oracle database regularly and on demand

Sections

Based on the above requirements the website is divided into the following sections:

Clients Section

The client's section will have a form that will capture and store their staffing requirements on the web site. This form essentially captures details of the category, quantity and specifications of staff required, salary payable, benefits accruable to an employee and so on.

Candidates Section

The candidate's section will have a form that captures their personal, educational, professional and work experience. This information can in turn be bound to multiple job categories, which is then stored in the database tables.

Back Office Administrative

The Back office personnel (**i.e.** the Administration Section) have their own section to monitor the database, and to perform clean up/archiving/other operations. Neither clients nor candidates are able to access this section, which is password protected.

REMINDER

Candidates can login to the website and post their profiles free of cost.

Clients are expected to pay a nominal service charge for viewing candidate profiles. Therefore, the site has to be integrated with an e-payment mechanism so as to enable the clients to pay their service charges while online.

From the above requirements, it is only the **Candidate and Administration sections** that are documented and built in this material. Additionally, as part of the site's access mechanism, a **login system** has been built as a challenge / response system, prior actual access to the web site's resources.

The following is a user manual of this system being built.

The Home page briefly describes the purpose of the Web Site. A link labeled **Member Login** is available on the Home page. This link is used to access a default login page for this (Internet based) application.

The Administrator Module

In any application, there is the need to populate this applications lookup tables with information like Job Categories, Countries, Languages know and so on. The site owner always does this job, (**i.e.** this could also be the site administrator). Web Site administrator(s) require to periodically alter/insert/delete this kind of lookup information in the base tables and therefore a Web based, User interface, is required which permits this.

Candidates at the time of their registration (and/or submission of their resumes) use this information. They do this by selecting from dropdown list boxes, across forms, which offers specific information to select from. This helps reduce a great deal of data entry errors.

For obvious reasons access to this User interface (**i.e.** Administration) should not be made available to the public. The simplest solution, for achieving this, is to use a login system prior gaining access to the administrative UI, which is part of the Web Site. Access to the administrators **U**ser **I**nterface (UI) is only permitted on keying in the correct Login ID and Password combination via an administrator login interface.

The Administrative module will consist of web pages, images, access to master tables and so on, which is not to be accessed by Clients or candidates.

To achieve this the administrator module files are stored in a separate directory within the web site directory tree (the directory is named **administration**).

To access the components in this Module, the Web Site administrator(s) have to **extend** the Web site URL by specifying the name of the sub-directory holding this module's resources. For example: http://sct.pms.com/administration

Ensure that a user named admin is registered in the database.

To further increase security, a PHP based authentication page acts a challenge / Response system to its components. Refer diagram 17.2.1.

The Administrator's login page accepts a username and password. When Login is clicked (after entering a username and password), a server side PHP script authenticates the values entered. If the username and password belongs to a genuine administrator, the page provides access to information held in the Category table, as displayed. Refer diagram 17.3.1.

If the authentication fails the login page is re-displayed and access is denied.

The login page provides a link labeled **Help! I've forgotten my password!** which allows sending an email (using the default email application registered in the system in use). This feature intimates the website owner is an administrator has forgotten their password and a new one has to be assigned. Refer diagram 17.2.2.

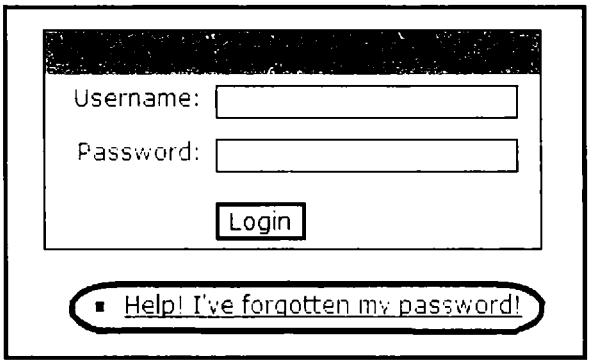


Diagram 17.2.1: The Login Page for the Web Site's administrator.

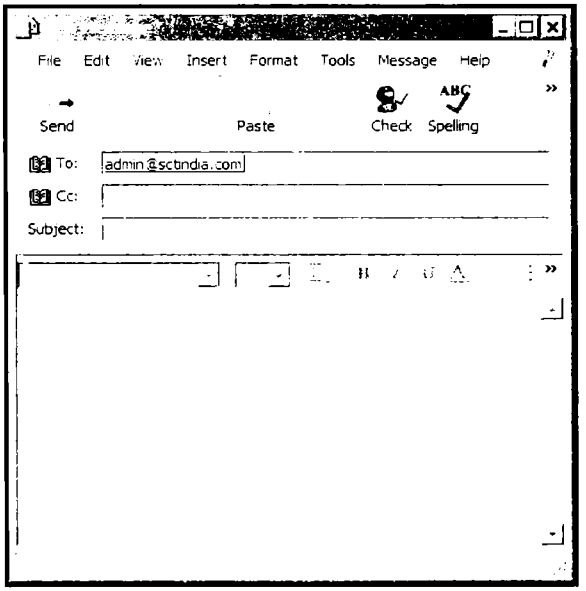


Diagram 17.2.2: Mailing application for the Web Site's administrator.

The Data Entry Forms Of The Administrative Module

When the administrative module is accessed the user interface to access information held in the **Category table (chosen as the default first page)** appears as shown in diagram 17.3.1.

The HTML's page header bar displays the name of the company owning the site and a link labeled LOGOUT. When **Logout** is clicked, the link will log out the currently authenticated administrator and return to the Administrator's Login page. The header bar remains consistent on all pages within the Administrator Module.

The topmost bar is followed by links, acting as tabs [illegible]. There are three tabs: Category, Country and Language. Each tab is a link to a data entry form bound to the table name maintained in the **Tabs** label.

The **Category form** is bound to the **Category table** of the Personnel Management System. This data entry form is used to manage information, held in the CATEGORY table, (as shown in diagram 17.3.1). It consists of two fields. The first field captures category name, while the second handles a remark (or a note) related to the category.

The command buttons [Save] and [Clear]. are visible which permit saving captured information or clearing form data from the data capture fields.

These control links are displayed on all data entry forms of the Administrator module.

Adding A New Category

The procedure for adding a category consists of entering the Category name and a remark or note related to the category. For example, store a category **Continental Cook** with the remark **Specialised in continental preparations** in the CATEGORY table simply enter the two values in their appropriate fields and click the [Save] button. Refer diagram 17.3.2.

When the [Save] button is clicked, the information entered is validated for **duplication** and/or **empty** values. Only valid information is stored in the CATEGORY table followed by emptying the data entry form fields for the next insertion of data, if any.

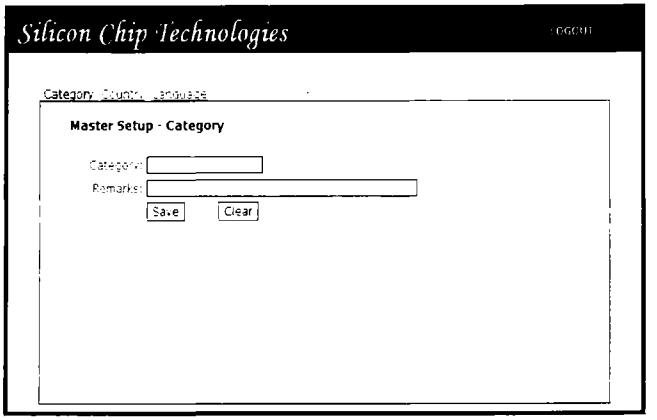


Diagram 17.3.1: The User Interface for accessing the CATEGORY table.

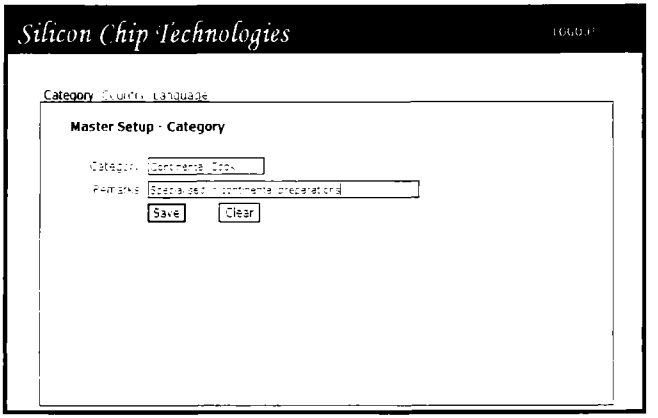


Diagram 17.3.2: Entering a new category.

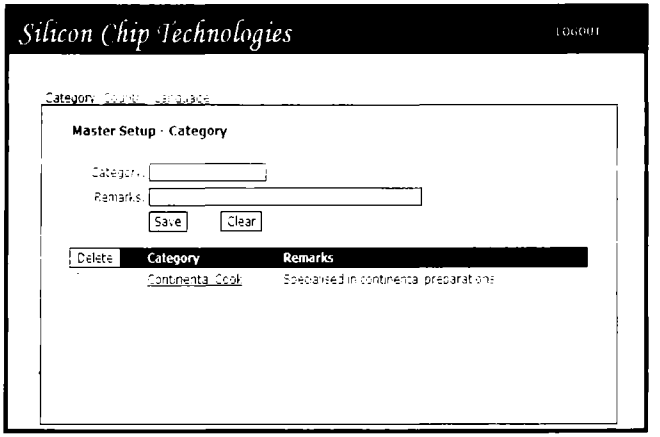


Diagram 17.3.3: Displaying the newly entered category.

Additionally, as soon as any information is entered via this form, it is displayed below the data capture section in a tabular format. Refer diagram 17.3.3.

Following the above steps several categories can be entered via this form and stored in database tables.

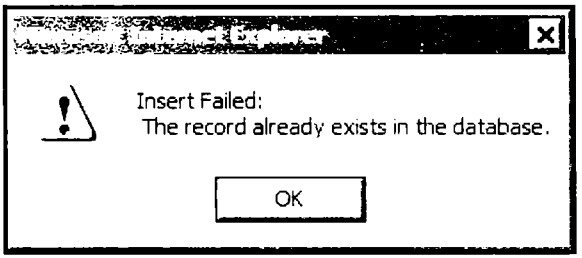


Diagram 17.3.4: Message for duplicate information.

In case an attempt is made to record the same category name twice, a data validation process identifies such an error. The information with a duplicate category name is ignored and the data entry form is cleared. In addition to this, a message is displayed which indicates that an attempt was made to store the same category name twice in the CATEGORY table. Refer diagram 17.3.4.

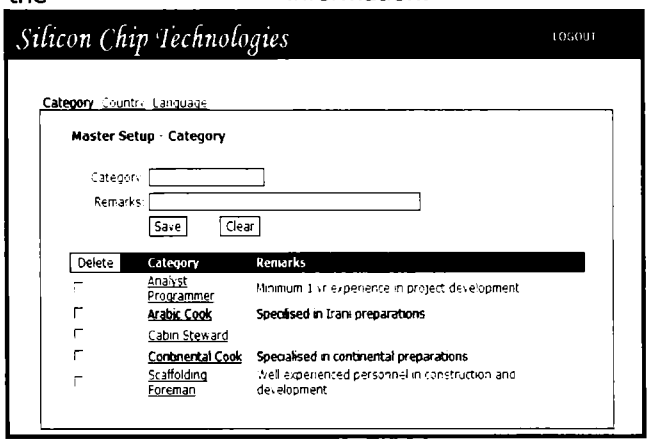


Diagram 17.3.5: Listing multiple categories.

When multiple categories have been stored in the database tables, the layout used to display this information, below the data entry form is capable of distinguishing **each entry** made. Alternate rows in the tabular layout have a different background color as shown in diagram 17.3.5. This improves the readability of information keyed in.

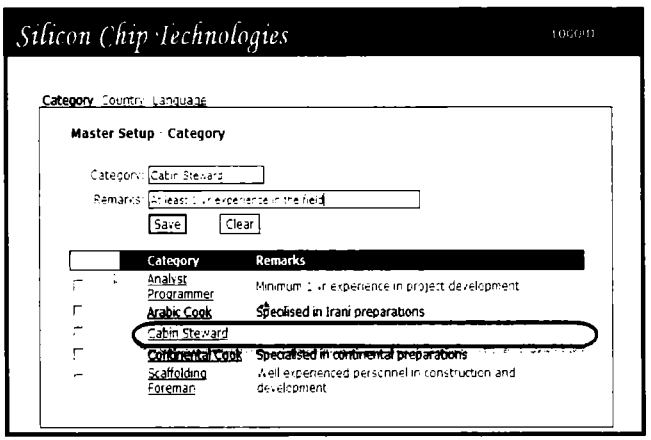


Diagram 17.3.6: Updating the selected entry.

Modifying Existing Categories

The first column of information (in this case category name) is a hyperlink allowing an entry to be selected for modification. When the hyper linked, category name is clicked, the fields in the data entry form are populated with information bound to that category. Refer diagram 17.3.6.

These fields can now be used to modify the information. Click Save to write this information back to the database table as well as display it in a tabular layout as explained earlier.

Click Clear to clear the form fields and return the page to the mode it was prior clicking this hyperlink.

Data validation is repeated during an update → save operation. Duplicate category names, while updating information is prevented and the form stays in the update mode.

Only valid information is stored into the database and the changes are reflected within tabular layout of the page. Refer diagram 17.3.7.

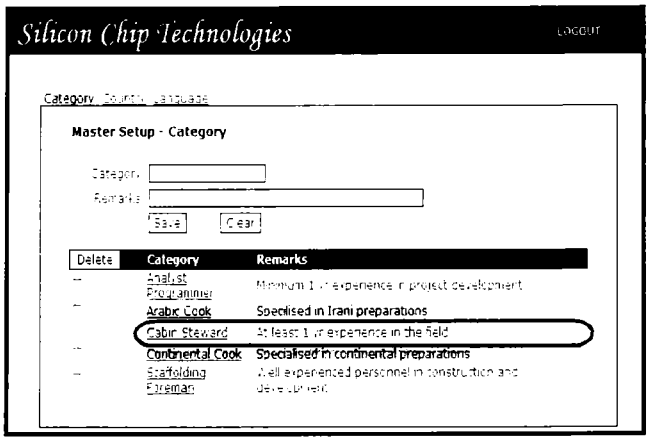


Diagram 17.3.7: Changes in the updated record displayed.

Deleting Existing Categories

The header section consists of a control link Delete **in addition** to the category name and remark column headers. For each entry recorded in the CATEGORY table the cell below Delete will display a checkbox. This checkbox is used to mark entries for deletion.

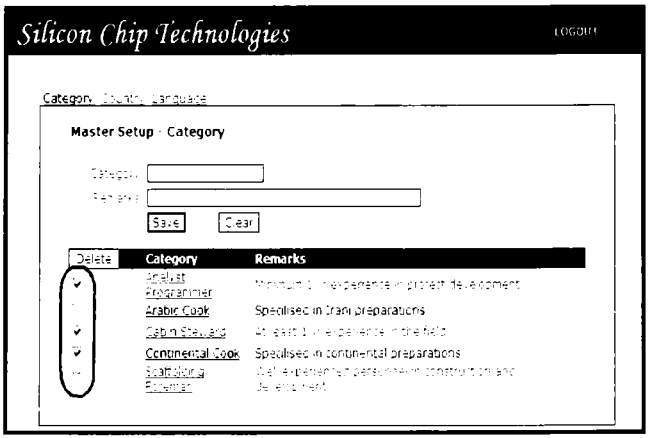


Diagram 17.3.8: Selecting multiple entries for deletion.

The user interface is designed to facilitate deletion of multiple entries in a single command. The selection of the checkbox to the extreme left of each entry marks a record for deletion. When the control link Delete is click the deletion process begins.

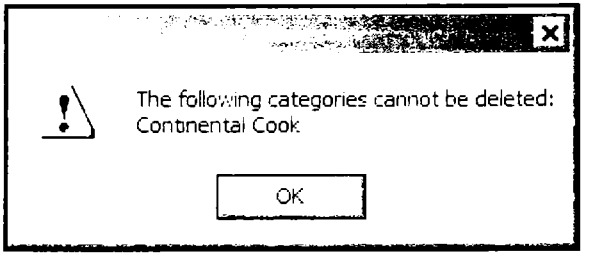


Diagram 17.3.9: Indicating the entries not deleted.

During the deletion process, a search is made for child records or dependencies bound to each entry marked for deletion. If such dependency exits in the other table(s), the deletion of that entry is skipped. Those entries that do not have any dependencies are deleted from the CATEGORY table.

Finally, a list of entries that have dependencies is generated and displayed in a message box indicating that deletion of these entries was not done due to the existence of child record. Refer diagram 17.3.9.

The updated list of entries existing in the database (after the delete operation) is displayed as shown in diagram 17.3.10.

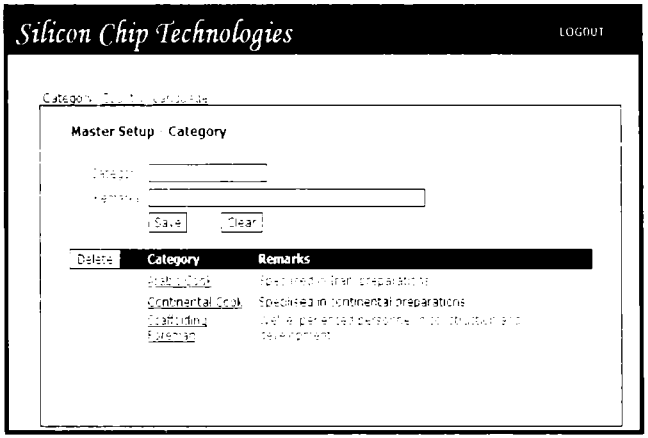


Diagram 17.3.10: List of entries after deletion.

Similarly the operations Delete, Update and Insert can be performed on the COUNTRY and LANGUAGE tables using their respective forms and an identical process as described above.

Click on the tab labeled **Country** to access the data entry form bound to the COUNTRY table. Diagrams 17.4.1 and 17.4.2 display the appearance of the page bound to the COUNTRY table.

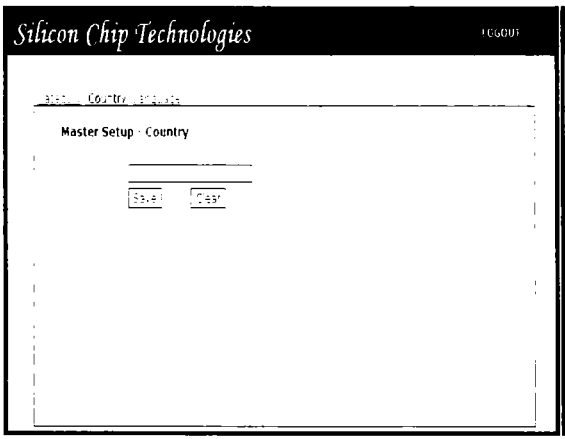


Diagram 17.4.1: The User Interface for accessing the COUNTRY table.

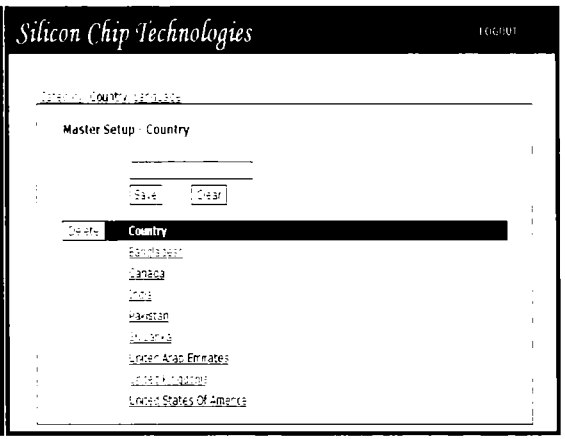


Diagram 17.4.2: The country form after performing inserts.

Click on the tab labeled **Language** to access the data entry form bound to the LANGUAGE table. Diagrams 17.4.3 and 17.4.4 display the appearance of the page bound to the LANGUAGE table.

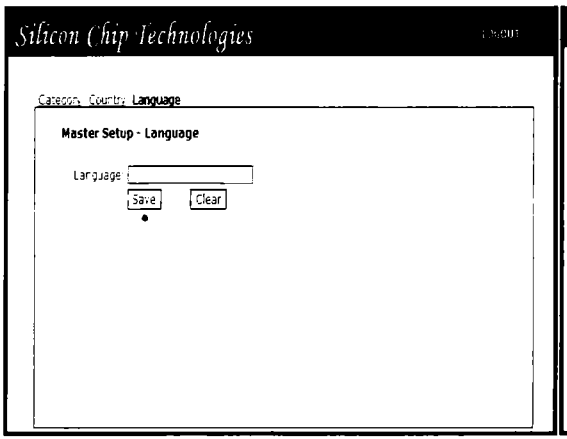


Diagram 17.4.3: The User Interface for accessing the LANGUAGE table.

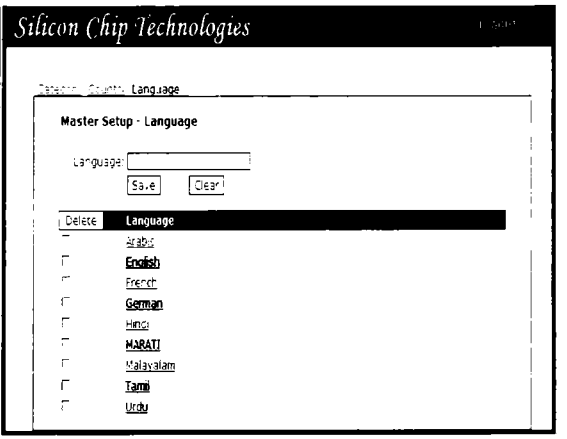


Diagram 17.4.4: The language form after performing inserts.

The Default Login Module

When a normal visitor accesses this application, the Web Site's Home page will appear. Refer diagram 17.5.1. The visitor needs to click the link labeled **Member Login** to access the Default Login page.

The Default Login Page appears as shown in diagram 17.5.1. The page allows capturing the visitor's login information (for visitors who have already registered with the system) or provides an option to register with the site.

The default login page accepts the username and password. When visitor clicks Login after entering the username and password, a PHP module authenticates the values entered. If the username and password belongs to a genuine visitor the PHP module determines the next page to be displayed based on whether the visitor is a **CANDIDATE** or a **CLIENT**.

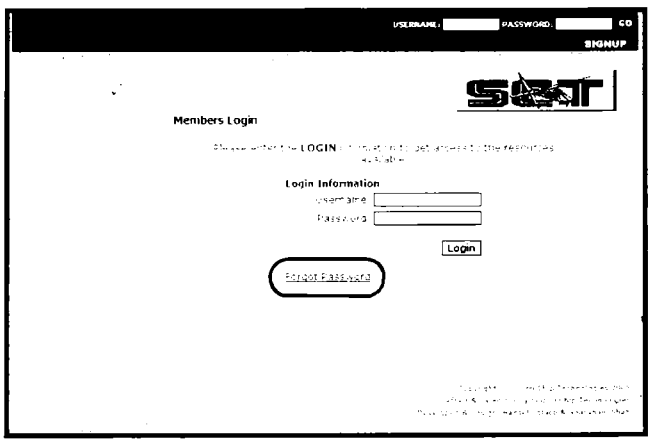


Diagram 17.5.1: The default login page accessible via the Member Login link on the Home page.

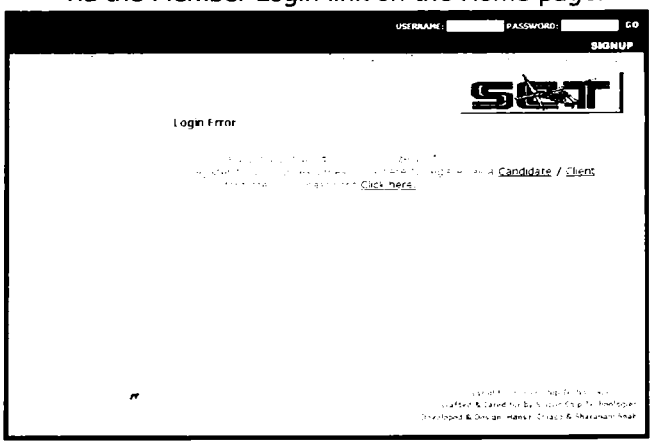


Diagram 17.5.2: The Login Error page.

If the authentication fails a **Login Error** page is displayed as shown in diagram 17.5.2. This page allows the following:

- Register as a Candidate, by clicking on the link labeled **Candidate**
- Register as a Client, by clicking on the link labeled **Client**
- Send E-Mail to the Web Site's Administrator when the visitor has forgotten their login information, by clicking on the link labeled **Click here**

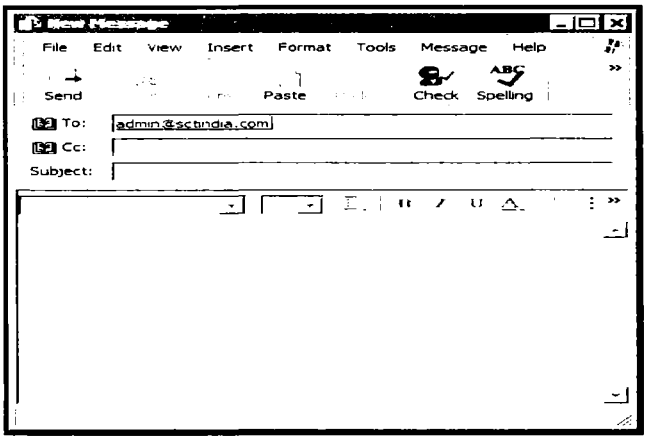


Diagram 17.5.3: Mailing application for the Web Site's administrator.

If the visitors have forgotten their login information, the login page provides a link labeled **Forgot Password**, which starts the default email application registered with the system.

Refer diagram 17.5.3.

Sign Up Options

The Member Login page displays a link, labeled **SIGNUP**, on the top right corner of the web page. Refer diagram 17.6.

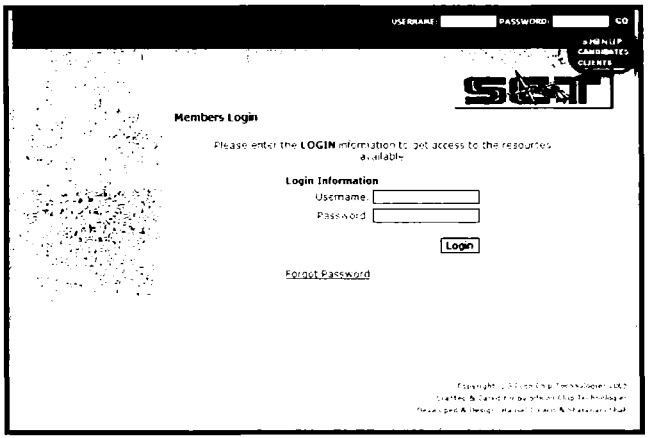


Diagram 17.6: The sign up options on the default login page.

This label acts as a menu item having two sub-menu options, namely:

- CANDIDATES
- CLIENTS

Each of the above options act as a link to startup a visitor's registration process, either as a candidate or a client.

The Candidate Module

The Candidate module is activated under two circumstances:

1. When visitors are authenticated as candidate during the login process
2. When visitors select the **CANDIDATES** link under the **SIGNUP** menu

The **CANDIDATE** link under the **SIGNUP** option allows a visitor to register as a **candidate**. As visitors, signing up as candidates, are generally seeking better employment opportunities, the registration process collects personal, educational, previous employments and other such relevant information from them for storage at the site. This is the information accessed by clients when they require specific types of employees.

Candidate Registration

On visitor signup as a candidate, the first page to appear is the Registration page. Refer diagram 17.7.

This page captures Login Information (username, password, hint question and answer) for the new candidate.

A PHP module processes the information captured when Register is clicked.

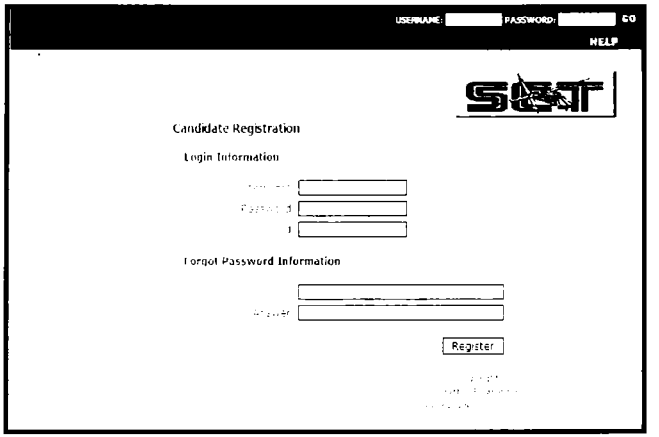


Diagram 17.7: The Candidate's Registration page.

The textboxes used for Username, Password and Retype Password are **mandatory** and cannot be left blank. The textboxes for Hint Question and Answer are **optional**.

This form has the capability to check for duplicate usernames. If a duplicate username is encountered the registration process is terminated and a message is displayed indicating this. Additionally some validations are performed to ensure that the same values are held in the Password and Retype Password textboxes. If the registration goes through then the personal information page is displayed in the client's browser to capture candidate information that would permit suitable employment at a client.

Personal Information

The second step in the Candidate's Sign Up process is the Personal Information page. Refer diagram 17.8.

The information collected on this page includes:

- Job category in which the candidate is seeking employment
- Personal information like Candidate's Name, religion, birthday, birthplace, email address and so on
- Postal information, captured as present address and permanent address
- Passport details, if any
- Driving License details, if any

A PHP module processes the information entered into the textboxes, when Proceed is clicked. The textboxes for Job Category, Candidate's name, father's name, birthday and email address are mandatory and cannot be left blank.

The **first** address textbox, city, pin code and state in the present postal address section are mandatory. Filling the permanent postal address section is optional.

The sections for Passport Details and Driving License Details do not accept **partial** information. This means that if a document number is entered, the remaining information becomes mandatory.

A PHP module validates the data captured by the Personal Information page. If missing or invalid information is encountered, the Personal Information page is re-rendered and a message displayed indicating the error.

If no errors are encountered then control passes to the Candidate's Qualification Details page. Refer diagram 17.9.1.

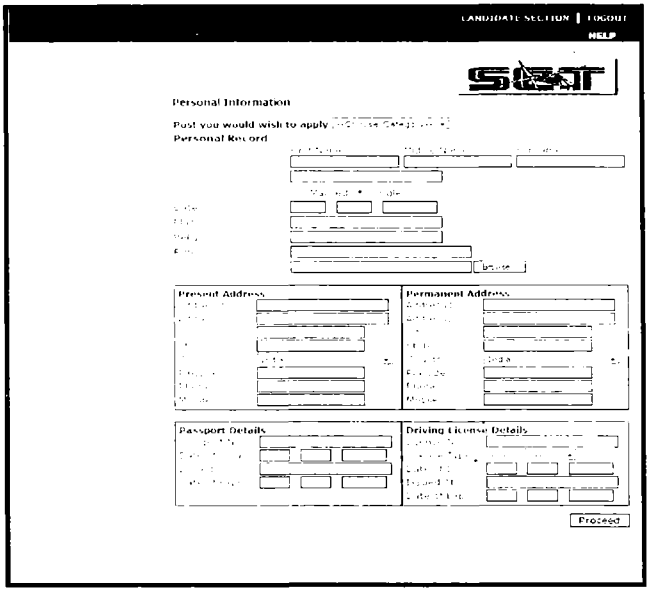


Diagram 17.8: The Candidate's Personal Information page.

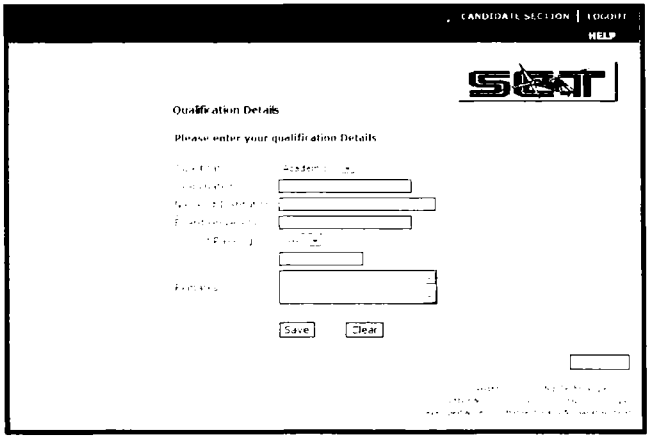


Diagram 17.9.1: The Candidate's Qualification Details page.

Qualification Details

The third step in a Candidate's Sign Up process is the Qualification Details page. Refer diagram 17.9.1.

Since this page is bound to the CAND_QUAL table, the information collected on this page includes:

- Qualification type: Academic, Professional or Technical
- Name of the Examination, Institution, Board, University
- Year of passing, class obtained and remark

When the Qualification Details page appears, it has textboxes that capture a candidate's qualifications. A pair of command buttons labeled Save and Clear are displayed in the row immediately following the textboxes.

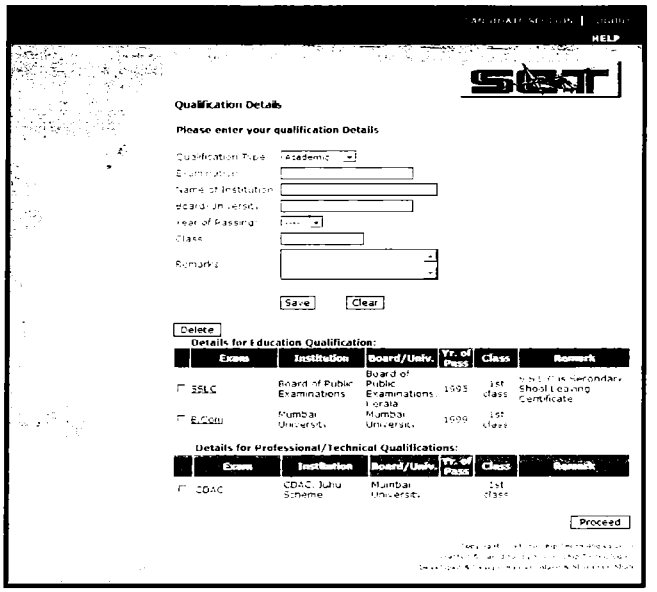


Diagram 17.9.2: Displaying candidate's qualification information.

The command button labeled Save calls a PHP module to validate the information captured and appropriately store it in the database, while the command button labeled Clear is used to clear the textboxes.

A third command button labeled Proceed is visible but **inactive**. Refer diagram 17.9.1. This button is activated (Refer diagram 17.9.2.) only after a candidate submits at least one set of qualification details.

REMINDER

The insert, update and delete operations **can be performed** using the steps described earlier in the CATEGORY data entry form under the Administrative module.

To submit the qualification details click Save after the textboxes are populated with appropriate information. A PHP module validates the information captured and also scans the database for entries belonging to the candidate to detect duplication.

Only a valid set of candidate qualification details is stored in the database and immediately displayed in a tabular layout when the page is redisplayed in the browser. Refer diagram 17.9.2.

Any invalid qualification details are rejected and a message indicating the error is displayed on the Qualification Details page when displayed.

While displaying the Qualification Details page, the CAND_QUAL table is scanned for entries bound to the candidate's qualification information entered earlier. If found this is displayed below the data entry form and [Proceed] is **activated**. If no qualification details are found the [Proceed] remains **inactive**.

When [Proceed] is active and clicked, a call to the Candidate's Employment Details page is made.

Employment Details

The forth step in the Candidate's Sign Up process is the Employment Details page. Refer diagram 17.10.1.

Since this page is bound to the CAND_EMPL table, the information collected on this page includes:

- Name and location of the previous employer
- Designation, duration and payment terms from the previous employments
- Additional information (If any)

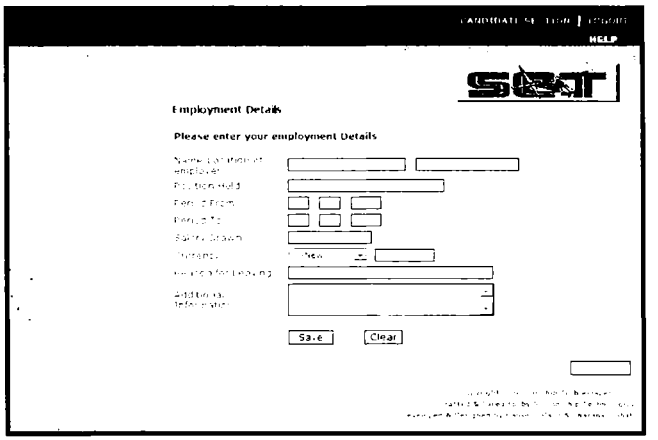


Diagram 17.10.1: The Candidate's Employment Details page.

The Employment Details page has a layout, similar to the Qualification Details page.

The command button labeled [Save] calls a PHP module to validate the information captured and store it in the database, while the command button labeled [Clear] is used to clear the form fields.

A third command button labeled [Proceed] is visible but **inactive**. This button is made **active** only after the candidate submits at least **one** set of previous employment details.
The **Reason for Leaving** and the **Additional Information** are optional and the rest of the fields are mandatory.

REMINDER

The insert, update and delete operations can be performed using the steps described earlier in the CATEGORY data entry form under the Administrative module.

To submit a set of previous employment details the forms textboxes are populated with appropriate information and Save button is clicked. A PHP module validates the information captured and also scans the database for entries belonging to the candidate to detect duplication.

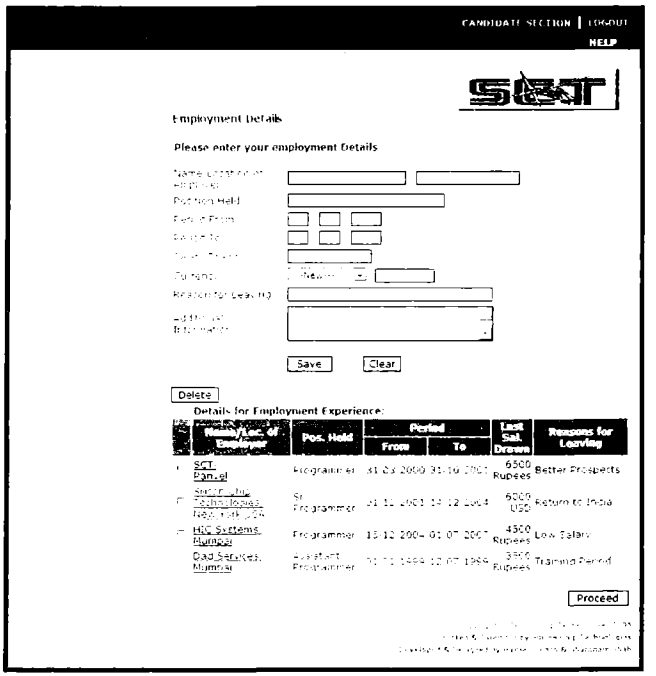


Diagram 17.10.2: Displaying candidate's previous employment information.

Only a valid set of candidate employment details is stored into the database and then displayed in tabular layout when the page is redisplayed. Refer diagram 17.10.2.

An invalid set of candidate employment details is rejected and a message indicating the error is displayed on the Employment Details page when it is redisplayed.

While redisplaying the Employment Details page, the CAND_EMPL table is scanned for entries bound to the candidate's previous employment information. If found it is displayed below the data entry form and Proceed is **activated**. If none are found Proceed remains **inactive**.

When Proceed is active and clicked, a call to the Candidate's Language Proficiency page is made.

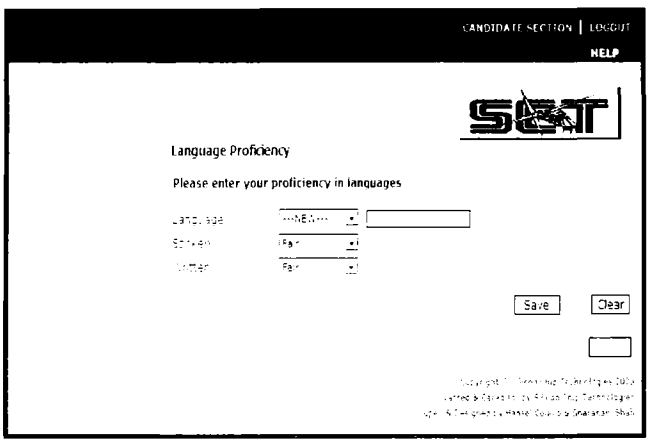


Diagram 17.11.1: The Candidate's Language Proficiency page.

Language Proficiency

The fifth step in the Candidate's Sign Up process is the Language Proficiency page. Refer diagram 17.11.1.

Since this page is bound to the CAND_LANG table, the information collected on this page includes:

- Language name
- Candidate's grade in oral and written communication

The layout of the Language Proficiency page is similar to pages for Qualification Details and Employment Details seen earlier. The only difference is that Proceed is replaced by Finish since this is the **last step** in a candidate registration process.

The command button labeled Save calls a PHP module to validate the information captured and store it in the database. Clear is used to clear the textboxes in the form.

A third command button labeled Finish is visible but **inactive**. This button is activated only after a candidate submits at least **one** set of details bound to languages known.

REMINDER

The insert, update and delete operations can be performed using the steps described earlier as in the CATEGORY data entry form under the Administrative module.

To submit a set of candidate's language skills the textboxes are populated with appropriate information and Save is clicked. A PHP module validates the information captured and also scans the database for entries belonging to the candidate to detect duplication.

Only a valid set of candidate language skills information is stored in the database and then displayed in tabular layout when the page is redisplayed. Refer diagram 17.11.2.

An invalid set of candidate language skills is rejected and a message indicating the error is displayed on the Language Proficiency page when redisplayed.

When the Language Proficiency page is redisplayed, the CAND_EMPL table is scanned for entries bound to the candidate's language skills information. If found it is displayed below the data entry form and Finish is **activated**. If nothing is found Finish remains **inactive**.

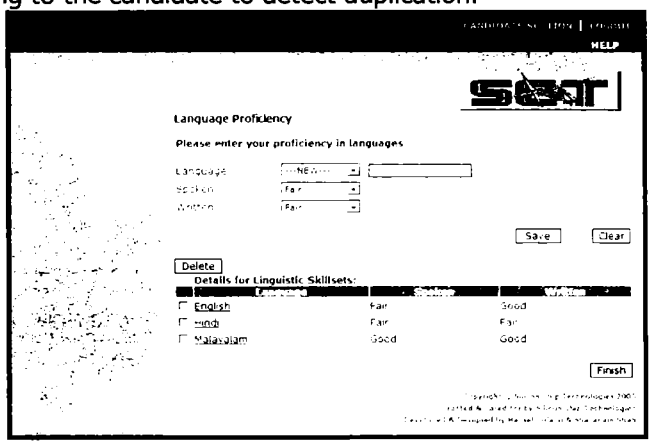


Diagram 17.11.2: Displaying candidate's skills in oral and written communication.

When Finish is clicked, a call to the Candidate's Home page is made.

Candidate's Home Page

A candidate's home page is displayed only when the sign up process for the candidate is **completed**.

This page welcomes the visitor to the Candidate Section and asks for a login, which is used to access the system **along with the day, date and time of login**.

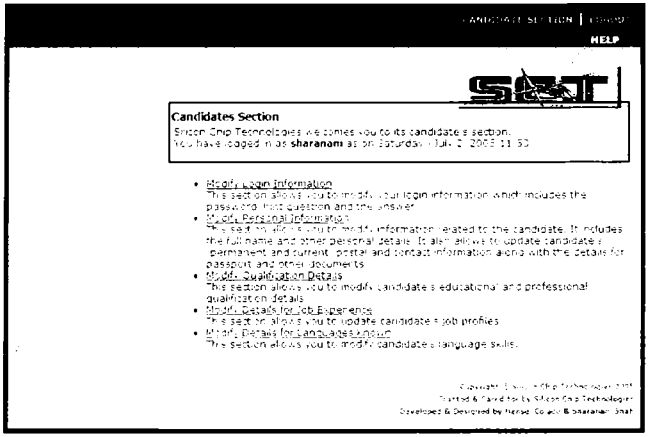


Diagram 17.12: The Candidate's Home page.

The page also provides links to update candidate information (based on the login). The links are as follows:

- **Modify Login Information:** This section allows the candidate to modify their login information, (i.e. password, hint question and answer **only**)
- **Modify Personal Information:** This section allows the candidate to modify information such as name and other personal details, changes in postal address(es), details of passport and driving license
- **Modify Qualification Details:** This section allows the candidate to modify educational and professional qualification details
- **Modify Details for Job Experience:** The section allows the candidate to update their employment profile
- **Modify Details for Languages Known:** This section allows the candidate to update their language skills

Modifying Login Information

If while accessing the Candidate's Home page, the link labeled Modify Login Information is clicked the Candidate Registration page is displayed in the **update mode**. Refer diagram 17.13.1.

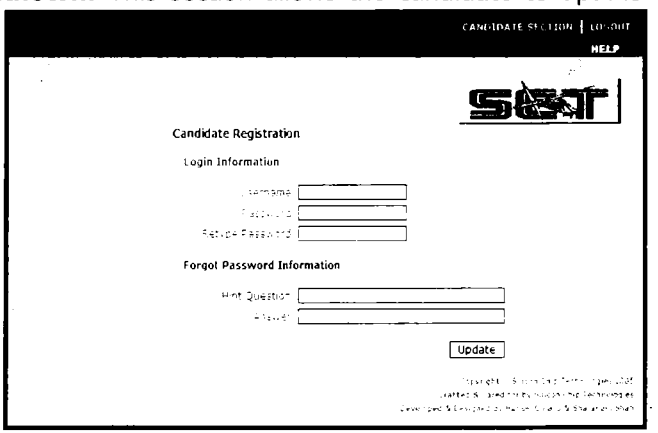


Diagram 17.13.1: The Modified Candidate Registration page.

This page has an Update button which when clicked updates the data captured by the form to the database. Additionally, the first row of the page describes the currently active module and provides a link to LOGOUT. Apart for the above differences, the behavior of the Candidate Registration page remains the same.

When Update is clicked a validation process is carried out on the data captured by the form to take care of the following:

- The field Username displays the login name which cannot be modified, (**i.e.** the field is disabled)
- The fields Password and Retype Password are mandatory and cannot be left blank
- The fields Hint Question and Answer are optional

In addition to validating mandatory fields the PHP module invoked does the following:

- Compares values held in the Password and Retype Password field. If a difference is found, the Registration page is re-rendered with a message indicating that the two values are different
- If information entered is valid, the updates are recorded

Modifying Personal Information

The Personal Information page is displayed when the link labeled Modify Personal Information is clicked.

When the Personal Information page appears in the update mode, the textboxes are populated with existing data. Refer diagram 17.13.2.

The left hand side of the page displays links to access other pages found under the Candidate Sign Up section. These links are:

- **Modify Qualification Details:** Invokes the Qualification Details page in the update mode
- **Modify Experience Details:** Invokes the Employment Details page in the update mode
- **Modify Languages Details:** Invokes the Language Proficiency page in the update mode

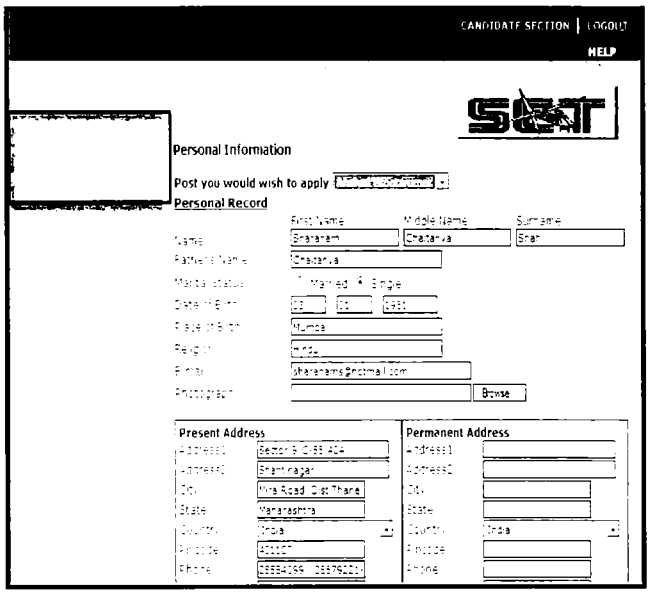


Diagram 17.13.2: The modified Candidate's Personal Information page.

Modifying Qualification Details

The Qualification Details page is displayed when the link labeled Modify Qualification Details is clicked.

When this page appears in the update mode, the existing data is displayed in a tabular layout (under the data entry form). Refer diagram 17.13.3.

The left hand side of the page displays links to access other pages found under the Candidate Sign Up section. These links are:

- **Modify Personal Info.:** Invokes the Personal Information page in the update mode
- **Modify Experience Details:** Invokes the Employment Details page in the update mode

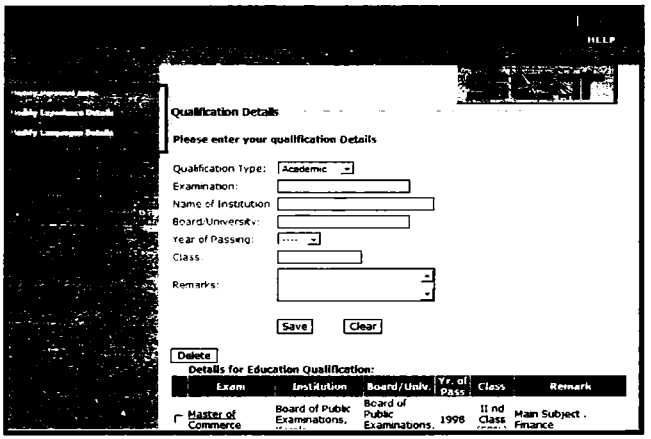


Diagram 17.13.3: The Modified Candidate's Qualification Details page.

- **Modify Languages Details:** Invokes the Language Proficiency page in the update mode

Modifying Details for Job Experience

The Employment Details page is displayed when the link labeled Modify Details for Job Experience is clicked.

When this page appears in the update mode, the existing data is displayed in a tabular layout (under the data entry form). Refer diagram 17.13.4.

The left hand side of the page displays links to access other pages found under the Candidate Sign Up section. These links are:

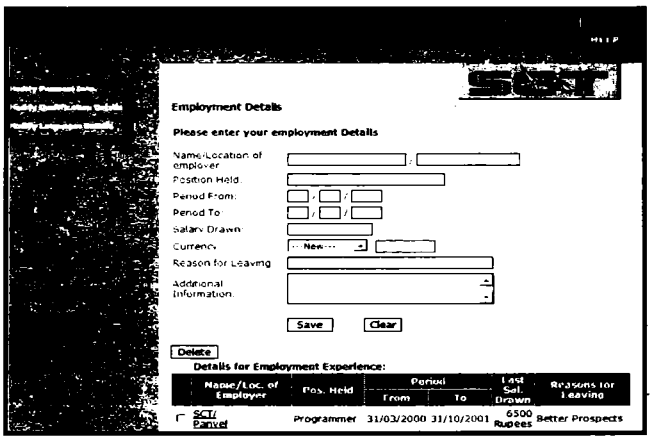


Diagram 17.13.4: The Modified Candidate's Employment Details page.

- **Modify Personal Info.:** Invokes the Personal Information page in the update mode
- **Modify Qualification Details:** Invokes the Qualification Details page in the update mode
- **Modify Languages Details:** Invokes the Language Proficiency page in the update mode

Modifying Details for Languages Known

The Language Proficiency page is displayed when the link labeled Modify Details for Languages Known is clicked.

When this page appears, in the update mode, the existing data is displayed in a tabular layout (under the data entry form). Refer diagram 17.13.5.

The left hand side of the page displays links to access other pages found under the Candidate Sign Up section. These links are:

- **Modify Qualification Details:** Invokes the modified Qualification Details page
- **Modify Experience Details:** Invokes the modified Employment Details page
- **Modify Personal Info.:** Invokes the modifies Personal Information page

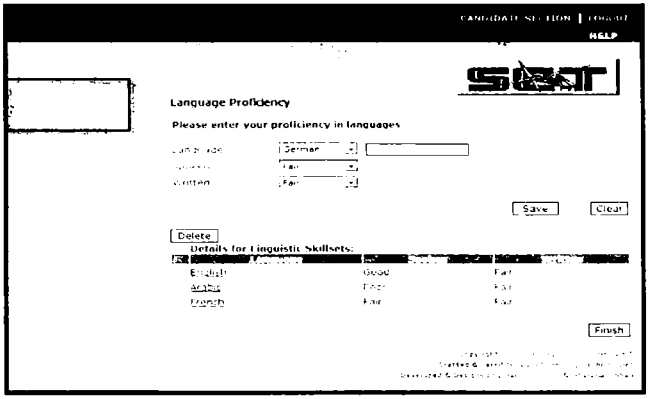


Diagram 17.13.5: The Modified Candidate's Qualification Details page.

Incomplete Resume

The visitor accesses the Default Login page via the Home page link labeled **Member Login** a page as shown in diagram 17.5.1 appears. When visitor clicks Login after entering the username and password, a PHP module authenticates the values entered. If the username and password belongs to a genuine candidate, the PHP module verifies whether the visitor has submitted all required candidate information in earlier visits.

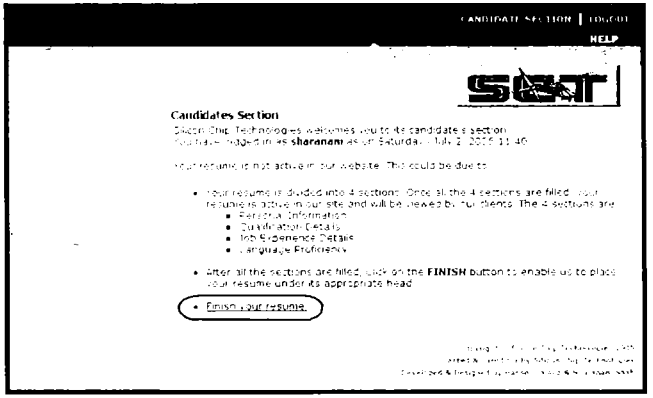


Diagram 17.14: The modified Candidate's Home page.

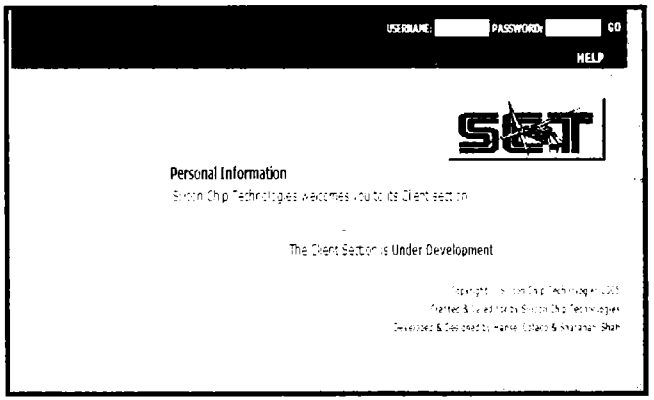


Diagram 17.15: The Client's Home page.

When the database indicates that the required candidate information has been collected, the PHP module displays the Candidate's Home page as shown in diagram 17.12. If the database indicates that particular set of information is **not collected** which means the resume is **incomplete**, the PHP module displays a page as seen in diagram 17.14.

The Candidate's Home page provides a link labeled **Finish your resume**, which when clicked invokes the appropriate page for capturing the remaining candidate information.

Client Login

When a visitor accesses the system as a client, a page as shown in diagram 17.15 is displayed.

This page welcomes the client but **indicates that the Client section is under development**.

Help Pages

Each module (**i.e.** Candidate and Client) in the system has individual HELP pages. These pages are accessible by clicking HELP (placed at the top right hand corner of the pages). Refer diagram 17.16.1 and 17.16.2.

When Help is clicked, the help page corresponding to the active module is displayed within a new browser window. Refer diagram 17.16.3 and diagram 17.16.4.

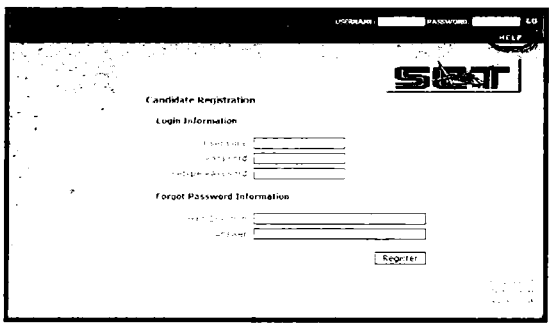


Diagram 17.16.1: Link for the Help page in the Candidate section.

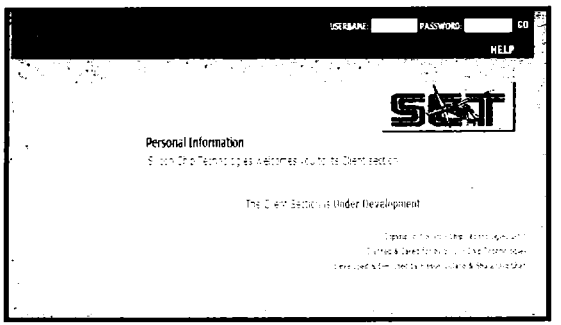


Diagram 17.16.2: Link for the Help page in the Client section.

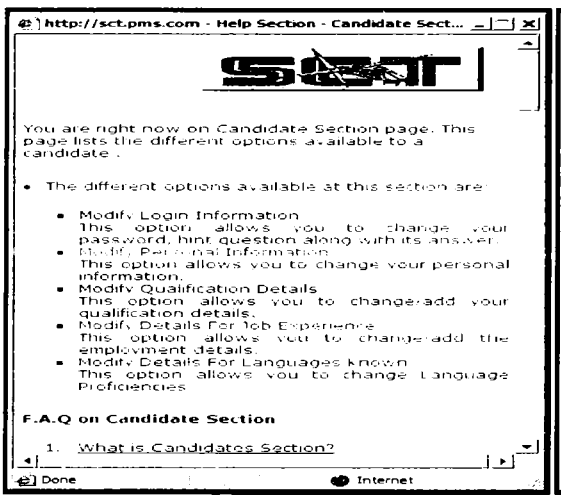


Diagram 17.16.3: The Help page for the Candidate section.

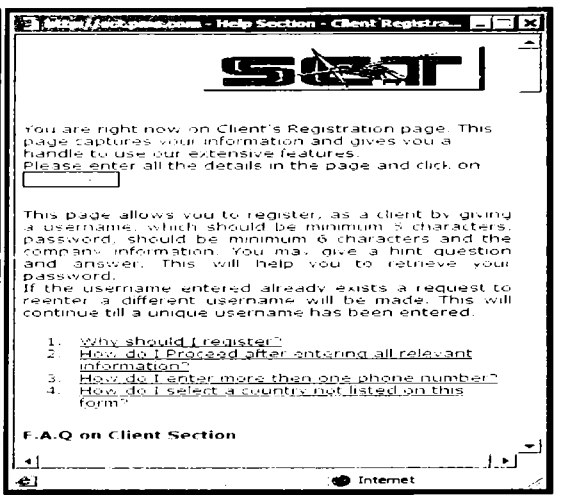


Diagram 17.16.4: The Help page for the Client section.

SECTION IV: PROJECT DEVELOPMENT USING ORACLE AND PHP

Project Processing For Personnel Management System

The **Personnel Management System** is developed using **HTML,** delivered to a client Browser, via **PHP** programs. The PHP programs access business data in real time from an **Oracle Database** run on a Linux machine.

The system runs as an Intranet composed of:

- Apache web server which will serve the php pages
- A browser acting as the client accessing these pages

Once tested and validated, this setup can be connected to any Internet pipeline for business data access without geographical boundaries.

Since this is a web-based application, the following services should be active on the Linux machine hosting the Personnel management system:

1. Apache **HTTP Server**, which hosts the web application
2. Oracle 10g **Database Engine**, which handles management of data stored
3. Oracle 10g **Database Listener**, which responds to the calls made by the PHP pages via a pre-defined port (e.g. 1521)

REMINDER

The entire source-code is provided on the Book's accompanying CDROM.

The Personnel Management System

The purpose of the Personnel Management System is to act as a link between employees and a prospective employer. It needs to collect information from prospective employees (CANDIDATES) and make them available to prospective employers (CLIENTS) on the basis of specific criteria.

The PMS consists of three modules:

- Employees (CANDIDATES)
- Employers (CLIENTS)
- The web site managers (ADMINISTRATION)

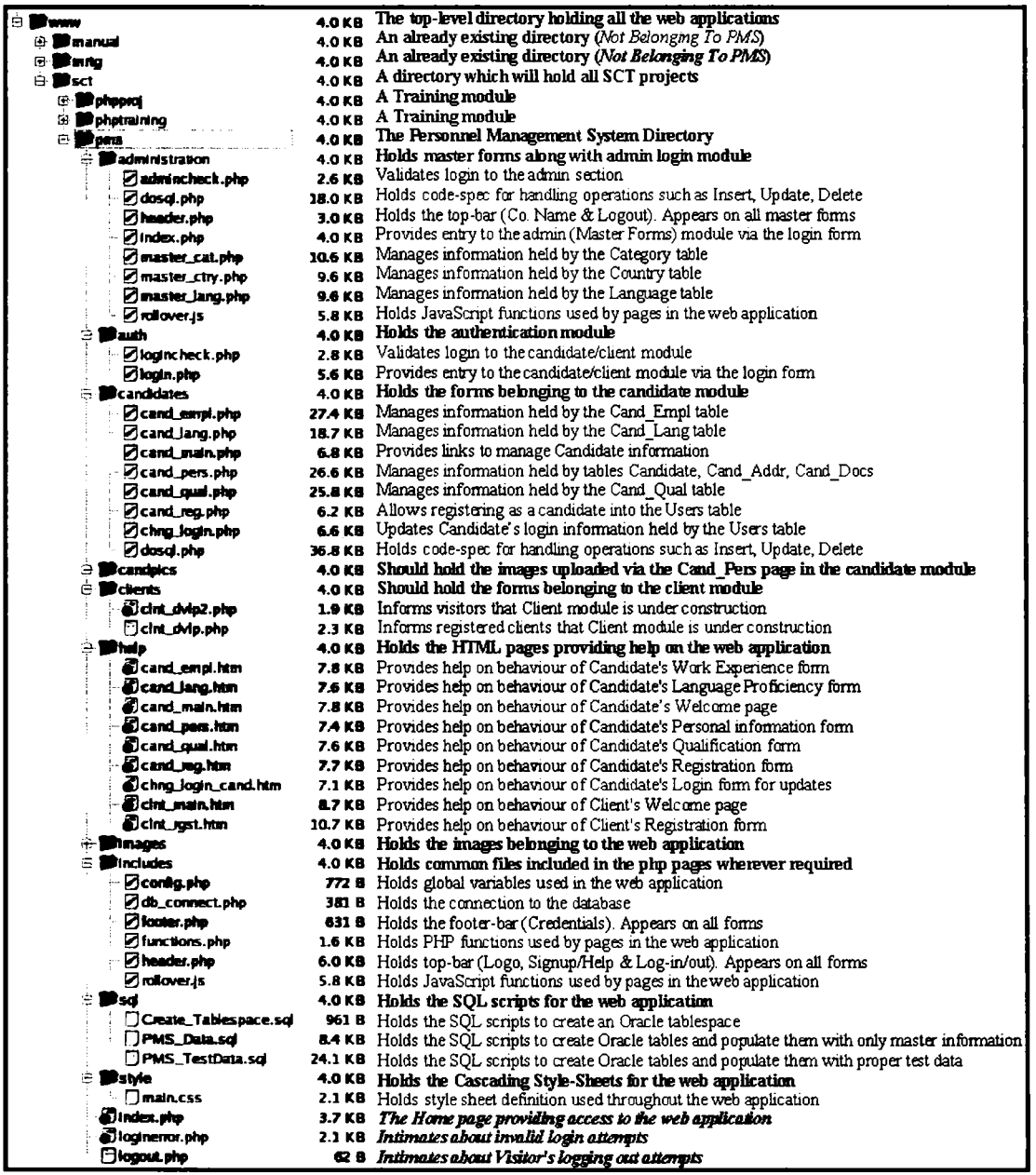
The **CANDIDATE** module handles candidate registration along with collection and management of candidate information. The **CLIENT** module handles client registration along with collection and management of client information and manpower requirements.

Users of this application can either be clients or candidates. They are provided access to a module (Client/Candidate) only after they register with the website. Registration procedure for both the modules is independent. The GENERAL LOGIN system authenticates any Login prior access to either of the modules.

For any application to run smoothly some information related to managing the application always needs to be entered or updated. This information helps maintenance of look up information usually stored in MASTER tables. This application comprises of three master tables. They hold information bound to countries, languages and the available job categories.

The management of MASTER table information is handled by the system administrator(s), via the administration module and is **not available** to either candidates or clients. The **ADMINISTRATION** module is independent of the main system. This module has its own independent authentication system.

The application comprises of files such as the .php, .html, images, .css and the help files. These files are stored on the Linux server in a structure as seen in diagram 18.0.



www	4.0 KB	**The top-level directory holding all the web applications**
manual	4.0 KB	**An already existing directory** *(Not Belongmg To PMS)*
mrng	4.0 KB	**An already existing directory** ***(Not Belonging To PMS)***
sct	4.0 KB	**A directory which will hold all SCT projects**
phpproj	4.0 KB	**A Training module**
phptraining	4.0 KB	**A Training module**
pms	4.0 KB	**The Personnel Management System Directory**
administration	4.0 KB	**Holds master forms along with admin login module**
admincheck.php	2.6 KB	Validates login to the admin section
dosql.php	18.0 KB	Holds code-spec for handling operations such as Insert, Update, Delete
header.php	3.0 KB	Holds the top-bar (Co. Name & Logout). Appears on all master forms
index.php	4.0 KB	Provides entry to the admin (Master Forms) module via the login form
master_cat.php	10.6 KB	Manages information held by the Category table
master_ctry.php	9.6 KB	Manages information held by the Country table
master_lang.php	9.6 KB	Manages information held by the Language table
rollover.js	5.8 KB	Holds JavaScript functions used by pages in the web application
auth	4.0 KB	**Holds the authentication module**
logincheck.php	2.8 KB	Validates login to the candidate/client module
login.php	5.6 KB	Provides entry to the candidate/client module via the login form
candidates	4.0 KB	**Holds the forms belonging to the candidate module**
cand_empl.php	27.4 KB	Manages information held by the Cand_Empl table
cand_lang.php	18.7 KB	Manages information held by the Cand_Lang table
cand_main.php	6.8 KB	Provides links to manage Candidate information
cand_pers.php	26.6 KB	Manages information held by tables Candidate, Cand_Addr, Cand_Docs
cand_qual.php	25.8 KB	Manages information held by the Cand_Qual table
cand_reg.php	6.2 KB	Allows registering as a candidate into the Users table
chng_login.php	6.6 KB	Updates Candidate's login information held by the Users table
dosql.php	36.8 KB	Holds code-spec for handling operations such as Insert, Update, Delete
candpics	4.0 KB	**Should hold the images uploaded via the Cand_Pers page in the candidate module**
clients	4.0 KB	**Should hold the forms belonging to the client module**
clnt_dvlp2.php	1.9 KB	Informs visitors that Client module is under construction
clnt_dvlp.php	2.3 KB	Informs registered clients that Client module is under construction
help	4.0 KB	**Holds the HTML pages providing help on the web application**
cand_empl.htm	7.8 KB	Provides help on behaviour of Candidate's Work Experience form
cand_lang.htm	7.6 KB	Provides help on behaviour of Candidate's Language Proficiency form
cand_main.htm	7.8 KB	Provides help on behaviour of Candidate's Welcome page
cand_pers.htm	7.4 KB	Provides help on behaviour of Candidate's Personal information form
cand_qual.htm	7.6 KB	Provides help on behaviour of Candidate's Qualification form
cand_reg.htm	7.7 KB	Provides help on behaviour of Candidate's Registration form
chng_login_cand.htm	7.1 KB	Provides help on behaviour of Candidate's Login form for updates
clnt_main.htm	8.7 KB	Provides help on behaviour of Client's Welcome page
clnt_rgst.htm	10.7 KB	Provides help on behaviour of Client's Registration form
images	4.0 KB	**Holds the images belonging to the web application**
includes	4.0 KB	**Holds common files included in the php pages wherever required**
config.php	772 B	Holds global variables used in the web application
db_connect.php	381 B	Holds the connection to the database
footer.php	631 B	Holds the footer-bar (Credentials). Appears on all forms
functions.php	1.6 KB	Holds PHP functions used by pages in the web application
header.php	6.0 KB	Holds top-bar (Logo, Signup/Help & Log-in/out). Appears on all forms
rollover.js	5.8 KB	Holds JavaScript functions used by pages in the web application
sql	4.0 KB	**Holds the SQL scripts for the web application**
Create_Tablespace.sql	961 B	Holds the SQL scripts to create an Oracle tablespace
PMS_Data.sql	8.4 KB	Holds the SQL scripts to create Oracle tables and populate them with only master information
PMS_TestData.sql	24.1 KB	Holds the SQL scripts to create Oracle tables and populate them with proper test data
style	4.0 KB	**Holds the Cascading Style-Sheets for the web application**
main.css	2.1 KB	Holds style sheet definition used throughout the web application
index.php	3.7 KB	***The Home page providing access to the web application***
loginerror.php	2.1 KB	***Intimates about invalid login attempts***
logout.php	62 B	***Intimates about Visitor's logging out attempts***

Diagram 18.0: The Directory Structure Of The **P**ersonnel **M**anagement **S**ystem

The files belonging to this application are stored in a modular fashion. This means that all files belonging to the candidates module are stored under the CANDIDATES folder. Similarly all files belonging to the clients module are stored in the CLIENTS folder and so on.

The Entity Relationship - Personnel Management System

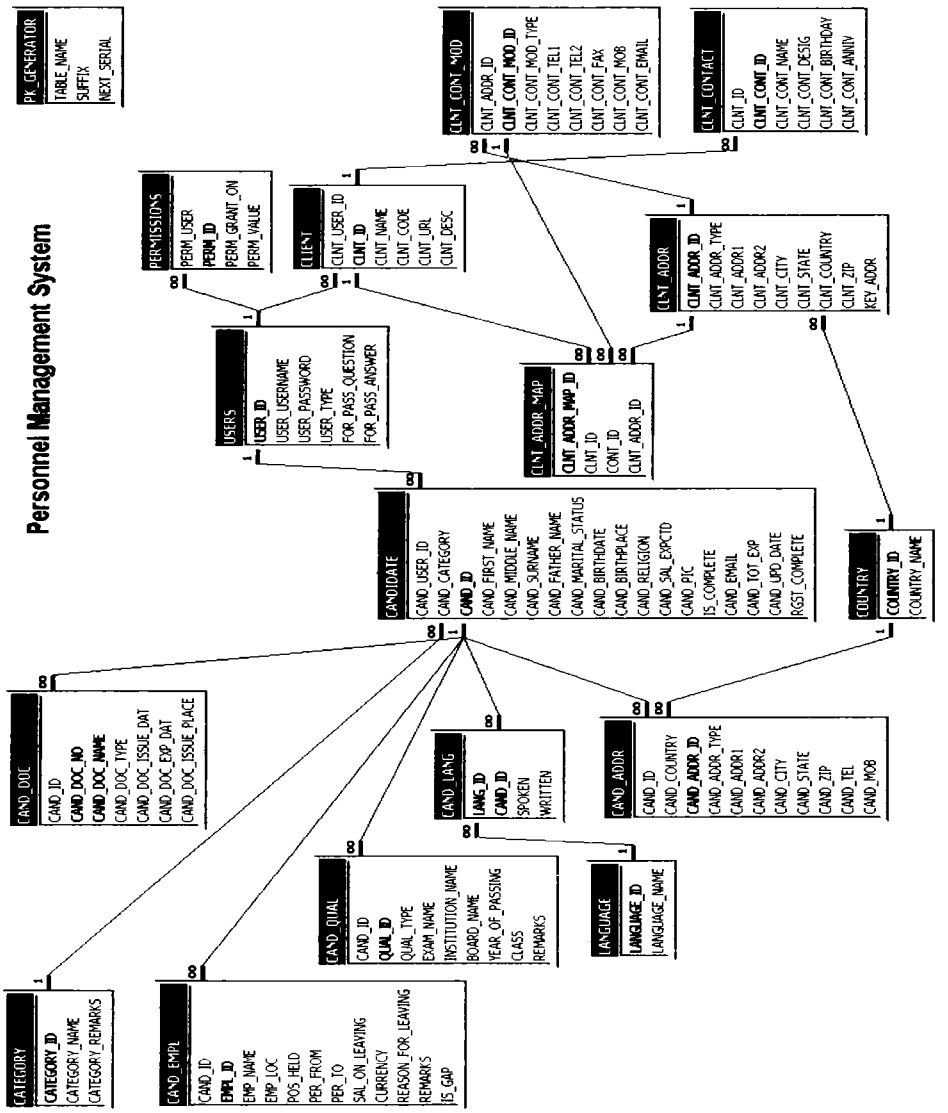


Diagram 18.0.1

Diagram 18.0.1 provides a visual representation of **E**ntity **R**elationships **i.e.** the relationships between the tables used to store business data of the Personnel Management System.

Site Map – Personnel Management System

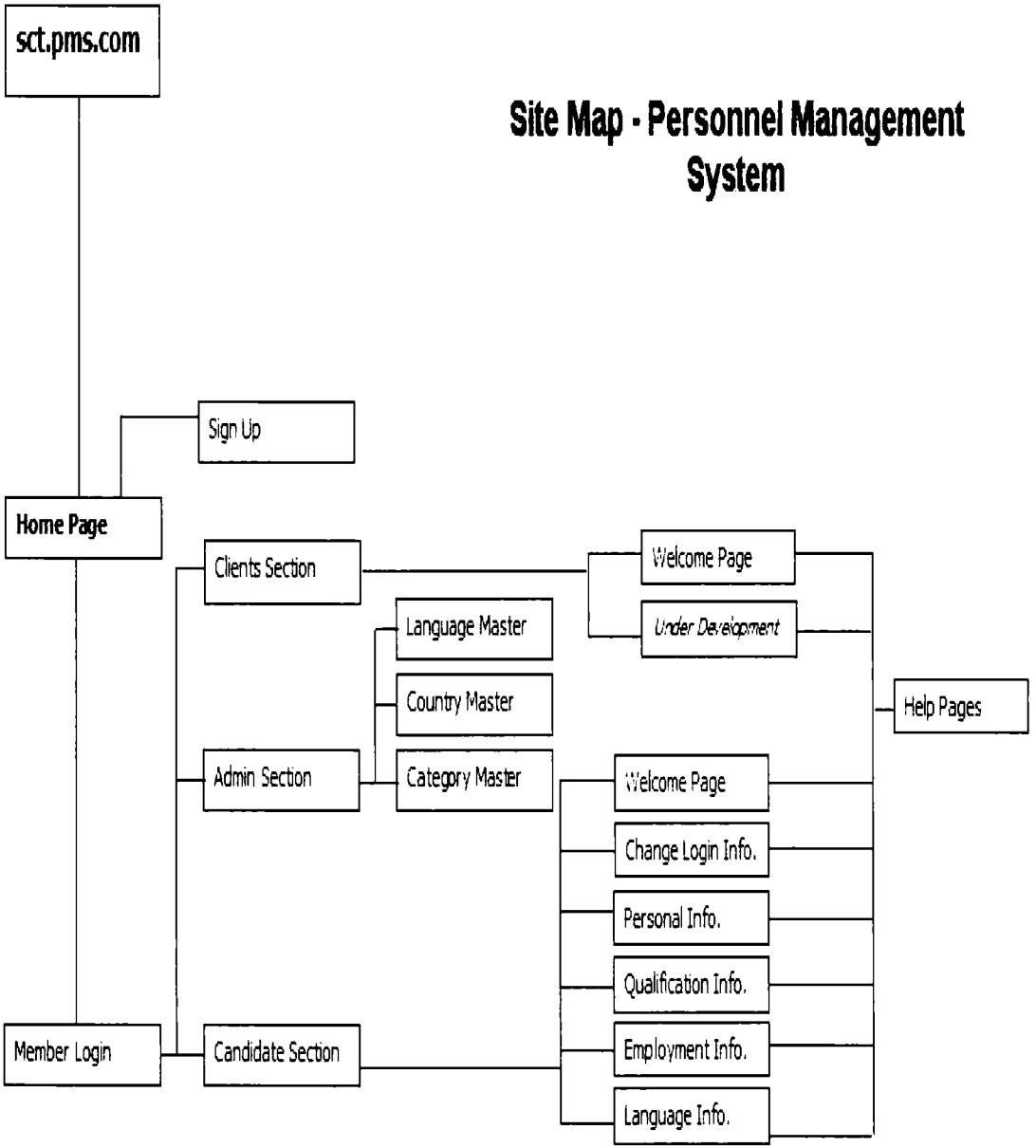


Diagram 18.0.2

Common Standards For Application Documentation Via Comments

Every php file in this application has the following comments at the beginning of the file.

```
/*
  Date :- 03/01/05
  Author :- Sharanam Shah
  Filename :- index.php
  Purpose :- Admin Login Page. The user submits the login
  information from this page.
*/
```

These comments simply indicate the following:

- The date when the file was created
- The author of the file
- The file name for identification
- The purpose of the file in terms of the operations allowed

Contents Of The Includes Folder

PHP allows the inclusion code in a module that is contained in another file by using the **include** keyword. This feature allows effortlessly recycling useful bits of code without having to resort to COPY AND PASTE the code in each PHP module. Additionally, this means that if some changes are desired in a particular piece of code, rather than having to track down every individual module where this code block exists and make changes, simply make changes **only** in a single file that holds the code that needs recoding. Since this file is included in other programs the changes will be reflected automatically.

The PMS application extensively uses the INCLUDES folder for accessing such independent code blocks.

The **includes** folder contains the following files: (Refer diagram 18.0)

1. config.php

This file initializes a **few** variables used by the web application throughout. These variables are:

- **dbuser** holds the user name bound to the PMS application i.e. **dba_pm**
- **dbpass** holds the **<password>** bound to the user **dba_pm**
- **company_name** holds the company name who owns the PMS application i.e. **Silicon Chip Technologies**
- **photourl** holds the path for uploading candidate's pictures, i.e. **/var/www/sct/pms/candpics**

- **title** holds the page title which will be used in the <TITLE> tag of every **php/html** page in this application
- Background colours for rows of tabular data displayed in master pages. Refer diagram 18.1.1
  - **gl_mstr_top_bar_color** holds the colour code for the header row, i.e. '**#400040**'
  - **gl_mstr_frst_bar_color** holds the colour code for the first row, i.e. '**#E4E4E4**'
  - **gl_mstr_scnd_bar_color** holds the colour code for the second row, i.e. '**#C0C0C0**'
- Background colour for rows of tabular data displayed in general pages. Refer diagram 18.1.2
  - **gl_top_bar_color** holds the colour code for the header row, i.e. '**#878676**'
  - **gl_first_bar_color** holds the colour code for the first row, i.e. '**#F4EFE3**'
  - **gl_second_bar_color** holds the colour code for the second row, i.e. '**#FFFFFF**'

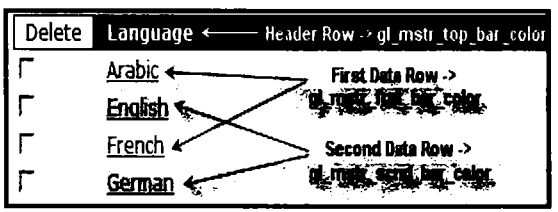


Diagram 18.1.1

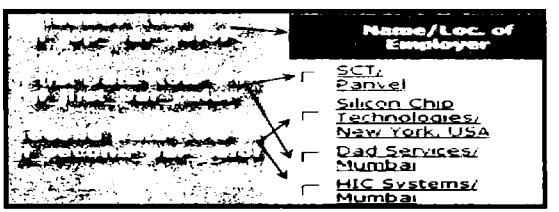


Diagram 18.1.2

The **config.php** file is used / included in all PHP files belonging to this project.

2. db_connect.php

This file contains the codespec that actually connects to the database. A PHP function named **ocilogon()** is used to perform the task of connecting to Oracle. The function accepts two parameters:

- **dbuser** – the user name existing under oracle
- **dbpass** – the password for this oracle user's login

The values for the above parameters are dynamically derived from the **config.php** file. On successful connection to the oracle database, a database handle is returned to a variable named **rcq**. This variable can be used to perform database operations across web pages.

If there is a failure in establishing the database connection, a message is generated and displayed. The message requests that the administrator be contacted and also provides a link to send an E-mail to the web site's administrator.

3. rollover.js

This file defines JavaScript functions, which are made available when included within a page with HTML code. Two separate files, having similar functionality are maintained under the following locations:

login_over signup_over help_over

Diagram 18.2

- /includes/rollover.js(used by Candidate/Client modules)
- /administration/rollover.js(used by Administration module)

This is done to bring functional independence in terms of code spec. The functions defined in this file are:

- **preloadimages** – Initializes the variables login_over, signup_over, help_over as objects of the image type by calling the **newImage()** function. Refer diagram 18.2.
- **newImage** – Creates objects of the image type based on the path and filename passed as a parameter
- **glow** – Creates a glow effect by interchanging images. This means a replacement take place with new image passed as parameter. Called on the **mouseOver** event of the image
- **unglow** – Removes the glow effect by interchanging images. This means a replacement take place with original image passed as parameter. Called on the **mouseOut** event of the image
- **chkNum** – Validates whether the content of the form field passed as parameter to this function is a number
- **checkDate** – Validates whether the content of the form fields, used for capturing a date holds valid information
- **strltrim** – Returns the string passed as a parameter after removing blank spaces, if any, at the beginning of the parameter
- **strrtrim** – Returns the string passed as a parameter after removing blank spaces, if any, at the end of the parameter
- **strtrim** – Returns the string passed as a parameter after removing blank spaces, if any, at both the beginning and the end of the parameter.
  The functions **strltrim, strrtrim, strtrim** *used for trimming string values are given aliases* **ltrim, rtrim, trim** *respectively.*
- **chngCase** – This function changes the first alphabet to an upper case character. This operation is performed on the value passed as a parameter to this function. The parameter accepted is in following format:
  - Document.FormName.FieldName.Value

  This function is called when a form cursor moves out of a form field.

- **formDeleteValues** – Generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same. This function is called when the DELETE button is clicked.
- **openHelpWin** – Opens a new browser window having the title as **HelpWindow**. It accepts the parameter value as the URL. *This function is used to open help pages in a separate window*
- **ChngLen** – Converts a single digit number to a two digit (i.e. 1 to 01). *This function is applied to date fields to implement uniformity*

4. header.php

This page generates the top bar consisting of two rows. The behavior of the top bar changes dynamically, depending upon the page in which it (header.php file) is included. This page has the following processes:

- A session is initialized using the PHP function **session_start()**
- A check is made if the visitor is logging-out. This means that the visitor has clicked Logout and the control has been passed to a page, which includes the header.php file. This is done as:
  - A variable **mode** holds the state in terms of **login** or **logout**.
  - If the variable **mode** holds the value **logout**, then any existing session variables bound to previous login are unregistered and re-initialised
- The following two files are included:
  - /includes/config.php
    - This file is included to make the variables declared available in the header.php
  - /includes/db_connect.php
    - This file is included to establish a connection with the database and thus make available the variable **rcq** which is the database handle
- A variable named **style_sheet** is initialised to hold the value **main.css**. This is used while embedding the style sheet using the <LINK> tag
- The JavaScript file **/includes/rollover.js** is embedded to make the functions available
- The top bar is now generated. The appearance and behaviour of the top bar is determined based on the following conditions:
  - If the visitor **has not logged in** (i.e. variable **userID** is empty), the appearance of the top bar is as shown in diagram 18.3.1.

Diagram 18.3.1

The first row consists of a login section that accepts the username and password. When GO is clicked, control is passed to the **admincheck.php** file. The second row consists of the SIGNUP link.

When the mouse is taken over the SIGNUP link, a drop-down menu appears as shown in diagram 18.3.1.1.

Diagram 18.3.1.1

- If the visitor **has logged in** as a **candidate** (i.e. variable **UserType** holds **cand**), the appearance of the top bar is as shown in diagram 18.3.2.

Diagram 18.3.2

The first row consists of two links:

- CANDIDATE SECTION – This link passes control to the candidate's main page, which allows to manage candidate information
- LOGOUT – This link logouts a Candidate and returns control to the login page

The second row consists of the HELP link, which provides help based on the active page.

- If the visitor **has logged in** as a **client** (i.e. variable **UserType** holds **clnt**), the appearance of the top bar is as shown in diagram 18.3.3.

Diagram 18.3.3

The first row consists of three links:

- CLIENTS SECTION – This link passes control to the client's main page, which allows the management of client information
- SEARCH RESUMES – This link passes control to a page which allows a client to search for prospective candidates
  Since the client's section is not fully operational, the link will display a message indicating that this functionality is under construction.
- LOGOUT – This link logouts a Client and returns the control to the login page

The second row consists of the HELP link, which provides help based on the active page.

- A check is made whether authentication is required on the page, which has the **header.php** file included. If authentication is required (This is done using a variable named **auth_reqd** holding value **YES**) and nobody is logged in (This is done using a variable named **userID** holding **no value**), a call is made to the **loginerror.php** file, which displays an error message.

- The above check is performed just to **suppress/bypass** the error message on pages, which do not require logging in, but have the **header.php** file included.
  *Pages like **login.php** and **cand_reg.php** (as the visitor has not yet registered and hence no userID will be available) are the examples.*
- This check **also restricts** a visitor from **reaching/accessing** pages such as cand_main.php, cand_qual.php, cand_empl.php, cand_lang.php, cand_pers.php and chng_login.php using direct URLS from a web browser without actually logging in. (Preventing being hacked into)
  *The above concept is achieved by initializing the variable **auth_reqd** to "**YES**" in the above stated pages. This action informs the **header.php** file as to which pages **require authentication***

5. footer.php

This file is responsible to render the Copyright, Developed By and Company information which will appear at the end of all pages, which have the **footer.php** file, included. Refer diagram 18.4.

Diagram 18.4

6. functions.php

This file defines php functions, which are made available when included within a page. The functions defined in this file are:

A. display_select_box

Function Name:	display_select_box	
Executed At:	Server Side	
Purpose:	Creates a select box (Drop Down Menu), populates it with required data and also allows setting an initial value for the select box	
Parameters:	**p_cntrl_name**	A name for the select box
	p_query	An SQL query accountable to retrieve desired data
	p_value_field	A column name from the result set (formed after execution of the SQL query) which forms the value part of the select box when retrieved
	p_display_field	A column name from the result set (formed after execution of the SQL query) which forms the display part of the select box when selected/displayed on a page

Parameters:	**p_sel_value**	The current value displayed by the select box. This is useful in update mode for setting the select box value returned from the database based on the current record
	p_initial	The initial value that the select box will display (Default Value)
	p_function	Any extra HTML attributes to be attached to the select box
	p_rcq	The Database handle
Table(s) Used:	Any table that is passed in the 'p_query' parameter	
Logic:	Parses and Executes the SQL query received as a parameter	
	Renders the HTML tag for creating the select box using the values held in the **p_cntrl_name** and **p_function** parameters	
	Sets the initial value for the select box using value held in the **p_initial** parameter	
	Fetches data into a result-set using a while loop	
	Populates the select box based on the records retrieved. This is done within a while loop	
	Renders the closing HTML tags for the select box	

B. gen_serial

Function Name:	gen_serial	
Executed At:	Server Side	
Purpose:	Generates unique numbers that form primary keys while entering form data into a table and then updates the same in a table maintaining unique numbers, thus making the unique number available for the next insert The unique number is made up of a prefix such as: **CD** for **Candidate** table **CA** for **Cand_Addr** table **CE** for **Cand_Empl** table **CQ** for **Cand_Qual** table **US** for **Users** table followed by a unique running number. CD00001, CD00002, US00001 are examples.	
Parameters:	**p_table_name**	The table name holding unique numbers
	p_rcq	The database handle
Table(s) Used:	**pk_generator**	Contains the prefix and the next serial no. for each table

Returns:	serial_number (String)	The next unique number generated

Logic:	Retrieves the contents from the table name passed as a parameter by accessing the pk_generator table
	Fetches the contents into a result set
	Retrieves the last generated unique number (e.g. **5**) from the result set using the NEXT_SERIAL field and stores it in a variable **A**
	Calculates the zero padding required based on the length of last unique number generated and stores it in a variable **B**
	Creates a copy of the variable **A** as **C** holding the last generated unique number
	Using a for loop prefix the last generated unique number with zeros, based on the value held by the variable **B** holding the padding required and update the contents of variable **C** with the unique number padded with zeros (e.g. **0000**5)
	Retrieves the first two characters that are to be prefixed with the unique number (e.g. CD, CA, US) from the result-set
	Prefixes the characters retrieved with the padded unique number held by the variable **C** and store it in a variable **D** (e.g. **CD**00005)
	Increments the value in the variable **A** by **1**
	Updates the contents of the table **pk_generator** with this incremented value for the next run
	Returns the value held by the variable **D**

The ADMINISTRATION Module

The administration module allows managing information stored in Master tables. To start a session of Personnel Management System's ADMINISTRATION module:

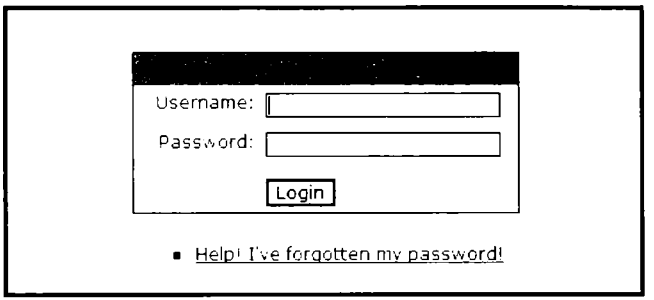


Diagram 18.5: ADIMINSTRATION module's Login page.

1. Enter the following URL in the Web Browser's address bar:
   http://sct.pms.com/administration/
2. Enter **admin** as the username and the appropriate password. Click Login. Refer diagram 18.5

The ADMINISTRATION module's Login page file is available under **/administration/index.php**. Refer diagram 18.0

Diagram 18.6: Top bar generated by the Header page for the ADMINISTRATION module

This module consists of the following pages:

1. header.php
The header page of the ADMINISTRATION module is available under **/administration/header.php**

Purpose

- Generate a top bar as shown in diagram 18.6. A style sheet named **main.css** is used to define the appearance of objects and text
- Avoid replication of similar code at the beginning of pages in the ADMINISTRATION module that require this top bar

GENERAL PAGE SPECIFICATIONS FOR header.php

FILES INCLUDED IN header.php:

PHP CODE BLOCK:	config.php	Declares PHP Global Variables
	db_connect.php	Creates PHP Database connection Object
	functions.php	Declares PHP based user-defined functions
HTML CODE BLOCK:	main.css	Defines the style sheeting for the HTML page
	rollover.js	Declares JavaScript based user-defined functions

FUNCTIONS DECLARED IN header.php:

JavaScript:	PHP:
N/A	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	LOGOUT	N/A	**Call Made:** index.php?wish=logout

FORM DETAILS:

FORM NAME: N/A	**BOUND TO:** N/A	**ACTION:** N/A

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
N/A	N/A	N/A	N/A

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
N/A	N/A	N/A	N/A

Processing - Header Page Of The ADMINISTRATION Module (header.php)

This page undergoes the following processes:

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A session is initialized using the PHP function **session_start()**
- A check is made if the user is logging-out. This means that the visitor has clicked Logout and the control has passed to a page, which includes header.php. This is done as:
  - A variable **mode** holds the state in terms of **login** or **logout**
  - If the variable **mode** holds the value **logout**, then any existing session variables bound to previous login, are unregistered and re-initialized
- This is followed by a check to verify that the session (global) variable named **AdminUser** holds a value. If no value is held by this variable, the control is passed to the login screen of the ADMINISTRATION module
- Header for this page is set with the following attributes:
  - Cache-Control - no-cache, must-revalidate
  - Pragma - no-cache
  - Last-Modified date
- The following three files are included:
  - /includes/config.php
    - This file is included to make the variables declared available in the header.php
  - /includes/db_connect.php
    - This file is included to establish a connection with the database and thus make available the variable rcq which is the database handle
  - /includes/functions.php
    - This file is included to use functions defined earlier
- A variable named **style_sheet** is initialized to hold the value **main.css**. This is used while embedding the style sheet using the <LINK> tag
- The JavaScript file **/administration/rollover.js** is embedded to make the functions available
- The top bar is now generated as shown in diagram 18.7

Diagram 18.7

The top bar as seen in diagram 18.7 consists of Company Name and a LOGOUT button which when clicked passes the control back to the **index.php** page with a parameter named **wish** holding value **logout**. This means the value logout is passed to the **index.php** page which informs the **index.php** page that the user has logged out and from now on till the user logs in again the user should not be allowed to access other pages belonging to the administration module.

When the parameter **wish** holds **logout** the following processes run:

- The session variable containing the name of the user is un-registered and the session is terminated
- The global variable used for storing the name of the administrator is re-initialized

These processes actually help prevent access of other pages in this module, without re-login.

2. **index.php** – This page is the ADMINISTRATION Module's LOGIN Page

Layout - Index Page Of The ADMINISTRATION Module's LOGIN Page (index.php)

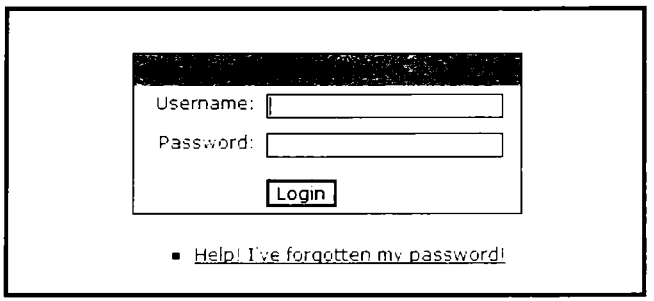


Diagram 18.8.0

Purpose

- Capture and validate the Administrator's Login information
- Provide an option to send e-mail to the Web Site Administrator, informing about having forgotten the password

GENERAL PAGE SPECIFICATIONS FOR index.php

FILES INCLUDED IN index.php:

PHP CODE BLOCK:	config.php	Declares PHP Global Variables
HTML CODE BLOCK:	main.css	Defines the style sheeting for the HTML page
	rollover.js	Declares JavaScript based user-defined functions

FUNCTIONS DECLARED IN index.php:

JavaScript:	PHP:
frmValidate()	- -

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Help! I've forgotten my password!	- -	mailto:admin@sctindia.com

FORM DETAILS:

FORM NAME: frmAuthLogin	BOUND TO: USERS	ACTION: admincheck.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Text Box	Username	txtUserName	USER_USERNAME
Text Box	Password	txtPassword	USER_PASSWORD

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Button	Login	- -	- -

Processing - Index Page Of The ADMINISTRATION Module's LOGIN Page (index.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- The following file is included:
  - /includes/config.php
    - This file is included to make the variables declared available in the index.php
- A session variable named **mode** is registered. This variable holds the value **logout** if the user had clicked logout in some previous page and the control was passed to this page. This is done using an IF condition which checks if a variable name **wish** holds the value **logout** (passed from some previous page as **?wish=logout**)
- This is followed by a java script function named **frmValidate**. This function performs validation of the username and password captured by this page
- A provision is made for displaying error messages. This is done by rendering the contents of a variable named **msg**. This variable is defined and populated with an error message in the file **admincheck.php**. This is the file which actually does the authentication of the username and password by the **index.php** file and on failure passes an error message via the variable **msg**

- A web page is designed to get an output as shown in diagram 18.5. A style sheet named **main.css** defines the appearance of objects and text.

Buttons On The Index Page Of The ADMINISTRATION Module's LOGIN Page

Login

Purpose
This button calls the JavaScript function **frmValidate()** to verify login information before allowing it to be submitted to the **/administration/admincheck.php** page.

Functionality

- Validates the data captured by the Username and Password fields and returns **true** when the process is successful
- Verifies that the value entered in the fields does not contain blank spaces while maintaining the minimum length of username and password fields as 5 and 6 characters respectively
- If any of these conditions fail, prevent the login information for being submitted

Links On The Index Page Of The ADMINISTRATION Module's LOGIN Page

Help! I've forgotten my password!

Purpose
To generate an e-mail informing the Web Site's Administrator about having forgotten an Admin password

Functionality

- Invoke the default mailing tool on the clients machine

3. **admincheck.php** – This page is used to authenticate the login information captured by the **index.php** page and is available under **/administration/admincheck.php**

Purpose

- Authenticate the Administrator's Login information

Processing - Login Authentication For The ADMINISTRATION Module (admincheck.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page

- The following two files are included:
  - /includes/config.php
    - This file is included to make the variables declared available in the admincheck.php
  - /includes/db_connect.php
    - This file is included to establish a connection with the database and thus make available the variable **rcq** which is the database handle
- An SQL query is build and executed which will verify availability of username and password captured by **index.php**. This verification is done on the USER_USERNAME (e.g. admin), USER_PASSWORD (e.g. xxxxxxx) and USER_TYPE (e.g. admn – administrator user) column of the **users** table
- If the username and password are acceptable (i.e. if any record is retrieved by the SQL query):
  - A session variable named **AdminUser** is registered. This is done to make the username available globally across the pages of the administration module
  - Control is now passed to the **/administration/master_cat.php** page. This page allows managing information related to the Job category. It also provides a link to other pages under the administration module
- If the username and password are not acceptable (i.e. if the SQL query fails):
  - The ADMINISTRATION module's login page (**/administration/index.php**) is re-displayed with an error message indicating an Incorrect login attempt. The error message is stored in a variable **msg**. The **index.php** page is instructed to **always render** the contents of **msg** variable. This instruction actually makes it possible to display error messages if any

4. **master_cat.php** – This page provides an interface for managing job categories and is available under **/administration/master_cat.php**

Layout - The Page Managing List Of Job Categories (master_cat.php)

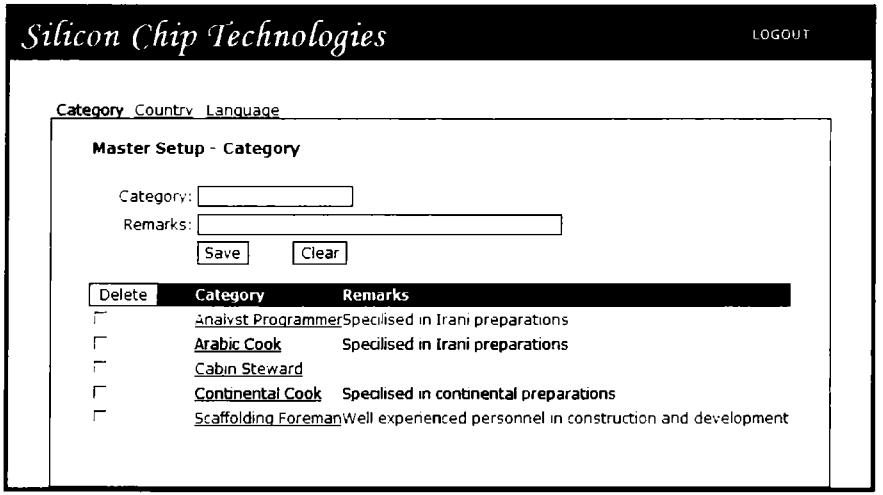


Diagram 18.8.1

GENERAL PAGE SPECIFICATIONS FOR master_cat.php

FILES INCLUDED IN master_cat.php:

PHP CODE BLOCK:	header.php	Generates the top bar that allows Logout
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN master_cat.php:

JavaScript:	PHP:
setEditMode(), setMode(), setDelMode(), chkBlanks()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Country	N/A	**Call Made:** master_ctry.php
Hyperlink	Language	N/A	**Call Made:** master_lang.php

FORM DETAILS:

FORM NAME: FrmMstrCat	**BOUND TO:** CATEGORY	**ACTION:** dosql.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Hidden	N/A	hidSubmit	N/A
Hidden	N/A	hidPage	N/A
Hidden	N/A	hidCatId	N/A
Hidden	N/A	hidCatName	N/A
Hidden	N/A	hidCatRmrk	N/A
Hidden	N/A	hidMode	N/A
Hidden	N/A	hidSelDel	N/A
Text Box	Category	txtCatName	CATEGORY.CATEGORY_NAME
Text Box	Remarks	txtCatRmrk	CATEGORY.CATEGORY_REMARKS
Checkbox	N/A	chk1, chk2, . . .	N/A

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Hyperlink	<Categories List>	N/A	JavaScript:setEditMode()
Button	Save	cmdSubmit	JavaScript:onSubmit()
Button	Clear	cmdReset	JavaScript:setMode()
Button	Delete	cmdDelete	JavaScript:setDelMode()

Purpose

- Adding new Job category
- Updating an existing Job category
- Deleting an existing Job category
- Link to Data Entry screen for managing List of Countries
- Link to Data Entry screen for managing List of Languages
- Logout and return to the ADMINISTRATION module's Login page (index.php)

Processing - The Page Managing List Of Job Categories (master_cat.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- The following file is included:
  - /administration/header.php
    - This file is included to make the top bar available in the master_cat.php. Refer diagram 18.8.2

Diagram 18.8.2

- A provision is made for displaying error messages. This is done by rendering the contents of variables named **hidMsg** and **hidErr**. These variables are defined and populated with appropriate error messages in the file **dosql.php**. This is the file which actually performs database operations requested by the data entry forms of the administration module and on non-performance of the requested database operation passes appropriate error messages via the variables **hidMsg** and **hidErr**
- This is now followed by following set of java script functions:
  - **setEditMode** – This function accepts three parameters based on the three fields available in the **category** table. It is called when a record is selected for updation and performs the following operations:
    - Transfers the value held by the first, second and the third parameters to the hidden variables
    - Transfers the value held by the second and the third parameter to the form field thus making it available for updation
    - Switches the form mode to update. This is done by setting the value of hidden variable named **hidMode** to **U**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Disables the Delete button available on the **master_cat.php** form. This is done to avoid any delete operation while the form is in update mode
  - **setMode** – This function is called when a clear button is clicked and performs the following operations which are **exactly inverse** of the ones performed by the **setEditMode:**
    - Clears the value held by the hidden variable defined in the **setEditMode**
    - Clears the value held by the form fields
    - Clears the value held by the hidden variable **hidSumit**. This variable holds **no value** (indicating no database operations are pending) or **1** (indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation)
    - Switches the form mode back to insert. This is done by setting the value of hidden variable named **hidMode** to **I**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Enables the Delete button available on the **master_cat.php** form
  - **setDelMode** – This function is called when a delete button is clicked and performs the following operations:
    - Sets the value held by the hidden variable **hidSumit** to **1** indicating that data has been submitted for processing, i.e. either Insert, Update or Delete operation

- Switches the form mode to delete. This is done by setting the value of hidden variable named **hidMode** to **D**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
- Calls a function named **formDeleteValues** available in the **/administration/rollover.js** file with a parameter to hold list of category ids for deletion. This function generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same

- **chkBlanks** – This function is called when a form data is about to be submitted. This function verifies that the category name field is not left blank. If it is left blank an error message indicates the same. Otherwise the value held by the hidden variable **hidSumit** to **1** indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation

- The page is designed using HTML tags. The FORM tag holds **ACTION** pointing to the **dosql.php** file and **onSubmit** pointing to the **chkBlanks** function
- Following hidden variables are declared using the INPUT tags:
  - **hidSumit** – Holds **0** (indicating no database operations are pending) or **1** (indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation)
  - **hidPage** – Holds the page name in this case **master_cat.php**
  - **hidCatId** – Holds the Category Identity for delete/update operations
  - **hidCatName** – Holds the Category Name for delete/update operations
  - **hidCatRmrk** – Holds the Category Remarks for delete/update operations
  - **hidMode** – Is used to determine the form mode. Holds 'I' for Insert, 'U' for Update and 'D' for Delete. It will be left empty when the page is rendered for the First time
  - **hidSelDel** – Holds the country identities for identifying records which have been selected for deletion
- HTML Code-spec follows for creating links to the other two forms namely country (master_ctry.php) and languages (master_lang.php). Links are created within table cells and hence generates a tab effect. Refer Diagram 18.9.0

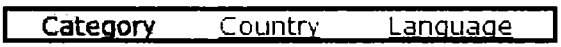


Diagram 18.9.0

- HTML Code-spec follows for designing the form fields such as **Category** and **Remarks** and buttons such as **Save** and **Clear**. Refer Diagram 18.9.1

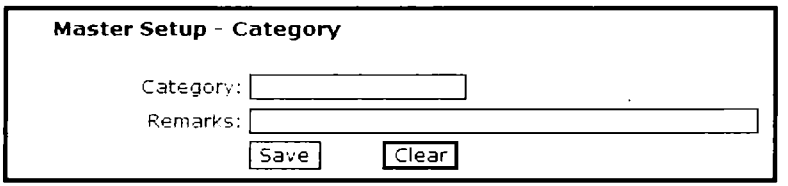


Diagram 18.9.1

- An SQL Query is formed and executed to retrieve the records available in the table. These records will be displayed in a tabular form. Refer Diagram 18.9.2

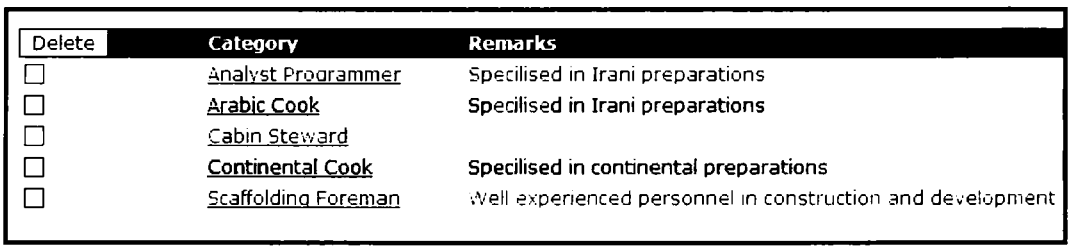


Diagram 18.9.2

- If the SQL query retrieves any records the following operations are performed:
  - HTML Code-spec follows for the creation of a Delete button with its **onClick** attribute set to a call a function named **setDelMode**
  - A tabular layout is created to hold the records retrieved. This is done using a **do while** loop, which traverses through the records retrieved from the table via the SQL query. While traversing through the loop the color code for each row displayed in the tabular layout as seen in diagram 18.9.2 is switched between two different colors. This is done purely to improve readability
  - **Category** as seen in the diagram 18.9.2 is linked to the function **setEditMode.** This function is responsible for populating the form fields as seen in diagram 18.9.1 when a record is clicked from the tabular layout.

5. **master_ctry.php** – This page provides an interface for managing countries and is available under **/administration/master_ctry.php**

Layout - The Page Managing List Of Countries (master_ctry.php)

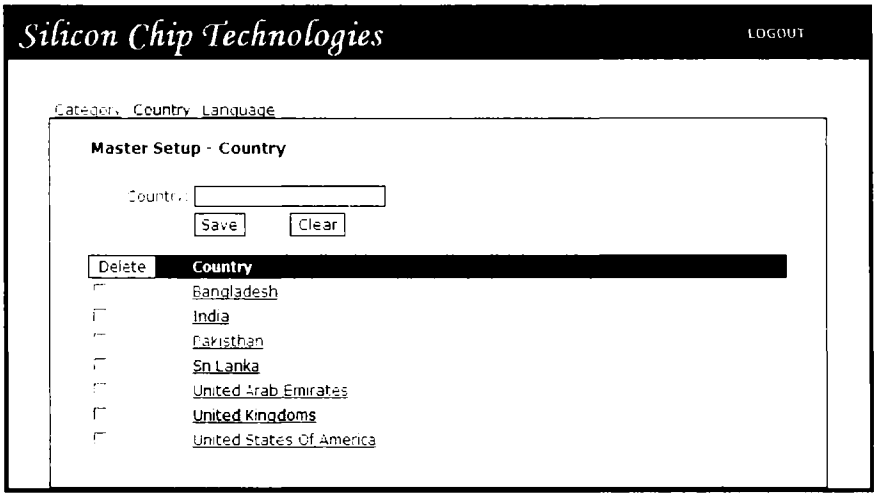


Diagram 18.10.1

Purpose

- Adding new countries
- Updating an existing country
- Deleting an existing country
- Link to Data Entry screen for managing List of Categories
- Link to Data Entry screen for managing List of Languages
- Logout and return to the ADMINISTRATION module's Login page (index.php)

GENERAL PAGE SPECIFICATIONS FOR master_ctry.php

FILES INCLUDED IN master_ctry.php:

PHP CODE BLOCK:	header.php	Generates the top bar that allows Logout
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN master_ctry.php:

JavaScript:	PHP:
setEditMode(), setMode(), setDelMode(), chkBlanks()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Category	N/A	**Call Made:** master_cat.php
Hyperlink	Language	N/A	**Call Made:** master_lang.php

FORM DETAILS:

FORM NAME: FrmMstrCtry	**BOUND TO:** COUNTRY	**ACTION:** dosql.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Hidden	N/A	hidSubmit	N/A
Hidden	N/A	hidPage	N/A
Hidden	N/A	hidCtryId	N/A
Hidden	N/A	hidMode	N/A
Hidden	N/A	hidSelDel	N/A
Text Box	Country	txtCtry	COUNTRY.COUNTRY_NAME
Checkbox	N/A	chk1, chk2, . . .	N/A

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Hyperlink	<Countries List>	N/A	JavaScript:setEditMode()
Button	Save	cmdSubmit	JavaScript:onSubmit()
Button	Clear	cmdReset	JavaScript:setMode()
Button	Delete	cmdDelete	JavaScript:setDelMode()

Processing - The Page Managing List Of Countries (master_ctry.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- The following file is included:
  - /administration/header.php
    - This file is included to make the top bar available in the master_ctry.php. Refer diagram 18.10.2

Diagram 18.10.2

- A provision is made for displaying error messages. This is done by setting the contents of the variables named **hidMsg** and **hidErr**. These variables are defined and populated with appropriate error messages in the file **dosql.php**. This is the file which actually performs database operations requested by the data entry forms of the administration module and on non-performance of the requested database operation passes appropriate error messages via the variables **hidMsg** and **hidErr**

- This is now followed by following set of java script functions:
  - **setEditMode** – This function accepts two parameters based on the two fields available in the **country** table. It is called when a record is selected for updation and performs the following operations:
    - Transfers the value held by the first parameter to the hidden variable.
    - Transfers the value held by the second parameter to the form field thus making it available for updation
    - Switches the form mode to update. This is done by setting the value of hidden variable named **hidMode** to **U**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Disables the Delete button available on the **master_ctry.php** form. This is done to avoid any delete operation while the form is in update mode
  - **setMode** – This function is called when a clear button is clicked and performs the following operations which are **exactly inverse** of the ones performed by the **setEditMode:**
    - Clears the value held by the hidden variable defined in the **setEditMode**
    - Clears the value held by the form fields
    - Clears the value held by the hidden variable **hidSumit**. This variable holds **no value** (indicating no database operations are pending) or **1** (indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation)
    - Switches the form mode back to insert. This is done by setting the value of hidden variable named **hidMode** to **I**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Enables the Delete button available on the **master_ctry.php** form
  - **setDelMode** – This function is called when a delete button is clicked and performs the following operations:
    - Sets the value held by the hidden variable **hidSumit** to **1** indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation
    - Switches the form mode to delete. This is done by setting the value of hidden variable named **hidMode** to **D**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Calls a function named **formDeleteValues** available in the **/administration/rollover.js** file with a parameter to hold list of country ids for deletion. This function generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same

- **chkBlanks** – This function is called when a the form data is about to be submitted. This function verifies that the country name field is not left blank. If it is left blank an error message indicates the same. Otherwise the value held by the hidden variable **hidSumit** to **1** indicating that data has been submitted for processing **i.e.** either Insert, Update or Delete operation

- The page is designed using HTML tags. The FORM tag holds **ACTION** pointing to the **dosql.php** file and **onSubmit** pointing to the **chkBlanks** function
- Following hidden variables are declared using the INPUT tags:
  - **hidSumit** – Holds **0** (indicating no database operations are pending) or **1** (indicating that data has been submitted for processing **i.e.** either Insert, Update or Delete operation)
  - **hidPage** – Holds the page name in this case **master_ctry.php**
  - **hidCtryId** – Holds the Country Identity for delete/update operations
  - **hidMode** – Is used to determine the form mode. Holds 'I' for Insert, 'U' for Update and 'D' for Delete. It will be left empty when the page is rendered for the First time
  - **hidSelDel** – Holds the country identities for identifying records which have been selected for deletion
- HTML Code-spec follows for creating links to the other two forms namely category (master_cat.php) and languages (master_lang.php). Links are created within table cells and hence generates a tab effect. Refer Diagram 18.11.0

Diagram 18.11.0

- HTML Code-spec follows for designing the form field Country and buttons such as **Save** and **Clear**. Refer Diagram 18.11.1

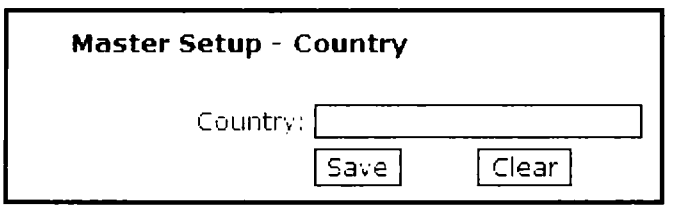


Diagram 18.11.1

- An SQL Query is formed and executed to retrieve the records available in the table. These records will be displayed in a tabular form. Refer Diagram 18.11.2

Diagram 18.11.2

- If the SQL query retrieves any records following operations are performed:
  - HTML Code-spec creates the Delete button whose **onClick** attribute set to call the function named **setDelMode**
  - A tabular layout is created to hold the records retrieved. This is done using a **do while** loop, which traverses through the records retrieved from the table via the SQL query. While traversing through the loop the color code for each row displayed in the tabular layout seen in diagram 18.11.2 is switched between two different colors. This is done purely to improve readability
  - The **Country** as seen in the diagram 18.11.2 is assigned a link to the function **setEditMode.** This function is responsible to populate the form fields as seen in diagram 18.11.1 when a record is clicked from the tabular layout

6. master_lang.php – This page provides an interface for managing languages and is available under **/administration/master_lang.php**

Layout - The Page Managing List Of Languages (master_lang.php)

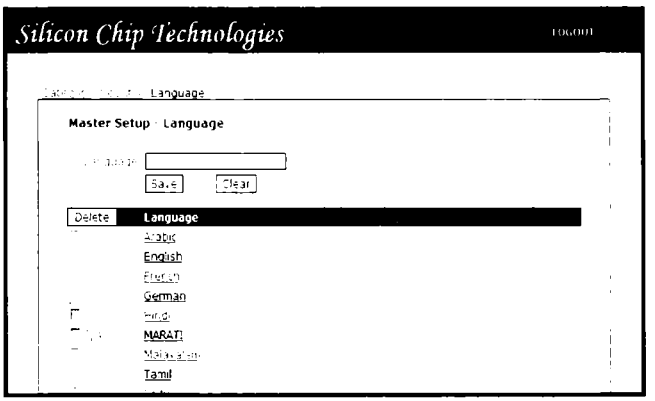


Diagram 18.12.1

Purpose

- Adding new languages
- Updating an existing language
- Deleting an existing language
- Link to Data Entry screen for managing List of Categories
- Link to Data Entry screen for managing List of Countries
- Logout and return to the ADMINISTRATION module's Login page (index.php)

GENERAL PAGE SPECIFICATIONS FOR master_lang.php

FILES INCLUDED IN master_lang.php:

PHP CODE BLOCK:	header.php	Generates the top bar that allows Logout
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN master_lang.php:

JavaScript:	**PHP:**
setEditMode(), setMode(), setDelMode(), chkBlanks()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	**LABEL**	**NAME**	**DESCRIPTION**
Hyperlink	Category	N/A	**Call Made:** master_cat.php
Hyperlink	Country	N/A	**Call Made:** master_ctry.php

FORM DETAILS:

FORM NAME: FrmMstrLang	**BOUND TO:** LANGUAGE	**ACTION:** dosql.php

DATA FIELDS:

OBJECT	**LABEL**	**NAME**	**BOUND TO**
Hidden	N/A	hidSubmit	N/A
Hidden	N/A	hidPage	N/A
Hidden	N/A	hidLangId	N/A
Hidden	N/A	hidMode	N/A
Text Box	Language	txtLang	LANGUAGE.LANGUAGE_NAME
Checkbox	N/A	chk1, chk2, . . .	N/A

DATA CONTROLS:

OBJECT	**LABEL**	**NAME**	**ACTION**
Hyperlink	<Countries List>	N/A	JavaScript:setEditMode()
Button	Save	cmdSubmit	JavaScript:onSubmit()
Button	Clear	cmdReset	JavaScript:setMode()
Button	Delete	cmdDelete	JavaScript:setDelMode()

Processing - The Page Managing List Of Languages (master_lang.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- The following file is included:
  - /administration/header.php
    - This file is included to make the top bar available in the master_lang.php. Refer diagram 18.12.2

Diagram 18.12.2

- A provision is made for displaying error messages. This is done by setting the contents of the variables named **hidMsg** and **hidErr**. These variables are defined and populated with appropriate error messages in the file **dosql.php**. This is the file which actually performs database operations requested by the data entry forms of the administration module and on non-performance of the requested database operation passes appropriate error messages via the variables **hidMsg** and **hidErr**
- This is now followed by following set of java script functions:
  - **setEditMode** – This function accepts two parameters based on the two fields available in the **language** table. It is called when a record is selected for updation and performs the following operations:
    - Transfers the value held by the first parameter to the hidden variable
    - Transfers the value held by the second parameter to the form field thus making it available for updation
    - Switches the form mode to update. This is done by setting the value of hidden variable named **hidMode** to **U**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Disables the [Delete] button available on the **master_lang.php** form. This is done to avoid any delete operation while the form is in update mode
  - **setMode** – This function is called when a clear button is clicked and performs the following operations which are **exactly inverse** of the ones performed by the **setEditMode:**
    - Clears the value held by the hidden variable defined in the **setEditMode**
    - Clears the value held by the form fields
    - Clears the value held by the hidden variable **hidSumit**. This variable holds **no value** (indicating no database operations are pending) or **1** (indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation)

- Switches the form mode back to insert. This is done by setting the value of the Shidden variable named **hidMode** to **I**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operations
- Enables the Delete button available on the **master_lang.php** form

- **setDelMode** – This function is called when a delete button is clicked and performs the following operations:
  - Sets the value held by the hidden variable **hidSumit** to **1** indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation
  - Switches the form mode to delete. This is done by setting the value of hidden variable named **hidMode** to **D**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
  - Calls a function named **formDeleteValues** available in the **/administration/rollover.js** file with a parameter to hold list of language ids for deletion. This function generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same
- **chkBlanks** – This function is called when the form data is about to be submitted. This function verifies that the language name field is not left blank. If it is left blank an error message indicates the same. Otherwise the value held by the hidden variable **hidSumit** to **1** indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation

- The page is designed using HTML tags. The FORM tag holds **ACTION** pointing to the **dosql.php** file and **onSubmit** pointing to the **chkBlanks** function
- Following hidden variables are declared using the INPUT tags:
  - **hidSumit** – Holds **0** (indicating no database operations are pending) or **1** (indicating that data has been submitted for processing i.e. either Insert, Update or Delete operation)
  - **hidPage** – Holds the page name in this case **master_lang.php**
  - **hidLangId** – Holds the Language Identity for delete/update operations
  - **hidMode** – Is used to determine the form mode. Holds 'I' for Insert, 'U' for Update and 'D' for Delete. It will be left empty when the page is rendered for the First time
  - **hidSelDel** – Holds the Language identities for identifying records which have been selected for deletion
- HTML Code-spec that creates links to the other two forms namely **category** (master_cat.php) and countries (master_ctry.php) follows. Links are created within table cells and hence generates a tab effect. Refer Diagram 18.13.0

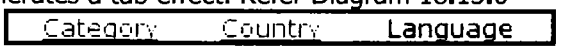


Diagram 18.13.0

- HTML Code-spec follows for designing the form field Language and buttons such as **Save** and **Clear**. Refer Diagram 18.13.1

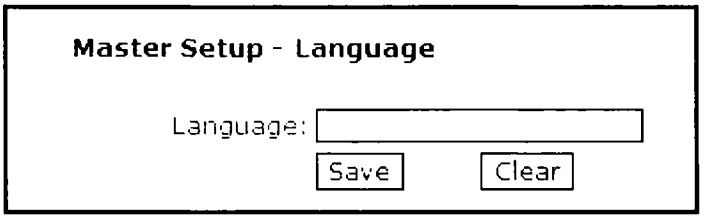


Diagram 18.13.1

- An SQL Query is formed and executed to retrieve the records available in the table. These records will be displayed in a tabular form. Refer Diagram 18.13.2

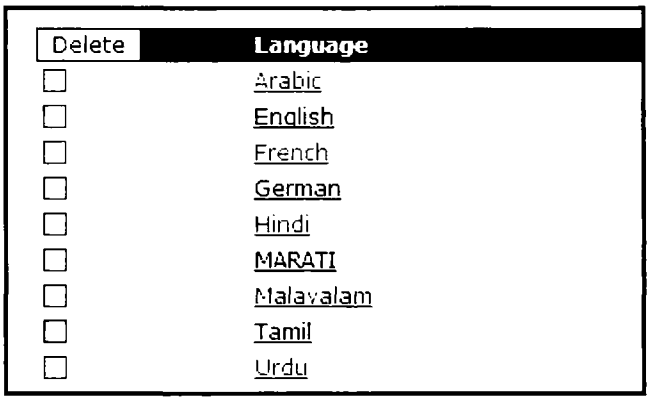


Diagram 18.13.2

- If the SQL query retrieves any records following operations are performed:
  - HTML Code-spec follows for creation of Delete button which has a **onClick** attribute set to a function call named **setDelMode**
  - A tabular layout is created to hold the records retrieved. This is done using a **do while** loop which traverses through the records retrieved from the table via the SQL query. While traversing through the loop the color code for each row displayed in the tabular layout seen in diagram 18.13.2 is switched between two different colors. This is done to improve the readability
  - The **Language** as seen in the diagram 18.13.2 is assigned a link to the function **setEditMode.** This function is responsible to populate the form fields as seen in diagram 18.13.1 when a record is clicked from the tabular layout.

7. **dosql.php** – This file performs the Insert, Update and Delete operations for all the data entry forms of the administration module and is available under **/administration/dosql.php**

Purpose

- Transfers data captured by the d/e form to the database
- Allows deleting data from the database based on the instructions given by the d/e form
- Allows modifications to the data based on the changes made via the d/e form

Processing - DOSQL.PHP

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- The following three files are included:
  - /includes/config.php
    - This file is included to make the variables declared available in the dosql.php
  - /includes/db_connect.php
    - This file is included to establish a connection with the database and thus make available the variable **rcq** which is the database handle
  - /includes/functions.php
    - This file is included to use functions defined earlier
- Following checks are made:
  - If the variable **hidPage** holds the value **master_lang**. This means control had been passed by the **master_lang.php** file for either **I**nsert, **U**pdate or **D**elete operations. If it holds **master_lang** then:
    - A check is made on the value held by the variable **hidSubmit**. If it holds the value **1** it means that some database operation is desired and processes further as follows
      - A check is made on the value held by the variable **hidMode**. If it holds the value **I** it means that an **Insert** operation is desired and processes further as:
        - Based on the data captured by the **master_lang.php** an SQL query is formed and executed to check for data duplication in terms of **language name**. If the language name is found to be duplicating then the control is passed back to the data entry form i.e. **master_lang.php** along with an appropriate error message for display. This error is populated in the variable **hidErr** which is used by the **master_lang.php** file to display an error message
        - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual insertion of data in the database. During this insert operation care is taken to retrieve the next primary key value. This is done using the highest primary key value inserted earlier + 1
      - A check is made on the value held by the variable **hidMode**. If it holds the value **U** it means that an **Update** operation is desired and processes further as:

- Based on the modified data captured by the **master_lang.php** an SQL query is formed and executed to check for data duplication in terms of **language name**. If the language name is found to be duplicating then the control is passed back to the data entry form i.e. **master_lang.php** along with an appropriate error message for display. This error is populated in the variable **hidErr** which is used by the **master_lang.php** file to display an error message
- If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual update of data in the database

o A check is made on the value held by the variable **hidMode**. If it holds the value **D** it means that a **Delete** operations is desired and processes further as:

- In this mode a variable **hidSelDel** holds comma separated string of the identities (in this case Language_ID) to be deleted. This comma separated string is created by a java-script function **formDeleteValues** available in the **rollover.js** file explained earlier. This comma separated string is transformed into an array using the split function
- A counter (**ctr**) is initialized based on the size of the array. This is required to traverse a specific number of times (based on the value held by the **ctr** variable) using a loop
- A while loop is used to perform the following operations:
  - An SQL query is formed and executed to check if the data marked for deletion based on the **Language_ID** is used in child tables.
  - If the current record is used as a child record then the delete operation is **terminated** for that particular record and that particular identity is populated/appended to a variable. This variable will hold all those identities that are available as child records and thus cannot be deleted. This variable will serve as an error message indicating the languages that were not deleted during this operation due to the child records existence
  - If the current record has no child record available then an SQL query is formed and executed to perform the actual delete operation

o Finally based on the identities (Language_ID) that were not deleted during this operation an SQL query is formed and executed which will retrieve the language names based on the identities passed as a parameter. These language names using a while loop are populated in a variable to hold them in a comma separated format. This variable is now passed to the **master_lang.php** file. The **master_lang.php** is instructed to always render the contents of the variable if it is populated and thus makes the error message visible if it exists

- If the variable **hidPage** holds the value **master_ctry**. This means control had been passed by the **master_ctry.php** file for either **I**nsert, **U**pdate or **D**elete operations. If it holds **master_ctry** then:

- A check is made on the value held by the variable **hidSubmit**. If it holds the value **1** it means that some database operation is desired and processes further as follows
  - A check is made on the value held by the variable **hidMode**. If it holds the value **I** it means that an **Insert** operation is desired and processes further as:
    - Based on the data captured by the **master_ctry.php** an SQL query is formed and executed to check for data duplication in terms of **country name**. If the country name is found to be duplicating then the control is passed back to the data entry form i.e. **master_ctry.php** along with an appropriate error message for display. This error is populated in the variable **hidErr** which is used by the **master_ctry.php** file to display an error message
    - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual insertion of data in the database. During this insert operation care is taken to retrieve the next primary key value. This is done using the highest primary key value inserted earlier + 1
  - A check is made on the value held by the variable **hidMode**. If it holds the value **U** it means that an **Update** operation is desired and processes further as:
    - Based on the modified data captured by the **master_ctry.php** an SQL query is formed and executed to check for data duplication in terms of **country name**. If the country name is found to be duplicating then the control is passed back to the data entry form i.e. **master_ctry.php** along with an appropriate error message for display. This error is populated in the variable **hidErr** which is used by the **master_ctry.php** file to display an error message
    - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual update of data in the database
  - A check is made on the value held by the variable **hidMode**. If it holds the value **D** it means that a **Delete** operation is desired and processes further as:
    - In this mode a variable **hidSelDel** holds comma separated string of the identities (in this case Country_ID) to be deleted. This comma separated string is created by a java-script function **formDeleteValues** available in the **rollover.js** file explained earlier. This comma separated string is transformed into an array using the split function
    - A counter (**ctr**) is initialized based on the size of the array. This is required to traverse a specific number of times (based on the value held by the **ctr** variable) using a loop

- A while loop is used to perform the following operations:
  - An SQL query is formed and executed to check if the data marked for deletion based on the **Country_ID** is used in child tables.
  - If the current record is used as a child record then the delete operation is **terminated** for that particular record and that particular identity is populated/appended to a variable. This variable will hold all those identities that are available as child records and thus cannot be deleted. This variable will serve as an error message indicating the languages that were not deleted during this operation due to the child records existence
  - If the current record has no child record available then an SQL query is formed and executed to perform the actual delete operation

- Finally based on the identities (Country_ID) that were not deleted during this operation an SQL query is formed and executed which will retrieve country names based on the identities passed as a parameter. These country names using a while loop are populated in a variable to hold them in a comma separated format. This variable is now passed to the **master_ctry.php** file. The **master_ctry.php** is instructed to always render the contents of the variable if it is populated and thus makes the error message visible if it exists

- If the variable **hidPage** holds the value **master_cat**. This means control had been passed by the **master_cat.php** file for either **I**nsert, **U**pdate or **D**elete operations. If it holds **master_cat** then:
  - A check is made on the value held by the variable **hidSubmit**. If it holds the value **1** it means that some database operation is desired and processes further as follows
    - A check is made on the value held by the variable **hidMode**. If it holds the value **I** it means that an **Insert** operation is desired and processes further as:
      - Based on the data captured by the **master_cat.php** an SQL query is formed and executed to check for data duplication in terms of **category name**. If the category name is found to be duplicating then the control is passed back to the data entry form i.e. **master_cat.php** along with an appropriate error message for display. This error is populated in the variable **hidErr** which is used by the **master_cat.php** file to display an error message
      - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual insertion of data in the database. During this insert operation care is taken to retrieve the next primary key value. This is done using the highest primary key value inserted earlier + 1

- A check is made on the value held by the variable **hidMode**. If it holds the value **U** it means that an **Update** operation is desired and processes further as:
  - A special check is made to verify if any changes (in terms of the Category Name and Category Remarks fields) are made at data entry form level before an update is attempted. This means that:
    - If the record is selected for modification but no changes are made to either Category Name or Category Remarks fields, but the **Save** button is clicked to perform an update, an error message will be generated by passing the control back to the **master_cat.php** form
    - If the record is selected for modification but only the Category Remarks field is changed and the **Save** button is clicked to perform an update, an SQL query is formed and executed to update the changes made to the Category Remarks field
  - If the above was not the case then based on the modified data captured by the **master_cat.php** an SQL query is formed and executed to check for data duplication in terms of **category name**. If the category name is found to be duplicating then the control is passed back to the data entry form i.e. **master_cat.php** along with an appropriate error message for display. This error is populated in the variable **hidErr** which is used by the **master_cat.php** file to display an error message
  - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual update of data in the database
- A check is made on the value held by the variable **hidMode**. If it holds the value **D** it means that a **Delete** operations is desired and processes further as:
  - In this mode a variable **hidSelDel** holds comma separated string of the identities (in this case Category_ID) to be deleted. This comma separated string is created by a java-script function **formDeleteValues** available in the **rollover.js** file explained earlier. This comma separated string is transformed into an array using the split function
  - A counter (**ctr**) is initialized based on the size of the array. This is required to traverse a specific number of times (based on the value held by the **ctr** variable) using a loop
  - A while loop is used to perform the following operations:
    - An SQL query is formed and executed to check if the data marked for deletion based on the **Category_ID** is used in child tables.

- ❖ If the current record is used as a child record then the delete operation is **terminated** for that particular record and that particular identity is populated/appended to a variable. This variable will hold all those identities that are available as child records and thus cannot be deleted. This variable will serve as an error message indicating the languages that were not deleted during this operation due to the child records existence
- ❖ If the current record has no child record available then an SQL query is formed and executed to perform the actual delete operation

o Finally based on the identities (Category_ID) that were not deleted during this operation an SQL query is formed and executed which will retrieve the category names based on the identities passed as a parameter. These category names using a while loop are populated in a variable to hold them in a comma separated format. This variable is now passed to the **master_cat.php** file. The **master_cat.php** Is instructed to always render the contents of the variable if it is populated and thus makes the error message visible if it exists

Entry Points To The Personnel Management System

1. **index.php** – This page provides an interface to enter the Personnel Management System and is available under **/index.php**

Layout - The Page allowing access to the PMS (index.php)

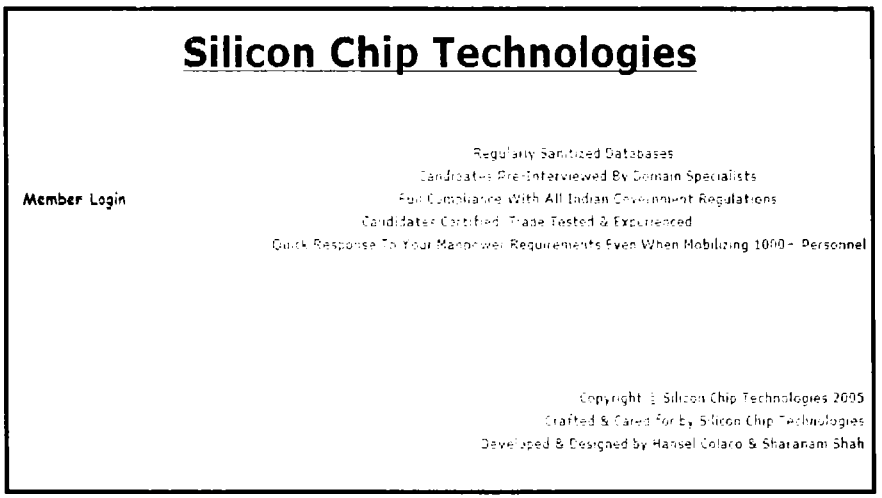


Diagram 18.14

Purpose

- Allows logging into the system
- Briefs the purpose of this website
- Provides details such as Copyrights, Company Name and web site developers

GENERAL PAGE SPECIFICATIONS FOR index.php

FILES INCLUDED IN index.php:

PHP CODE BLOCK:	config.php	Declares PHP Global Variables
HTML CODE BLOCK:	main.css	Defines the style sheeting for the HTML page
	rollover.js	Declares JavaScript based user-defined functions

FUNCTIONS DECLARED IN index.php:

JavaScript:	**PHP:**
N/A	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	**LABEL**	**NAME**	**DESCRIPTION**
Hyperlink	Member Login	N/A	**Call Made:** login.php

FORM DETAILS:

FORM NAME: N/A	**BOUND TO:** N/A	**ACTION:** N/A

DATA FIELDS:

OBJECT	**LABEL**	**NAME**	**BOUND TO**
N/A	N/A	N/A	N/A

DATA CONTROLS:

OBJECT	**LABEL**	**NAME**	**ACTION**
N/A	N/A	N/A	N/A

To start a session of Personnel Management System enter the URL **http://sct.pms.com** in the Web Browser's address bar.

Processing - The Page allowing access to the PMS (index.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page

- The following file is included:
  - /includes/config.php
    - This file is included to make the variables declared available in the index.php
- HTML code-spec follows that creates the page layout using tables to hold:
  - A link Member Login to allow members to login
  - The website purpose
  - Company information
  - Copyright details
  - Developers of the website
- A style sheet named **main.css** defines the appearance of objects and text.

2. **login.php** – This page provides an interface to attempt a login to the Personnel Management System and is available under **/auth/login.php**. This page also allows to signup as a candidate or a client using the SIGNUP button (This functionality is provided by the included **header.php** page explained earlier)

Layout - The Page capturing the login information (login.php)

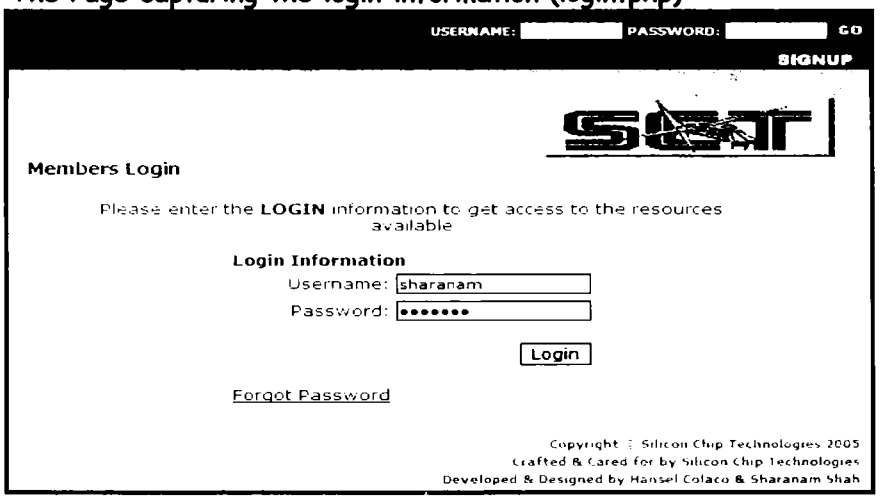


Diagram 18.15

Purpose

- Capture the username and the password
- Allows logging into the system
- Allows the user to send an email to the website administrator for having forgotten the password

GENERAL PAGE SPECIFICATIONS FOR login.php

FILES INCLUDED IN login.php:

PHP CODE BLOCK:	header.php, footer.php	Generates the top bar for the HTML page
HTML CODE BLOCK:	main.css	Defines the style sheeting for the HTML page
	rollover.js	Declares JavaScript based user-defined functions

FUNCTIONS DECLARED IN login.php:

JavaScript:	PHP:
frmValidate()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Forgot Password	N/A	mailto:admin@sctindia.com

FORM DETAILS:

FORM NAME: frmAuthLogin	**BOUND TO:** USERS	**ACTION:** logincheck.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Text Box	Username	txtUserName	USER_USERNAME
Text Box	Password	txtPassword	USER_PASSWORD

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Button	Login	cmdSubmit	N/A

Click  on the **index.php** page to access the **login.php** page. Refer diagram 18.14 and diagram 18.15.

Processing - The Page Capturing The Login Information (login.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A variable named **mode** is registered as a session variable to track the mode of the system in terms of **logout**. This variable will be available globally across all the php pages

- A check is made if the control had been passed to this page by clicking LOGOUT on some other page. This check is performed by validating the contents of variable **wish**. This variable is passed using **?wish=logout** technique via a URL. If variable **wish** holds **logout**, session variable **mode** is initialized with the same value. Based on the value held in the session variable **mode** some **clean up tasks** will be performed by the **header.php** (explained earlier)
- The following file is included:
  - /includes/header.php
    - This file is included to make the top bar available in the **login.php**. Refer diagram 18.16

Diagram 18.16

- Java script code-spec follows to validate username and password captured by this page. This is done using a function named **frmValidate**. This function is called when the information captured by this page is submitted using the Login button. This means a client side validation is performed before validating the username/password against the database at the server end
- Initial HTML code-spec corresponding to the template begins
- HTML code-spec for designing the layout of the form in terms of textboxes and login button responsible to capture the user name and password follows
- This includes a FORM tag definition wherein the **ACTION** attribute of the FORM tag is pointed to the **logincheck.php** file and the **OnSubmit** attribute is pointed to the **frmValidate** java script function
- Finally the **footer.php** file is included to render the **copyright and the developers information** on the **login.php** page. Refer Diagram 18.17

Diagram 18.17

3. **logincheck.php** – This page validates the username and password captured by the **/auth/login.php** page against the database and is available under **/auth/logincheck.php**.

Purpose

- Validates the login information captured by the file **login.php**
- Transfers the control to either Candidate or Client module based on the user logging in

Clicking Login button on the **login.php** page transfers control to the **loginchk.php** page.

Processing - Validating The Login Information Captured (logincheck.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- The following two files are included:
  - /includes/config.php
    - This file is included to make the variables declared available in the logincheck.php
  - /includes/db_connect.php
    - This file is included to establish a connection with the database and thus make available the variable **rcq** which is the database handle
- An SQL query is formed and executed to validate the username and password captured by the **login.php** page
- A check is made to verify if the execution of SQL query has retrieved any information from the Oracle database
- If it retrieves any information it means that the login information exists in the database and thus has fulfilled the validation performed. This is followed by the following processes:
  - Session variables such as **userID** to store the user name, **userType** to store the type in terms of cand/clnt and **loginDateTime** to store the date and time when the user logged in are registered. These variables will be available across all the pages
  - A check is made on the value held by the **userType** session variable. If the **userType** holds **cand** for candidates then:
    - Control is passed to the **cand_main.php** file which is the entry to the candidate module
  - If the **userType** holds **clnt** for clients then:
    - Control is passed to the **clnt_dvlp.php** file which should be is the entry to the client module but currently displays that the client module is under developement
- If no information is retrieved from the database then:
  - Session variables such as **userID** to store the user name, **userType** to store the type in terms of cand/clnt, **loginDateTime** to store the date and time when the user logged in and **mode** to store the form mode in terms of logout (The session variable **mode** is defined in the **login.php** file explained earlier) are unregistered as a default clean-up process

- The following two files are included:
  - /includes/header.php
    - This file is included to generate the top bar
  - /includes/loginerror.php
    - This file is included to render an error message for the failure in the login attempted

The CANDIDATE Module

The candidate module allows the management of information stored in the following tables:

- candidate
- cand_addr
- cand_doc
- cand_qual
- cand_empl
- cand_lang

To start a session of Personnel Management System's CANDIDATE module:

1. Enter the following URL in the Web Browser's address bar:
   http://sct.pms.com
2. User can either **SIGNUP** to register himself as a fresh candidate or **LOGIN** to the CANDIDATE module if the user has already registered as a candidate

This module consists of the following pages:

1. **cand_reg.php**
2. **cand_main.php**
3. **cand_pers.php**
4. **cand_qual.php**
5. **cand_empl.php**
6. **cand_lang.php**
7. **chng_login.php**
8. **dosql.php**

The SIGNUP Process Page Flow

If the user opts to **SIGNUP** as a candidate the **control flow** of the php pages is as follows:

1. cand_reg.php

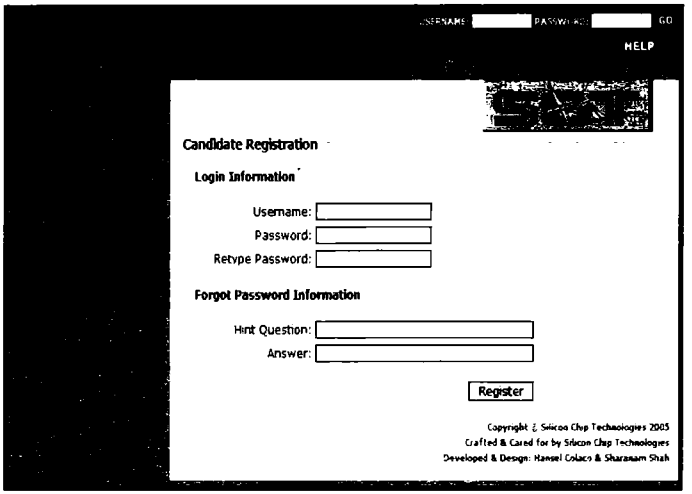


Diagram 18.18.1: The Candidate's Registration page.

2. cand_pers.php

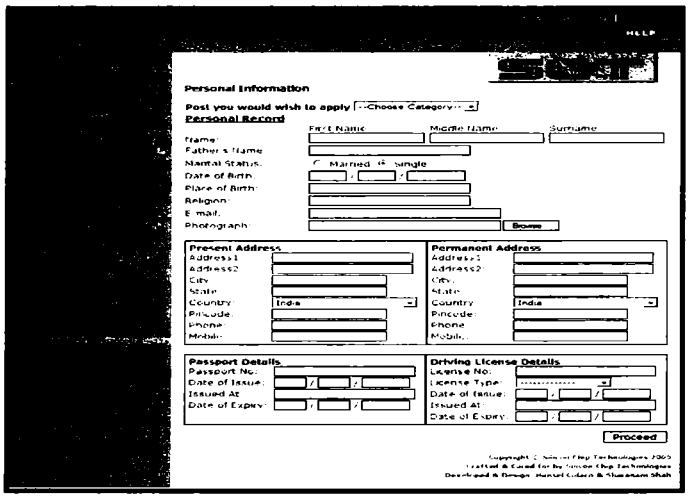


Diagram 18.18.2: The Candidate's Personal Information page.

3. cand_qual.php

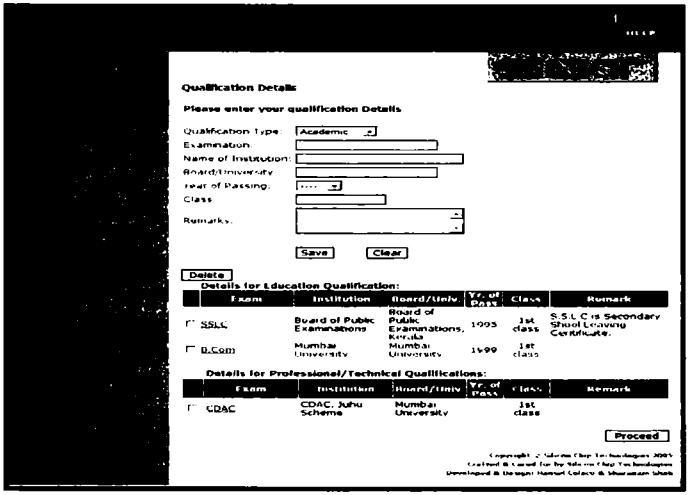


Diagram 18.18.3: The Candidate's Qualification Details page.

4. cand_empl.php

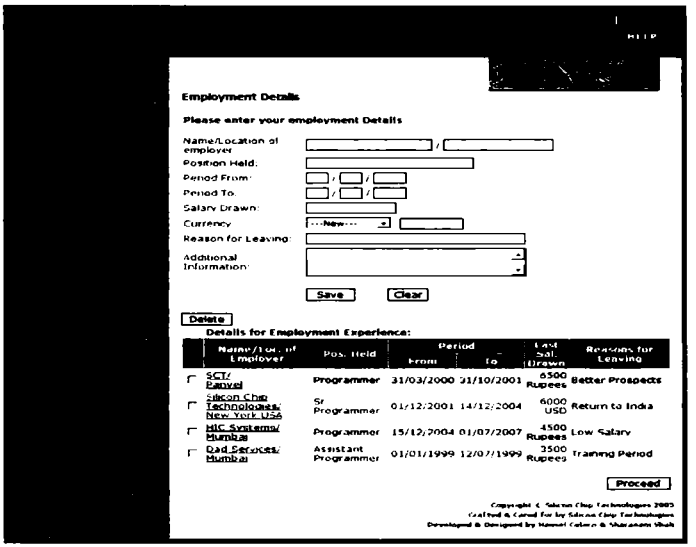


Diagram 18.18.4: The Candidate's Employment Details page.

5. cand_lang.php

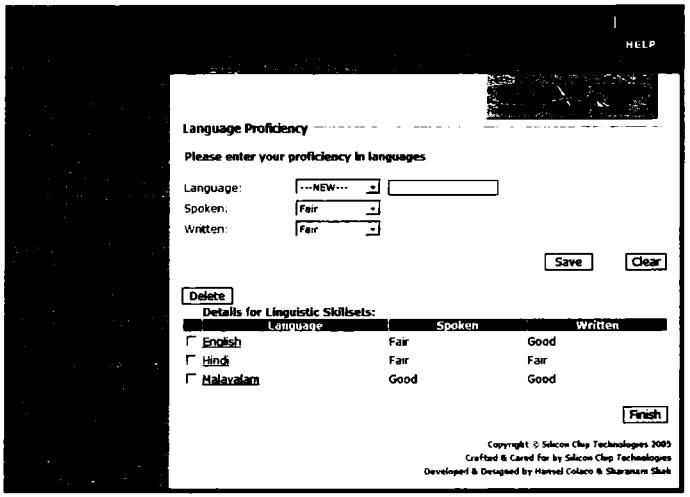


Diagram 18.18.5: The Candidate's Language Proficiency page.

6. cand_main.php

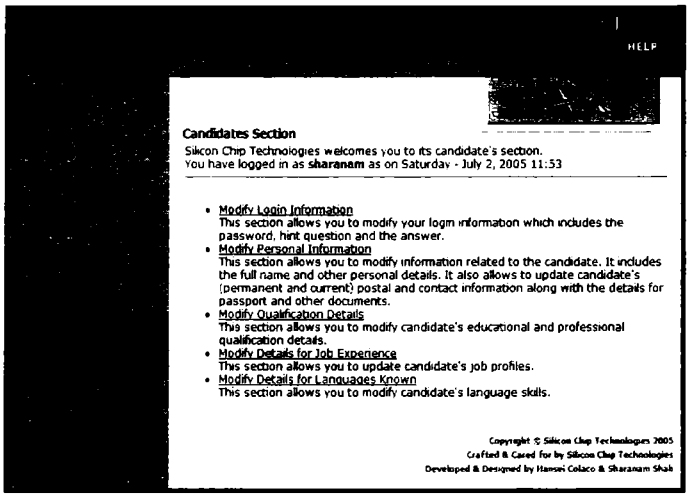


Diagram 18.18.6: The Candidate's Home page.

The LOGIN Process Page Flow

1. cand_main.php

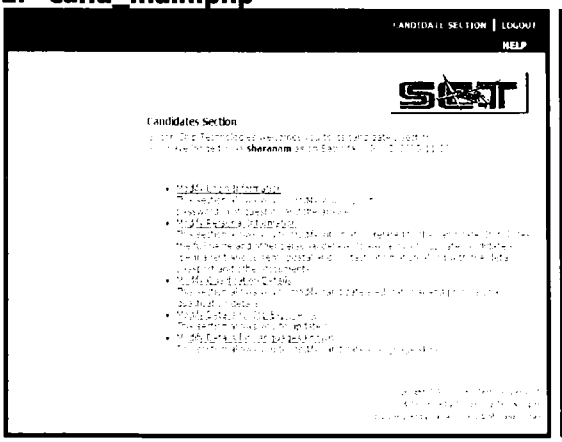


Diagram 18.19.1: The Candidate's Home page – Complete Resume.

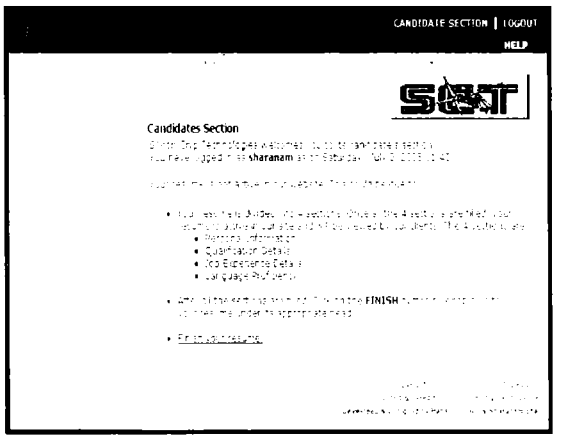


Diagram 18.19.2: The Candidate's Home page – Incomplete Resume.

Here the control is passed to the **cand_main** page, which will render itself in either of the states as seen in diagram 18.19.1 or diagram 18.19.2. The page as seen in diagram 18.19.1 appears if the user has completed the registration in terms of resume details in all required aspects.

The pages under the **CANDIDATE** module:

1. **cand_reg.php** – Allows registering as a candidate. Control is shifted to this page when the user selects the SIGNUP → Candidate option. This page is available under **/candidates/cand_reg.php**

Layout - Candidate Registration Page Of The CANDIDATE Module (cand_reg.php)

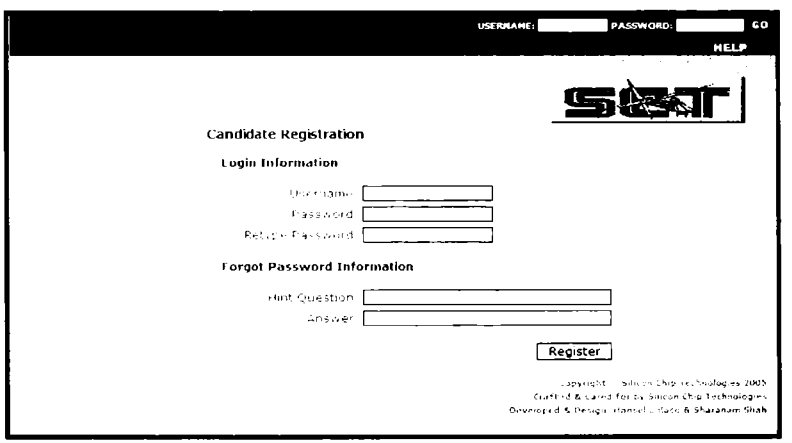


Diagram 18.20

Purpose

- Captures the login information such as username, password and hint question/answer
- Stores the login information after validating it in the database
- Allows passing the control to the **cand_pers.php** page after the login information is validated and stored

GENERAL PAGE SPECIFICATIONS FOR cand_reg.php

FILES INCLUDED IN cand_reg.php:

PHP CODE BLOCK:	functions.php	Declares PHP based user-defined functions
	header.php	Generates the top bar for the HTML page
	footer.php	Generates the footer bar for the HTML page
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN cand_reg.php:

JavaScript:	**PHP:**
frmValidate()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	**LABEL**	**NAME**	**DESCRIPTION**
N/A	N/A	N/A	N/A

FORM DETAILS:

FORM NAME: frmCandReg	**BOUND TO:** USERS	**ACTION:** dosql.php

DATA FIELDS:

OBJECT	**LABEL**	**NAME**	**BOUND TO**
Hidden	N/A	hidPage	
Text Box	Username	txtUsername	USER_USERNAME
Text Box	Password	txtPassword	USER_PASSWORD
Text Box	Retype Password	txtRepassword	N/A
Text Box	Hint Question	txtQuestion	FOR_PASS_QUESTION
Text Box	Answer	txtAnswer	FOR_PASS_ANSWER

DATA CONTROLS:

OBJECT	**LABEL**	**NAME**	**ACTION**
Button	Register	subForm	JavaScript:onSubmit()

Processing - Candidate Registration Page Of The CANDIDATE Module (cand_reg.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A variable named **helppage** is defined to store the name of the HTML page that will serve help on demand
- The following two files are included:
  - /includes/functions.php
    - This file is included to make available the functions defined
  - /includes/header.php
    - This file is included to make the top bar available in the **cand_reg.php**. Refer diagram 18.21.1

Diagram 18.21.1

- Java script code-spec follows to validate username and password (including the retyped password) captured by this page. This is done using a function named **frmValidate**. This function is called when the information captured by this page is submitted using the Register button. This means a client side validation is performed before validating the username/password (including the retyped password) against the database at the server end
- Initial HTML code-spec corresponding to the template begins
- HTML code-spec for designing the layout of the form in terms of textboxes and register button responsible to capture the user name, password, retyped password, Hint question and answer follows
- This includes a FORM tag definition wherein the **ACTION** attribute of the FORM tag is pointed to the **dosql.php** file and the **OnSubmit** attribute is pointed to the **frmValidate** java script function
  *The **dosql.php** file (explained later) does the actual processing bit for this page in terms of insertion of the data captured by this page*
- Finally the **footer.php** file is included to render the **copyright and the developer's information** on the **cand_reg.php** page. Refer Diagram 18.21.2

Diagram 18.21.2

2. **cand_main.php** – This page appears after the user logs in as a candidate. This means that the **logincheck.php** page passes control to this page. It provides links to pages that will allow modifying the data captured while registering. The page appearance differs based on the resume details status in terms of complete/incomplete. This page is available under **/candidates/cand_main.php**

Layout - Main Page Of The CANDIDATE Module (cand_main.php)

Purpose

- Appears as the first page after the user logs in as a candidate
- Provides login details such as the user who has logged in and the date-time when the user logged in
- Provide links to pages allowing modification of data entered while registration process
- Intimate the candidate on an incomplete resume

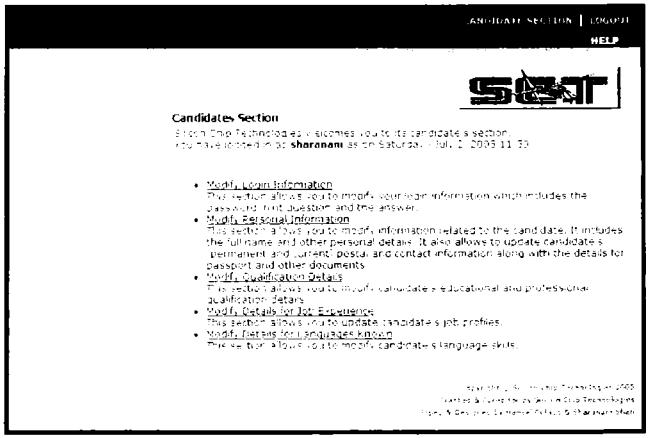


Diagram 18.22

GENERAL PAGE SPECIFICATIONS FOR cand_main.php

FILES INCLUDED IN cand_main.php:

PHP CODE BLOCK:	functions.php	Declares PHP based user-defined functions
	header.php	Generates the top bar for the HTML page
	footer.php	Generates the footer bar for the HTML page
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN cand_main.php:

JavaScript:	PHP:
N/A	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Modify Login Information	N/A	chng_login.php?hidEditID=<current_UserID>
Hyperlink	Modify Personal Information	N/A	chng_pers.php?hidEditID=<current_UserID>
Hyperlink	Modify Qualification Details	N/A	chng_qual.php?hidEditID=<current_UserID>
Hyperlink	Modify Details for Job Experience	N/A	chng_empl.php?hidEditID=<current_UserID>
Hyperlink	Modify Details for Languages Known	N/A	chng_lang.php?hidEditID=<current_UserID>

FORM DETAILS:

FORM NAME: N/A	**BOUND TO:** N/A	**ACTION:** N/A

DATA FIELDS:

OBJECT	**LABEL**	**NAME**	**BOUND TO**
N/A	N/A	N/A	N/A

DATA CONTROLS:

OBJECT	**LABEL**	**NAME**	**ACTION**
N/A	N/A	N/A	N/A

Processing - Main Page Of The CANDIDATE Module (cand_main.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A variable named **auth_reqd** is defined holding value **YES** indicating that no user can access this page directly without having logged in as a candidate.
- A variable named **helppage** is defined to store the name of the HTML page that will serve help on demand
- The following two files are included:
  - /includes/functions.php
    - This file is included to make available the functions defined
  - /includes/header.php
    - This file is included to make the top bar available in the **cand_main.php**. Refer diagram 18.23.1

Diagram 18.23.1

- An SQL query is formed and executed to retrieve candidate identity (using the CAND_ID field), the registration status (using the RGST_COMPLETE field) in terms of its completion, the Level of completion (using the IS_COMPLETE field) in terms of the incomplete pages and the user name (using the USER_USERNAME field) of the candidate logged in
  Based on the data retrieved by SQL query the appearance of the page is determined
- Initial HTML code-spec corresponding to the template begins
- HTML code-spec, to design, an HTML page that will be rendered when data retrieved by SQL query indicates that registration is complete follows. This includes:

- Displaying a welcome message along with **user name** and the **login date** and **time**
- A check is made if the registration process is complete. If it is complete then: (Refer Diagram 18.23.2)
  - HTML code-spec to design the page with links to other pages allowing modifications follows
    Every link created is passed the candidate identity as:
    `?hidEditID=<?=$rslt_arr[0];?>`
    hidEditID *is a variable accepted by the pages to be linked to.* ***rslt_arr[0]*** *is the first element of the result set holding the candidate identity retrieved by SQL query.*
    The value held by the variable ***hidEditID*** *decides the details of the candidate that need to be rendered when the control shifts to that page*
- If the registration is incomplete then: (Refer Diagram 18.23.3)
  - The page that holds incomplete data is determined.
    This is done based on the value held by the ***IS_COMPLETE*** *field. This field will store 0 – indicates* ***cand_pers.php*** *page, 1 – indicates* ***cand_qual.php*** *page, 2 – indicates* ***cand_empl.php*** *page and 3/4 – indicates* ***cand_lang.php*** *page*
  - This is now followed by some text matter which gives an explanation of resume section breakup

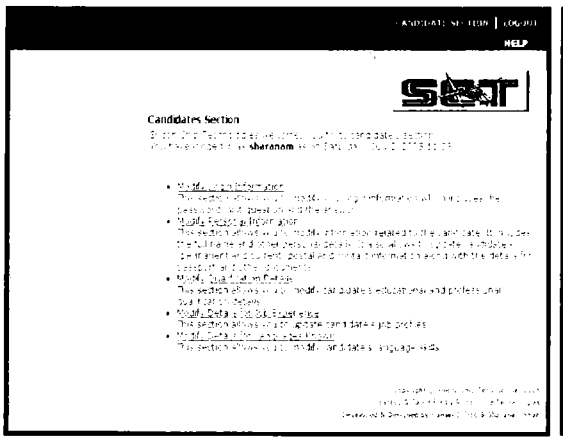


Diagram 18.23.2: Complete Resume.

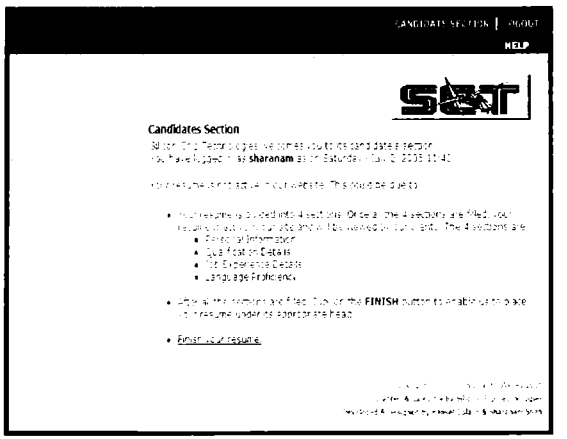


Diagram 18.23.3: Incomplete Resume.

- Finally the **footer.php** file is included to render the **copyright and the developer's information** on the **cand_reg.php** page. Refer Diagram 18.23.4

Copyright © Silicon Chip Technologies 2005
Crafted & Cared for by Silicon Chip Technologies
Developed & Designed by Hansel Colaco & Sharanam Shah

Diagram 18.23.4

3. **cand_pers.php** – This page provides an interface for managing candidate personal information stored in the CANDIDATE, CAND_ADDR and CAND_DOC tables. This form allows capturing (Insert) and also modifying (Update) one time information. This means that multiple entries will not be allowed. It is available under **/candidates/cand_pers.php**

Layout - The Page Managing Candidate's Personal Information (cand_pers.php)

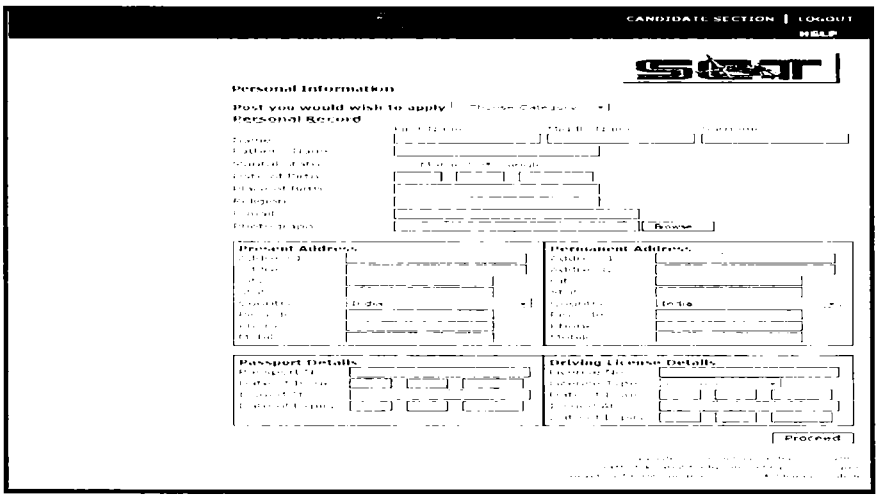


Diagram 18.24

Purpose

- Capture Candidate's personal information
- Update information captured
- Allows proceeding to the Candidate's Qualification page

GENERAL PAGE SPECIFICATIONS FOR cand_pers.php

FILES INCLUDED IN cand_pers.php:

PHP CODE BLOCK:	functions.php	Declares PHP based user-defined functions
	header.php	Generates the top bar for the HTML page
	footer.php	Generates the footer bar for the HTML page
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN cand_pers.php:

JavaScript:	**PHP:**
vldtFrmFlds()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Modify Qualification Details	N/A	chng_qual.php?hidEditID=<current_UserID>
Hyperlink	Modify Experience Details	N/A	chng_empl.php?hidEditID=<current_UserID>
Hyperlink	Modify Languages Details	N/A	chng_lang.php?hidEditID=<current_UserID>

FORM DETAILS:

FORM NAME: frmCandPers	**BOUND TO:** CANDIDATE, CAND_ADDR, CAND_DOC	**ACTION:** dosql.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Hidden	N/A	hidPage	N/A
Hidden	N/A	hidEditID	USERS.USER_ID
Hidden	N/A	IsComplete	CANDIDATE.ISCOMPLETE
Select Box	Post you would wish to apply	selCategory	CANDIDATE.CAND_CATEGORY
Text Box	First Name	txtFirstName	CANDIDATE.CAND_FIRST_NAME
Text Box	Middle Name	txtMiddleName	CANDIDATE.CAND_MIDDLE_NAME
Text Box	Surname	txtSurname	CANDIDATE.CAND_SURNAME
Text Box	Father's Name	txtFatherName	CANDIDATE.CAND_FATHER_NAME
Radio Button	Marital Status	rdoMarital	CANDIDATE.CAND_MARITAL_STATUS
Text Box	Date of Birth	txtDOB	CANDIDATE.CAND_BIRTHDATE
Text Box		txtDOB1	
Text Box		txtDOB2	
Text Box	Place of Birth	txtBirthPlace	CANDIDATE.CAND_BIRTHPLACE
Text Box	Religion	txtReligion	CANDIDATE.CAND_RELIGION
Text Box	E-mail	txtEmail	CANDIDATE.CAND_EMAIL
Text Box	Photograph	txtPhoto	CANDIDATE.CAND_PIC
Hidden	Present Address	hidPreAddrID	CAND_ADDR.CAND_ADDR_ID
Text Box	Address1	txtPreAddr1	CAND_ADDR.CAND_ADDR1
Text Box	Address2	txtPreAddr2	CAND_ADDR.CAND_ADDR2
Text Box	City	txtPreCity	CAND_ADDR.CAND_CITY
Text Box	State	txtPreState	CAND_ADDR.CAND_STATE
Select Box	Country	selPreCountry	CAND_ADDR.CAND_COUNTRY
Text Box	Pincode	txtPrePin	CAND_ADDR.CAND_ZIP
Text Box	Phone	txtPrePhone	CAND_ADDR.CAND_TEL
Text Box	Mobile	txtPreMob	CAND_ADDR.CAND_MOB

DATA FIELDS: (Continued)

OBJECT	LABEL	NAME	BOUND TO
Hidden	Permanent Address	hidPerAddrID	CAND_ADDR.CAND_ADDR_ID
Text Box	Address1	txtPerAddr1	CAND_ADDR.CAND_ADDR1
Text Box	Address2	txtPerAddr2	CAND_ADDR.CAND_ADDR2
Text Box	City	txtPerCity	CAND_ADDR.CAND_CITY
Text Box	State	txtPerState	CAND_ADDR.CAND_STATE
Select Box	Country	selPerCountry	CAND_ADDR.CAND_COUNTRY
Text Box	Pincode	txtPerPin	CAND_ADDR.CAND_ZIP
Text Box	Phone	txtPerPhone	CAND_ADDR.CAND_TEL
Text Box	Mobile	txtPerMob	CAND_ADDR.CAND_MOB
Hidden	Passport Details	hidPass	N/A
Text Box	Passport No	txtPassNo	CAND_DOC.CAND_DOC_NO
Text Box	Date of Issue	txtDOI	CAND_DOC.CAND_DOC_ISSUE_DAT
Text Box		txtDOI1	
Text Box		txtDOI2	
Text Box	Issued At	txtIssuedAt	CAND_DOC.CAND_DOC_ISSUE_PLACE
Text Box	Date Of Expiry	txtDOE	CAND_DOC.CAND_DOC_EXP_DAT
Text Box		txtDOE1	
Text Box		txtDOE2	
Hidden	Driving License Details	hidLic	N/A
Text Box	License No	txtLicNo	CAND_DOC.CAND_DOC_NO
Select Box	License Type	selLicType	CAND_DOC.CAND_DOC_TYPE
Text Box	Date of Issue	txtLDOI	CAND_DOC.CAND_DOC_ISSUE_DAT
Text Box		txtLDOI1	
Text Box		txtLDOI2	
Text Box	Issued At	txtIssuedAt	CAND_DOC.CAND_DOC_ISSUE_PLACE
Text Box	Date Of Expiry	txtLDOE	CAND_DOC.CAND_DOC_EXP_DAT
Text Box		txtLDOE1	
Text Box		txtLDOE2	

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Button	Proceed	subForm	JavaScript:onSubmit()

Processing - The Page Managing Candidate's Personal Information (cand_pers.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page

- A check if the variable **hidEditID** holds a value. If it is empty then the control passes to the **cand_reg.php** page. **hidEditID** being empty simply indicates that the candidate has not registered and thus does not have a candidate identity number
- A variable named **auth_reqd** is defined holding value **YES** indicating that no user can access this page directly without having logged in as a candidate.
- A variable named **helppage** is defined to store the name of the HTML page that will serve help on demand
- The following files are included:
  - /includes/functions.php
    - This file is included to make available the functions defined
  - /includes/header.php
    - This file is included to make the top bar available in the **cand_reg.php**. Refer diagram 18.25.1

Diagram 18.25.1

- Codespec to form the view mode follows:
- An SQL query is formed and executed. This query will **validate the existence** of the candidate identity received via the variable **hidEditID** in the **CANDIDATE** table
- If the SQL query returns any records then:
  - Data retrieved is transferred from the record-set to the form variables. This action actually populates the form fields
- An SQL query is formed and executed. This query will **validate the existence** of the candidate identity received via the variable **hidEditID** in the **CAND_ADDR** table
- If the SQL query returns any records then:
  - Data retrieved is transferred from the record-set to the form variables. This action actually populates the form fields
- An SQL query is formed and executed. This query will **validate the existence** of the candidate identity received via the variable **hidEditID** in the **CAND_DOC** table
- If the SQL query returns any records then:
  - Data retrieved is transferred from the record-set to the form variables. This action actually populates the form fields
- An SQL query is formed and executed. This query will retrieve the country identity for the country INDIA. This is done to have **India** selected as the default country when the form is in INSERT MODE. This is done as:

- A check is made on the value held by the select box in both the sections (i.e. Present Address and Permanent Address). If the select boxes hold nothing (i.e. have no current selection) then the value of select boxes is set to the current identity retrieved by the SQL query

- This is followed by a java script function:
  - **vldtFrmFlds** – This function is called when the form data is about to be submitted. This function verifies that mandatory form fields are not blank, dates entered are valid dates and so on. If an error is encountered then an error message indicates the same.
- Initial HTML code-spec corresponding to the template begins
- Links are generated to all the candidate module pages if the registration procedure is complete. This is verified using the value held by the variable **IS_COMPLETE**
- This is followed by the FORM tag which holds **ACTION** pointing to the **dosql.php** file and **onSubmit** pointing to the **vldtFrmFlds** function. The dosql.php file performs the form operations in terms of insert/update/delete. The control passes on click of the **Proceed** button
- Following hidden variables are declared using the INPUT tags:
  - **hidPage** – Holds the **page name** in this case **cand_pers.php**
  - **hidEditId** – Holds **Candidate Identity** for update operations
  - **IS_COMPLETE** – Is used to determine the page which is incomplete
  - **hidPreAddrID** – Holds **Address Identity** for update operations in this case Present Address
  - **hidPerAddrID** – Holds **Address Identity** for update operations in this case Permanent Address
  - **hidPass** – Holds **Document Identity** for update operations in this case Passport
  - **hidLic** – Holds **Document Identity** for update operations in this case Driving License
- HTML Code-spec follows for designing the form
- Finally the **footer.php** file is included to render the **copyright and the developer's information** on the **cand_pers.php** page. Refer Diagram 18.25.2

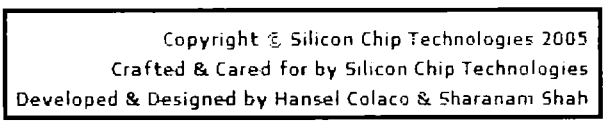


Diagram 18.25.2

4. **cand_qual.php**– This page provides an interface for managing candidate's qualification details stored in the **CAND_QUAL** table. This form allows capturing (Insert), modifying (Update) and deleting data. It is available under **/candidates/cand_qual.php**

Layout - The Page Managing Candidate's Qualification Information (cand_qual.php)

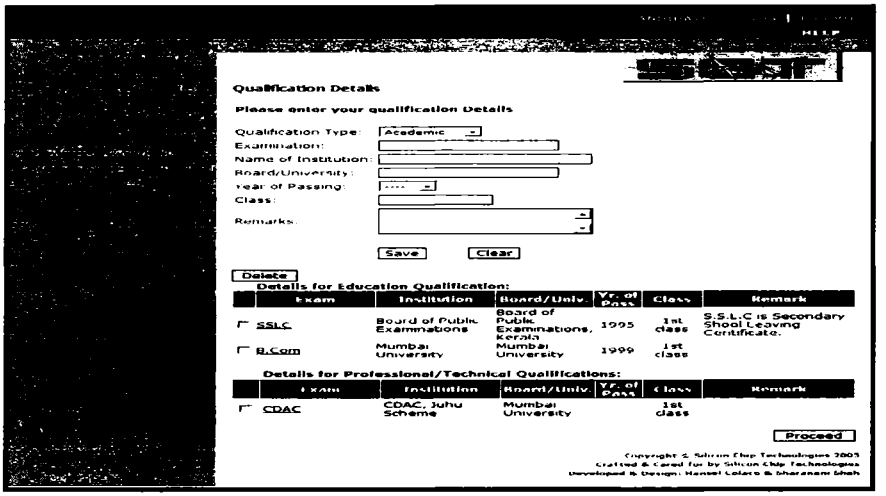


Diagram 18.26.1

Purpose

- Capture Candidate's qualification information
- Update / Delete information captured
- Allows proceeding to the Candidate's Employment page

GENERAL PAGE SPECIFICATIONS FOR cand_qual.php

FILES INCLUDED IN cand_qual.php:

PHP CODE BLOCK:	functions.php	Declares PHP based user-defined functions
	header.php	Generates the top bar for the HTML page
	footer.php	Generates the footer bar for the HTML page
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN cand_qual.php:

JavaScript:	PHP:
clrFrmFlds(), delRcrd(), vldtFrmFlds(), proceedTo()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Modify Personal Info.	N/A	chng_pers.php?hidEditID=<current_UserID>
Hyperlink	Modify Experience Details	N/A	chng_empl.php?hidEditID=<current_UserID>
Hyperlink	Modify Languages Details	N/A	chng_lang.php?hidEditID=<current_UserID>

FORM DETAILS:

FORM NAME: frmCandQual	BOUND TO: CAND_QUAL	ACTION: dosql.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Hidden	N/A	hidPage	N/A
Hidden	N/A	rgst_complete	CANDIDATE.RGST_COMPLETE
Hidden	N/A	hidEditID	USERS.USER_ID
Hidden	N/A	hidQualID	CAND_QUAL.QUAL_ID
Hidden	N/A	hidCandID	CAND_QUAL.CAND_ID
Hidden	N/A	hidQualType	CAND_QUAL.QUAL_TYPE
Hidden	N/A	hidExam	CAND_QUAL.EXAM_NAME
Hidden	N/A	hidInst	CAND_QUAL.INSTITUTION_NAME
Hidden	N/A	hidBoard	CAND_QUAL.BOARD_NAME
Hidden	N/A	hidPassYr	CAND_QUAL.YEAR_OF_PASSING
Hidden	N/A	hidClass	CAND_QUAL.CLASS
Hidden	N/A	hidRmrk	CAND_QUAL.REMARKS
Hidden	N/A	hidMode	N/A
Hidden	N/A	hidDelRcrdLst	N/A
Select Box	Qualification Type	cboQualType	hidQualType
Text Box	Examination	txtExam	hidExam
Text Box	Name of Institution	txtInst	hidInst
Text Box	Board/University	txtBoard	hidBoard
Text Box	Year of Passing	cboPassYr	hidPassYr
Text Box	Class	txtClass	hidClass
Text Box	Remarks	txtRmrk	hidRmrk
Checkbox	N/A	chk1, chk2, . . .	N/A
Hyperlink	<Academic Qualification List>	N/A	cand_qual.php?hidEditID=<Current Candidate ID>&hidQualID=<Current Qualification ID>
Checkbox	N/A	chk1, chk2, . . .	N/A
Hyperlink	<Non-Academic Qualification List>	N/A	cand_qual.php?hidEditID=<Current Candidate ID>&hidQualID=<Current Qualification ID>

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Button	Save	cmdSbmt	JavaScript:onSubmit()
Button	Clear	cmdClr	JavaScript:clrFrmFlds()
Button	Delete	cmdDelete1	JavaScript:delRcrd()
Button	Delete	cmdDelete2	JavaScript:delRcrd()
Button	Proceed	cmdProceed	JavaScript:proceedTo()

Processing - The Page Managing Candidate's Qualification Information (cand_qual.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A check if the variable **hidEditID** holds a value. If it is empty then the control passes to the **cand_main.php** page. The **cand_main.php** page thus when rendered with **header.php** file included fires an error indicating either to register or provides an option to login (Refer diagram 18.26.2). **hidEditID** being empty simply indicates that the candidate has not logged in and thus the candidate identity number is not available

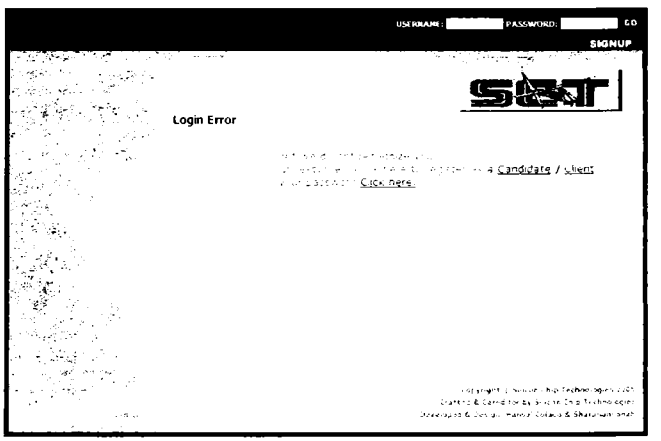


Diagram 18.26.2

*The variable **hidEditID** is populated by either the **dosql.php** file or the other files such as cand_empl.php, cand_lang.php, cand_qual.php, cand_pers.php and cand_main.php files via the Links (Href). This variable holds the **CANDIDATE_ID** of the candidate **currently logged in**. This is how the **cand_qual.php** file understands what to display when control comes to it. This variable is used across all php pages in the candidate module.*

- A variable named **auth_reqd** is defined holding value **YES** indicating that no user can access this page directly without having logged in as a candidate
- A variable named **heippage** is defined to store the name of the HTML page that will serve help on demand
- A variable named **disButton** is initialized to hold the status of the **Proceed** button code-spec in terms of **enabled/disabled**
  This is done to determine the appearance of the Proceed button. This button is disabled when there are no records available in the tabular layout

- A variable named **delEnabled** is initialized with a blank string. This variable will hold the status of **Delete** button code-spec in terms of **enabled/disabled**
  This is done to determine the appearance of the Delete button. This button is disabled when there are no records available in the tabular layout
- A variable named **hidMode** is initialized to **I**. This variable hold the form status in terms of **I** - **I**nsert, **U** - **U**pdate and **D** – **D**elete
- A variable **count** is initialized to hold **zero**. This is used in **while loops** to generate **unique checkbox names (e.g.** chk1, chk2 and so on**)** for the tabular layout
- A check is made to determine whether to include the **functions.php** file. This is done to avoid re-inclusion of the same file as follows**:**
  - If **func_incl** holds the value **YES** the inclusion of the **function.php** file is skipped. This means that the **functions.php** file is already included earlier. This inclusion is done by the **dosql.php** file.
- The following file is included**:**
  - /includes/header.php
    - This file is included to make the top bar available in the **cand_qual.php**. Refer diagram 18.27.1

Diagram 18.27.1

- The default value for the qualification type select box is set to **Academic**
- A variable **curYr** is initialized to hold the **current year**. This is used to populate the **Year Of Passing** select box
- A check is made to verify if the variable **hidQualID** holds a value
  *This variable will hold a valid value, on selection of a record for editing it, from the tabular layout. This variable holds the **QUAL_ID** chosen for editing. This is how the **cand_qual.php** file understands which record is to be displayed when in edit mode. This check determines the view/update mode*
- Codespec to form the view mode follows**:**
- An SQL query is formed and executed. This query will **validate the existence** of the candidate identity and the qualification identity received via the variable **hidEditID** and **hidQualID respectively** in the **CAND_QUAL** table
- If the SQL query returns any records then**:**
  - Data retrieved is transferred from the record-set to the hidden form variables
  - A variable named **delEnabled** is assigned a value, (i.e. disabled=true). This value indicates the delete button status code-spec in terms of **enabled/disabled**
  - The form mode is switched to Update by setting the value of variable **hidMode** to **U**

- A check is made if the variable **msg** holds a value. This check displays an error message when duplication while inserting the data captured by the **cand_qual.php** file into the database is encountered
  *The value held by the variable **msg** can either be a **one dimensional** or **two dimensional** (in this case). In case of **one-dimensional** the value held would be an error message string, which has to be displayed.*
  Examples:
  1. **Please enter a valid username and password.**
  2. **Select an appropriate Currency.**
  3. **The username you entered has already been chosen by another user. Please enter a different username.**
  4. **Language already exists.**

  *In case of **two-dimensional** the first value (Dimension) will be an error number and the second value (Dimension) will be some value sent back as a duplicate entry in database. This means the first column holds **–1** to indicate that an error has occurred and the second parameter holds the Exam Name (in this case) which will be a duplicate value which will be a part of the error message.*
  Examples:
  1. **Details for <Exam Name> already exists. Please modify the information, if required.**

  *This variable is populated by the **dosql.php** file with an appropriate error message with/without the error number as the case may be. It is populated to transfer any errors if they occur during the processing carried out by the **dosql.php** file on behalf of the **cand_qual.php** file*
- If the variable **msg** holds a value then**:**
  - The value held is split into **two** portions in this case the **error number** i.e. **–1** and the **exam name**
  - Based on the number of dimensions, the **msg** variable holds error message is generated and populated in a variable namely **Message**
- This is followed by a set of java script functions**:**
  - **clrFrmFlds** – This function is called when a clear button is clicked and performs the following operations**:**
    - Clears the value held by the hidden variable defined
    - Sets the default value for the Qualification Type select box
    - Clears the value held by the form fields
    - Switches the form mode back to insert. This is done by setting the value of hidden variable named **hidMode** to **I**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation

- Enables the Delete buttons available on the **cand_qual.php** form

- **delRcrd** – This function is called when delete button is clicked and performs the following operations:
  - Calls the **clrFrmFlds** function to reset/clear the variables
  - Switches the form mode to delete. This is done by setting the value of hidden variable named **hidMode** to **D**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
  - Calls a function named **formDeleteValues** available in the **/includes/rollover.js** file with a parameter to hold list of qualification identities for deletion. This function generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same

- **vldtFrmFlds** – This function is called when the form data is about to be submitted. This function verifies that mandatory form fields are not blank, dates entered are valid dates and so on. If an error is encountered then an error message indicates the same. These validations are performed only if the form is not in Delete mode. When the form is in Update mode, a check is carried out which will verify that there were changes made on the form. This is done by a comparison between the form field values and the hidden variables (holding the values prior update). This additional check is carried out just to avoid unnecessary submission to the **dosql.php** file which does the actual database update

- **proceedTo** – This function is called when Proceed button is clicked. It is responsible to pass the control to the **cand_empl.php** file:
  - Points the action attribute of the form tag to the **cand_empl.php** file
  - Submits the form to implement the change

- Initial HTML code-spec corresponding to the template begins
- An SQL query is formed and executed. This query is responsible to extract the **IS_COMPLETE** and **RGST_COMPLETE** values from the **CANDIDATE** table
- Links are generated to all the candidate module pages if the registration procedure is complete. This is verified using the value held by the variable **IS_COMPLETE**
- This is followed by the FORM tag, which holds **ACTION** pointing to the **dosql.php** file and **onSubmit** pointing to the **vldtFrmFlds** function. The **dosql.php** file performs the form operations in terms of insert/update/delete. The control passes on click of the **Proceed** button
- Following hidden variables are declared using the INPUT tags:
  - **hidPage** – Holds the **page name** in this case **cand_qual.php**
  - **hidMode –** Holds the form mode in terms of **I**nsert, **U**pdate or **D**elete
  - **RGST_COMPLETE** – Is used to determine the registration status

- **hidDelRcrdLst** – Holds a list of identities to be deleted
- **hidEditId** – Holds **Candidate Identity** for update operations
- **hidQualId** – Holds **Candidate's Qualification Identity** for update operations
  Following hidden variables are defined as a copy of the form fields. These will hold values prior updates and will be used by the **vldtFrmFlds** function to check (by comparing) if any modifications were performed on the form before passing the control to the **dosql.php** file for performing database operations
- **hidCandID**
- **hidQualType**
- **hidExam**
- **hidInst**
- **hidBoard**
- **hidPassYr**
- **hidClass**
- **hidRmrk**

- HTML Code-spec follows for designing the form
- An SQL query is formed and executed. This query retrieves the data belonging to the **Academic section** from the **CAND_QUAL** table based on the **Candidate Identity**. The data retrieved will be used to form the tabular layout belonging to the Academic section
- If the SQL query retrieves any records following operations are performed**:**
  - HTML Code-spec follows for creation of [Delete] button which has **onClick** attribute set to a function call named **DelRcrd**
  - A tabular layout is created to hold the records retrieved. This is done using a **do while** loop, which traverses through the records retrieved from the table via the SQL query. While traversing through the loop the color code for each row displayed in the tabular layout is switched between two different colors. This is done to improve the readability
  - The **Exam** as seen in the diagram 18.27.2 is assigned a link to the same page i.e. **cand_qual.php** by passing two parameters**: hidEditID** and **hidQualID**. Based on the value held by these parameters the form fields are populated using an SQL query (explained earlier in Codespec to form the view mode follows point)

Delete

Details for Education Qualification:

	Exam	Institution	Board/Univ.	Yr. of Pass	Class	Remark
☐	SSLC	Board of Public Examinations	Board of Public Examinations, Kerala	1995	1st class	S.S.L.C is Secondary Shool Leaving Ceritificate.
☐	B.Com	Mumbai University	Mumbai University	1999	1st class	

Diagram 18.27.2

- An SQL query is formed and executed. This query retrieves the data belonging to the **Non-Academic section** from the **CAND_QUAL** table based on the **Candidate Identity**. The data retrieved will be used to form the tabular layout belonging to the Academic section
- If the SQL query retrieves any records following operations are performed:
  - HTML Code-spec follows for creation of delete button which has **onClick** attribute set to a function call named **DelRcrd**
  - A tabular layout is created to hold the records retrieved. This is done using a **do while** loop, which traverses through the records retrieved from the table via the SQL query. While traversing through the loop the color code for each row displayed in the tabular layout is switched between two different colors. This is done to improve the readability
  - The **Exam** as seen in the diagram 18.27.3 is assigned a link to the same page i.e. **cand_qual.php** by passing two parameters: **hidEditID** and **hidQualID**. Based on the value held by these parameters the form fields are populated using an SQL query (explained earlier in Codespec to form the view mode follows point)

Details for Professional/Technical Qualifications:

	Exam	Institution	Board/Univ.	Yr. of Pass	Class	Remark
☐	CDAC	CDAC, Juhu Scheme	Mumbai University		1st class	

Diagram 18.27.3

- Finally the **footer.php** file is included to render the **copyright and the developer's information** on the **cand_qual.php** page. Refer Diagram 18.27.4

Diagram 18.27.4

5. **cand_empl.php** – This page provides an interface for managing candidate's employment details stored in the **CAND_EMPL** table. This form allows capturing (Insert), modifying (Update) and deleting data. It is available under **/candidates/cand_empl.php**

Layout - The Page Managing Candidate's Employment Information (cand_empl.php)

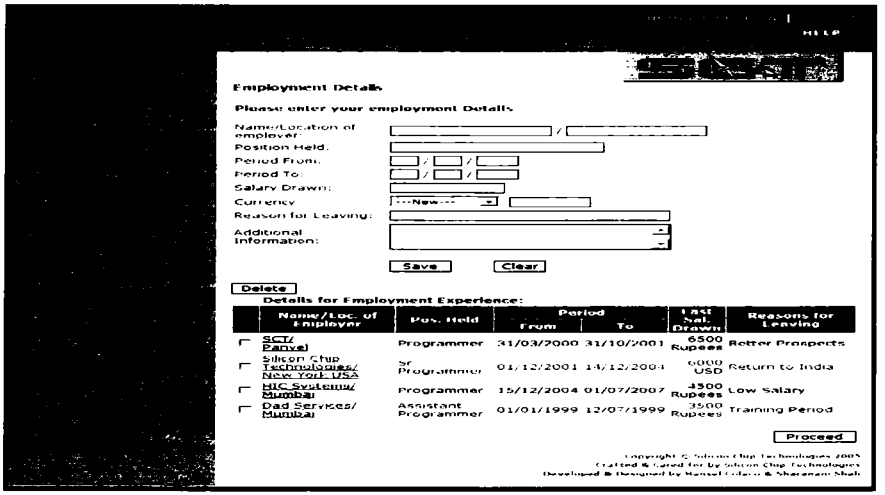


Diagram 18.28.1

Purpose

- Capture Candidate's employment information
- Update / Delete information captured
- Allows proceeding to the Candidate's Language page

GENERAL PAGE SPECIFICATIONS FOR cand_empl

FILES INCLUDED IN cand_empl.php:

PHP CODE BLOCK:	functions.php	Declares PHP based user-defined functions
	header.php	Generates the top bar for the HTML page
	footer.php	Generates the footer bar for the HTML page
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN cand_empl.php:

JavaScript:	PHP:
valCombo(), clrFrmFlds(), delRcrd(), ChngLen(), vldtFrmFlds(), proceedTo()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Modify Personal Info.	N/A	chng_pers.php?hidEditID=<current_UserID>
Hyperlink	Modify Qualification Details	N/A	chng_qual.php?hidEditID=<current_UserID>
Hyperlink	Modify Languages Details	N/A	chng_lang.php?hidEditID=<current_UserID>

FORM DETAILS:

FORM NAME: frmCandEmpl	**BOUND TO:** CAND_EMPL	**ACTION:** dosql.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Hidden	N/A	hidPage	N/A
Hidden	N/A	rgst_complete	CANDIDATE.RGST_COMPLETE
Hidden	N/A	hidEditID	USERS.USER_ID
Hidden	N/A	hidEmpID	CAND_EMPL.EMP_ID
Hidden	N/A	hidCandID	CAND_EMPL.CAND_ID
Hidden	N/A	hidEmp	CAND_EMPL.EMP_NAME
Hidden	N/A	hidLoc	CAND_EMPL.EMP_LOC
Hidden	N/A	hidPosHeld	CAND_EMPL.POS_HELD
Hidden	N/A	hidPerFrom	CAND_EMPL.PER_FROM
Hidden	N/A	hidPerFrom1	
Hidden	N/A	hidPerFrom2	
Hidden	N/A	hidPerTo	CAND_EMPL.PER_TO
Hidden	N/A	hidPerTo1	
Hidden	N/A	hidPerTo2	
Hidden	N/A	hidSalary	CAND_EMPL.SAL_ON_LEAVING
Hidden	N/A	hidSelCurrency	CAND_EMPL.CURRENCY
Hidden	N/A	hidTxtCurrency	N/A
Hidden	N/A	hidReason	CAND_EMPL.REASON_FOR_LEAVING
Hidden	N/A	hidAddl	CAND_EMPL.REMARKS
Hidden	N/A	hidMode	N/A
Hidden	N/A	hidDelRcrdLst	N/A
Text Box	Name/Location of employer	txtEmp	hidEmp
Text Box		txtLoc	hidLoc
Text Box	Position Held	txtPosHeld	hidPosHeld
Text Box	Period From	txtPerFrom	hidPerFrom
Text Box		txtPerFrom1	hidPerFrom1
Text Box		txtPerFrom2	hidPerFrom2

DATA FIELDS: (Continued)

OBJECT	LABEL	NAME	BOUND TO
Text Box	Period To	txtPerTo	hidPerTo
Text Box		txtPerTo1	hidPerTo1
Text Box		txtPerTo2	hidPerTo2
Text Box	Salary Drawn	txtSalary	hidSalary
Select Box	Currency	selCurrency	hidSelCurrency
Text Box			hidTxtCurrency
Text Box	Reason for Leaving	txtReason	hidReason
Text Box	Additional Information	TxtAddl	hidAddl
Checkbox	N/A	chk1, chk2, . . .	N/A
Hyperlink	<Employment List>	N/A	cand_empl.php?hidEditID=<Current Candidate ID>&hidQualID=<Current Employment ID>

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Button	Save	cmdSbmt	JavaScript:onSubmit()
Button	Clear	cmdClr	JavaScript:clrFrmFlds()
Button	Delete	cmdDelete	JavaScript:delRcrd()
Button	Proceed	cmdProceed	JavaScript:proceedTo()

Processing - The Page Managing Candidate's Employment Information (cand_empl.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A check is made if the variable **hidEditID** holds a value. If it is empty then the control passes to the **cand_main.php** page. The **cand_main.php** page thus when rendered with **header.php** file included fires an error indicating either to register or provides an option to login (Refer diagram 18.28.2). **hidEditID** being empty simply indicates that the candidate has not logged in and thus the candidate identity number is not available
  *The variable **hidEditID** is populated by either the **dosql.php** file or the other files such as cand_empl.php, cand_lang.php, cand_qual.php, cand_pers.php and cand_main.php files via the Links (Href). This variable holds the **CANDIDATE_ID** of the candidate **currently logged In**. This is how the **cand_empl.php** file understands what to display when control comes to it. This variable is used across all php pages in the candidate module.*

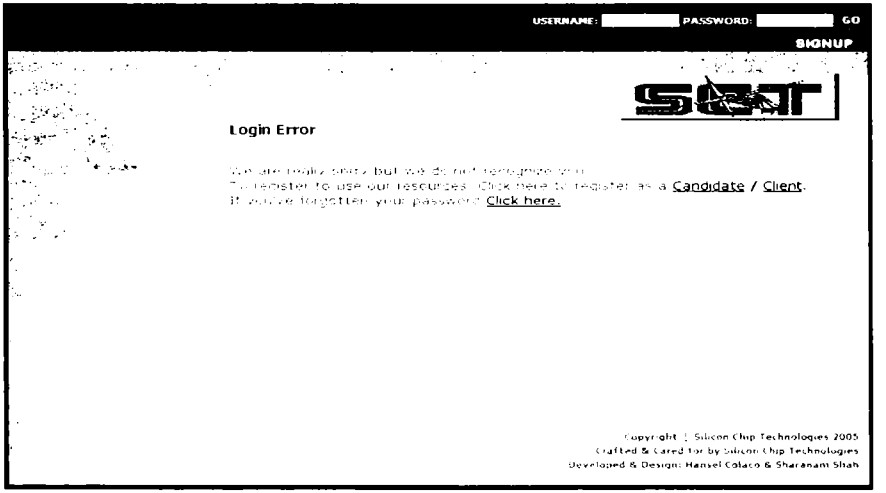


Diagram 18.28.2

- A variable named **auth_reqd** is defined holding value **YES** indicating that no user can access this page directly without having logged in as a candidate
- A variable named **helppage** is defined to store the name of the HTML page that will serve help on demand
- A variable named **disButton** is initialized to hold the status of the **Proceed** button code-spec in terms of **enabled/disabled**
  This is done to determine the appearance of the Proceed button. This button is disabled when there are no records available in the tabular layout
- A variable named **delEnabled** is initialized with a blank string. This variable will hold the status of **Delete** button code-spec in terms of **enabled/disabled**
  This is done to determine the appearance of the Delete button. This button is disabled when there are no records available in the tabular layout
- A variable named **hidMode** is initialized to **I**. This variable hold the form status in terms of **I** - **I**nsert, **U** - **U**pdate and **D** – **D**elete
- A variable **count** is initialized to hold **zero**. This is used in **while loops** to generate **unique checkbox names (e.g.** chk1, chk2 and so on**)** for the tabular layout
- A check is made to determine whether to include the **functions.php** file. This is done to avoid re-inclusion of the same file as follows**:**
  - If **func_incl** holds the value **YES** the inclusion of the **function.php** file is skipped. This means that the **functions.php** file is already included earlier. This inclusion is done by the **dosql.php** file.
- The following file is included**:**
  - /includes/header.php
    - This file is included to make the top bar available in the **cand_empl.php**. Refer diagram 18.29.1

Diagram 18.29.1

- A check is made to verify if the variable **hidEmpID** holds a value
  *This variable will hold a valid value, on selection of a record for editing it, from the tabular layout. This variable holds the **EMPL_ID** chosen for editing. This is how the **cand_empl.php** file understands which record is to be displayed when in edit mode. This check determines the view/update mode*
- Codespec to form the view mode follows**:**
- An SQL query is formed and executed. This query will **validate the existence** of the candidate identity and the qualification identity received via the variable **hidEditID** and **hidEmpID respectively** in the **CAND_EMPL** table
- If the SQL query returns any records then**:**
  - Data retrieved is transferred from the record-set to the hidden form variables
  - A variable named **delEnabled** is assigned a value, (i.e. disabled=true). This value indicates the delete button status code-spec in terms of **enabled/disabled**
  - The form mode is switched to Update by setting the value of variable **hidMode** to **U**
- A check is made if the variable **msg** holds a value. This check displays an error message when duplication while inserting the data captured by the **cand_empl.php** file into the database is encountered
  *The value held by the variable **msg** can either be a **one dimensional** (in this case) or **two dimensional**. In case of **one-dimensional** the value held would be an error message string, which has to be displayed.*
  Examples:
  1. **Please enter a valid username and password.**
  2. **Select an appropriate Currency.**
  3. **The username you entered has already been chosen by another user. Please enter a different username.**
  4. **Language already exists.**

  *In case of **two-dimensional** the first value (Dimension) will be an error number and the second value (Dimension) will be some value sent back as a duplicate entry in database.*
  Examples:
  1. **Details for <Exam Name> already exists. Please modify the information, if required.**

  *This variable is populated by the **dosql.php** file with an appropriate error message with/without the error number as the case may be. It is populated to transfer any errors if they occur during the processing carried out by the **dosql.php** file on behalf of the **cand_empl.php** file*

- If the variable **msg** holds a value then:
  - The value held is split into **two** portions in this case the **error number** i.e. **–1** and the **duplicate value**
  - Based on the number of dimensions, the **msg** variable holds error message is generated and populated in a variable namely **Message**
- This is followed by a set of java script functions:
  - **clrFrmFlds** – This function is called when a clear button is clicked and performs the following operations:
    - Clears the value held by the hidden variable defined
    - Clears the value held by the form fields
    - Switches the form mode back to insert. This is done by setting the value of hidden variable named **hidMode** to **I**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Disables the currency text box available on the **cand_empl.php** form
    - Enables the delete buttons available on the **cand_empl.php** form
  - **delRcrd** – This function is called when delete button is clicked and performs the following operations:
    - Calls the **clrFrmFlds** function to reset/clear the variables
    - Switches the form mode to delete. This is done by setting the value of hidden variable named **hidMode** to **D**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Calls a function named **formDeleteValues** available in the **/includes/rollover.js** file with a parameter to hold list of employment identities for deletion. This function generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same
  - **vldtFrmFlds** – This function is called when the form data is about to be submitted. This function verifies that mandatory form fields are not blank, dates entered are valid dates and so on. If an error is encountered then an error message indicates the same. These validations are performed only if the form is not in Delete mode. When the form is in Update mode, a check is carried out which will verify that there were changes made on the form. This is done by a comparison between the form field values and the hidden variables (holding the values prior update). This additional check is carried out just to avoid unnecessary submission to the **dosql.php** file which does the actual database update
  - **proceedTo** – This function is called when Proceed button is clicked. It is responsible to pass the control to the **cand_lang.php** file:

- Points the action attribute of the form tag to the **cand_empl.php** file
- Submits the form to implement the change

- **valCombo** – This function is called when an option from the currency Select box is selected. This function will check the selected value and accordingly determine whether the text field for Currency should be enabled or disabled. This means that if the currency select box has holds **---New---** as the selection of choice then the text box to hold the New Currency is **enabled otherwise** is **disabled**.

- Initial HTML code-spec corresponding to the template begins
- An SQL query is formed and executed. This query is responsible to extract the **IS_COMPLETE** and **RGST_COMPLETE** values from the **CANDIDATE** table
- Links are generated to all the candidate module pages if the registration procedure is complete. This is verified using the value held by the variable **IS_COMPLETE**
- This is followed by the FORM tag, which holds **ACTION** pointing to the **dosql.php** file and **onSubmit** pointing to the **vldtFrmFlds** function. The **dosql.php** file performs the form operations in terms of insert/update/delete. The control passes on click of the **Proceed** button
- Following hidden variables are declared using the INPUT tags:
  - **hidPage** – Holds the **page name** in this case **cand_empl.php**
  - **hidMode –** Holds the form mode in terms of **I**nsert, **U**pdate or **D**elete
  - **RGST_COMPLETE** – Is used to determine the registration status
  - **hidDelRcrdLst –** Holds a list of identities to be deleted
  - **hidEditId** – Holds **Candidate Identity** for update operations
  - **hidEmpId** – Holds **Candidate's Employment Identity** for update operations
    Following hidden variables are defined as a copy of the form fields. These will hold values prior updates and will be used by the **vldtFrmFlds** function to check (by comparing) if any modifications were performed on the form before passing the control to the **dosql.php** file for performing database operations
  - **hidCandID**
  - **hidEmp**
  - **hidLoc**
  - **hidPosHeld**
  - **hidPerFrom**
  - **hidPerFrom1**
  - **hidPerFrom2**
  - **hidPerTo**
  - **hidPerTo1**

- **hidPerTo2**
- **hidSalary**
- **hidSelCurrency**
- **hidTxtCurrency**
- **hidReason**
- **hidAddl**

- HTML Code-spec follows for designing the form
- An SQL query is formed and executed. This query retrieves the employment data from the **CAND_EMPL** table based on the **Candidate Identity**. The data retrieved will be used to form the tabular layout
- If the SQL query retrieves any records following operations are performed:
  - HTML Code-spec follows for creation of delete button which has **onClick** attribute set to a function call named **DelRcrd**
  - A tabular layout is created to hold the records retrieved. This is done using a **do while** loop, which traverses through the records retrieved from the table via the SQL query. While traversing through the loop the color code for each row displayed in the tabular layout is switched between two different colors. This is done to improve the readability
  - The **Name/Loc. Of Employer** as seen in the diagram 18.29.2 is assigned a link to the same page i.e. **cand_empl.php** by passing two parameters: **hidEditID** and **hidEmpID**. Based on the value held by these parameters the form fields are populated using an SQL query (explained earlier in Codespec to form the view mode follows point)

Delete

	Name/Loc. of Employer	Pos. Held	Period From	Period To	Last Sal. Drawn	Reasons for Leaving
☐	SCT/ Panvel	Programmer	31/03/2000	31/10/2001	6500 Rupees	Better Prospects
☐	Silicon Chip Technologies/ New York, USA	Sr. Programmner	01/12/2001	12/12/2004	6000 USD	Return To India
☐	Dad Services/ Mumbai	Asst. Programmer	01/01/1999	12/07/1999	3500 Rupees	Training Period
☐	HIC Systems/ Mumbai	Programmer	28/08/1999	31/03/2000	4500 Rupees	Low Salary

Diagram 18.29.2

- Finally the **footer.php** file is included to render the **copyright and the developer's information** on the **cand_empl.php** page. Refer Diagram 18.29.3

Diagram 18.29.3

6. **cand_lang.php** – This page provides an interface for managing candidate's language proficiency details stored in the **CAND_LANG** table. This form allows capturing (Insert), modifying (Update) and deleting data. It is available under **/candidates/cand_lang.php**

Layout - The Page Managing Candidate's Language Proficiency Information (cand_lang.php)

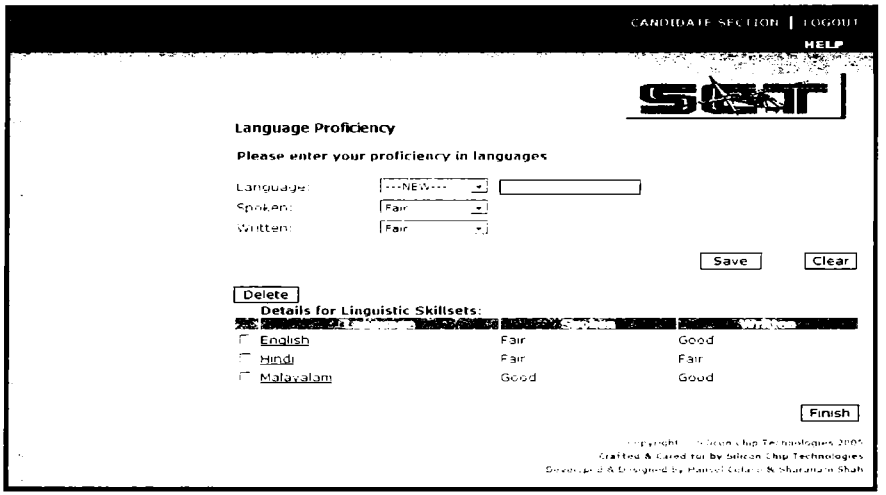


Diagram 18.30.1

Purpose

- Capture Candidate's language proficiency information
- Update / Delete information captured
- Allows completing/finishing the resume

GENERAL PAGE SPECIFICATIONS FOR cand_lang

FILES INCLUDED IN cand_lang.php:

PHP CODE BLOCK:	functions.php	Declares PHP based user-defined functions
	header.php	Generates the top bar for the HTML page
	footer.php	Generates the footer bar for the HTML page
HTML CODE BLOCK:	N/A	N/A

FUNCTIONS DECLARED IN cand_lang.php:

JavaScript:	PHP:
valCombo(), clrFrmFlds(), delRcrd(), vldtFrmFlds(), proceedTo()	N/A

OBJECTS ON THE HTML PAGE:

OBJECT	LABEL	NAME	DESCRIPTION
Hyperlink	Modify Personal Info.	N/A	chng_pers.php?hidEditID=<current_UserID>
Hyperlink	Modify Qualification Details	N/A	chng_qual.php?hidEditID=<current_UserID>
Hyperlink	Modify Experience Details	N/A	chng_empl.php?hidEditID=<current_UserID>

FORM DETAILS:

FORM NAME: frmCandLang	BOUND TO: CAND_LANG	ACTION: dosql.php

DATA FIELDS:

OBJECT	LABEL	NAME	BOUND TO
Hidden	N/A	hidPage	N/A
Hidden	N/A	rgst_complete	CANDIDATE.RGST_COMPLETE
Hidden	N/A	hidEditID	USERS.USER_ID
Hidden	N/A	hidLangID	CAND_LANG.LANG_ID
Hidden	N/A	hidCandID	CAND_LANG.CAND_ID
Hidden	N/A	hidLang	N/A
Hidden	N/A	hidSpoken	CAND_LANG.SPOKEN
Hidden	N/A	hidWritten	CAND_LANG.WRITTEN
Hidden	N/A	hidMode	N/A
Hidden	N/A	hidDelRcrdLst	N/A
Select Box	Language	selLang	hidLangID
Text Box		txtLang	hidLang
Select Box	Spoken	selSpoken	hidSpoken
Select Box	Written	selWritten	hidWritten
Checkbox	N/A	chk1, chk2, . . .	N/A
Hyperlink	<Employment List>	N/A	cand_lang.php?hidEditID=<Current Candidate ID>&hidQualID=<Current Language ID>

DATA CONTROLS:

OBJECT	LABEL	NAME	ACTION
Button	Save	cmdSbmt	JavaScript:onSubmit()
Button	Clear	cmdClr	JavaScript:clrFrmFlds()
Button	Delete	cmdDelete	JavaScript:delRcrd()
Button	Finish	cmdProceed	JavaScript:proceedTo()

Processing - The Page Managing Candidate's Language Proficiency Information (cand_lang.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A check if the variable **hidEditID** holds a value. If it is empty then the control passes to the **cand_main.php** page. The **cand_main.php** page thus when rendered with **header.php** file included fires an error indicating either to register or provides an option to login (Refer diagram 18.30.2). **hidEditID** being empty simply indicates that the candidate has not logged in and thus the candidate identity number is not available
  *The variable **hidEditID** is populated by either the **dosql.php** file or the other files such as cand_empl.php, cand_lang.php, cand_qual.php, cand_pers.php and cand_main.php files via the Links (Href). This variable holds the **CANDIDATE_ID** of the candidate **currently logged in**. This Is how the **cand_lang.php** file understands what to display when control comes to it. Thls variable is used across all php pages in the candidate module.*

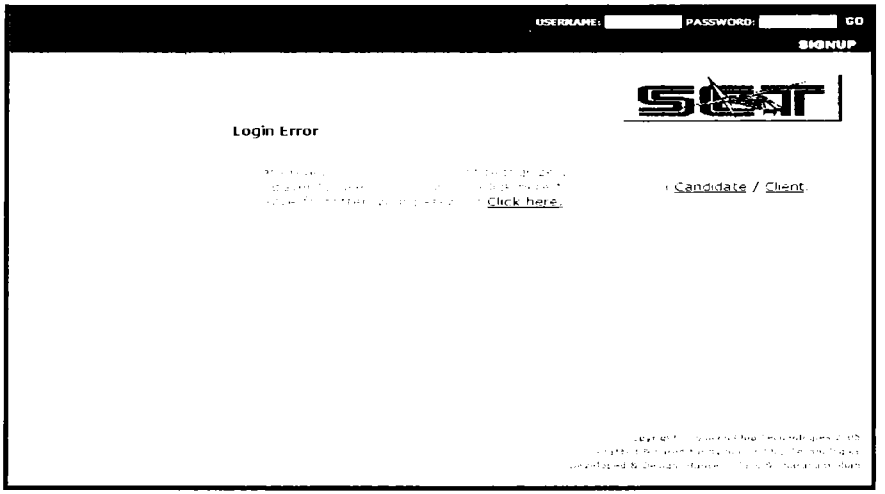


Diagram 18.30.2

- A variable named **auth_reqd** is defined holding value **YES** indicating that no user can access this page directly without having logged in as a candidate
- A variable named **helppage** is defined to store the name of the HTML page that will serve help on demand
- A variable named **disButton** is initialized to hold the status of the **Proceed** button code-spec in terms of **enabled/disabled**
  This is done to determine the appearance of the Proceed button. This button is disabled when there are no records available in the tabular layout
- A variable named **delEnabled** is initialized with a blank string. This variable will hold the status of **Delete** button code-spec in terms of **enabled/disabled**

This is done to determine the appearance of the Delete button. This button is disabled when there are no records available in the tabular layout

- A variable named **hidMode** is initialized to **I**. This variable hold the form status in terms of **I** - **I**nsert, **U** - **U**pdate and **D** – **D**elete
- A variable **count** is initialized to hold **zero**. This is used in **while loops** to generate **unique checkbox names (e.g.** chk1, chk2 and so on**)** for the tabular layout
- A check is made to determine whether to include the **functions.php** file. This is done to avoid re-inclusion of the same file as follows:
  - If **func_incl** holds the value **YES** the inclusion of the **function.php** file is skipped. This means that the **functions.php** file is already included earlier. This inclusion is done by the **dosql.php** file.
- The following file is included:
  - /includes/header.php
    - This file is included to make the top bar available in the **cand_lang.php**. Refer diagram 18.31.1

Diagram 18.31.1

- A check is made to verify if the variable **hidLangID** holds a value
  *This variable will hold a valid value, on selection of a record for editing it, from the tabular layout. This variable holds the **LANG_ID** chosen for editing. This is how the **cand_lang.php** file understands which record is to be displayed when in edit mode. This check determines the view/update mode*
- Codespec to form the view mode follows:
- An SQL query is formed and executed. This query will **validate the existence** of the candidate identity and the language proficiency identity received via the variable **hidEditID** and **hidLangID respectively** in the **CAND_LANG** table
- If the SQL query returns any records then:
  - Data retrieved is transferred from the record-set to the hidden form variables
  - A variable named **delEnabled** is assigned a value (i.e. disabled=true). This value indicates the delete button status code-spec in terms of **enabled/disabled**
  - The form mode is switched to Update by setting the value of variable **hidMode** to **U**
- A check is made if the variable **msg** holds a value. This check displays an error message when duplication while inserting the data captured by the **cand_lang.php** file into the database is encountered
  *The value held by the variable **msg** can either be a **one dimensional** (in this case) or **two dimensional**. In case of **one-dimensional** the value held would be an error message string, which has to be displayed.*

Examples:

1. **Please enter a valid username and password.**
2. **Select an appropriate Currency.**
3. **The username you entered has already been chosen by another user. Please enter a different username.**
4. **Language already exists.**

*In case of **two-dimensional** the first value (Dimension) will be an error number and the second value (Dimension) will be some value sent back as a duplicate entry in database.*

Examples:

1. **Details for <Exam Name> already exists. Please modify the information, if required.**

*This variable is populated by the **dosql.php** file with an appropriate error message with/without the error number as the case may be. It is populated to transfer any errors if they occur during the processing carried out by the **dosql.php** file on behalf of the **cand_lang.php** file*

- If the variable **msg** holds a value then:
  - The value held is split into **two** portions in this case the **error number** i.e. **–1** and the **duplicate value**
  - Based on the number of dimensions, the **msg** variable holds error message is generated and populated in a variable namely **Message**
- The default value for the spoken and written type select box is set to **Fair**
- This is followed by a set of java script functions:
  - **clrFrmFlds** – This function is called when a clear button is clicked and performs the following operations:
    - Clears the value held by the hidden variable defined
    - Clears the value held by the form fields
    - Switches the form mode back to insert. This is done by setting the value of hidden variable named **hidMode** to **I**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation
    - Enables the delete buttons available on the **cand_lang.php** form
  - **delRcrd** – This function is called when delete button is clicked and performs the following operations:
    - Calls the **clrFrmFlds** function to reset/clear the variables
    - Switches the form mode to delete. This is done by setting the value of hidden variable named **hidMode** to **D**. This variable will be used by the **dosql.php** file to understand the mode in which the form is and accordingly perform appropriate database operation

- Calls a function named **formDeleteValues** available in the **/includes/rollover.js** file with a parameter to hold list of employment identities for deletion. This function generates a string of comma-separated identities of the records selected for deletion. If no records are selected, a message indicates the same

- **vldtFrmFlds** – This function is called when the form data is about to be submitted. This function verifies that mandatory form fields are not blank, dates entered are valid dates and so on. If an error is encountered then an error message indicates the same. These validations are performed only if the form is not in Delete mode. When the form is in Update mode, a check is carried out which will verify that there were changes made on the form. This is done by a comparison between the form field values and the hidden variables (holding the values prior update). This additional check is carried out just to avoid unnecessary submission to the **dosql.php** file which does the actual database update
- **proceedTo** – This function is called when Proceed button is clicked. It is responsible to pass the control to the **dosql.php** file with the parameter **cand_final:**
  - Setting the **hidPage** variable to hold **cand_final**. This indicates that the candidate has completed the registration procedure
  - Points the action attribute of the form tag to the **dosql.php** file
  - Submits the form to implement the change
- **valCombo** – This function is called when an option from the language Select box is selected. This function will check the selected value and accordingly determine whether the text field for language should be enabled or disabled. This means that if the language select box holds **---New---** as the selection of choice then the text box to hold the New Language is **enabled otherwise** is **disabled**.

- Initial HTML code-spec corresponding to the template begins
- An SQL query is formed and executed. This query is responsible to extract the **IS_COMPLETE** and **RGST_COMPLETE** values from the **CANDIDATE** table
- Links are generated to all the candidate module pages if the registration procedure is complete. This is verified using the value held by the variable **IS_COMPLETE**
- This is followed by the FORM tag, which holds **ACTION** pointing to the **dosql.php** file and **onSubmit** pointing to the **vldtFrmFlds** function. The **dosql.php** file performs the form operations in terms of insert/update/delete. The control passes on click of the **Proceed** button
- Following hidden variables are declared using the INPUT tags**:**
  - **hidPage** – Holds the **page name** in this case **cand_empl.php**
  - **hidMode –** Holds the form mode in terms of **I**nsert, **U**pdate or **D**elete
  - **RGST_COMPLETE** – Is used to determine the registration status
  - **hidDelRcrdLst –** Holds a list of identities to be deleted

- **hidEditId** – Holds **Candidate Identity** for update operations
- **hidLangId** – Holds **Candidate's Language Proficiency Identity** for update operations
  Following hidden variables are defined as a copy of the form fields. These will hold values prior updates and will be used by the **vldtFrmFlds** function to check (by comparing) if the any modifications were performed on the form before passing the control to the **dosql.php** file for performing database operations
- **hidCandID**
- **hidLang**
- **hidSpoken**
- **hidWritten**

- HTML Code-spec follows for designing the form
- An SQL query is formed and executed. This query retrieves the language proficiency data from the **CAND_LANG and LANGUAGE** table based on the **Candidate Identity**. (Since the **language name** is also required the **LANGUAGE** table is used with a JOIN). The data retrieved will be used to form the tabular layout
- If the SQL query retrieves any records following operations are performed**:**
  - HTML Code-spec follows for the creation of a delete button whose **onClick** attribute set to call a function named **DelRcrd**
  - A tabular layout is created to hold the records retrieved. This is done using a **do while** loop, which traverses through the records retrieved from the table via the SQL query. While traversing through the loop the color code for each row displayed in the tabular layout is switched between two different colors. This is done to improve the readability
  - The **Language** as seen in the diagram 18.31.2 is assigned a link to the same page i.e. **cand_lang.php** by passing two parameters**: hidEditID** and **hidLangID**. Based on the value held by these parameters the form fields are populated using an SQL query (explained earlier in Codespec to form the view mode follows point)

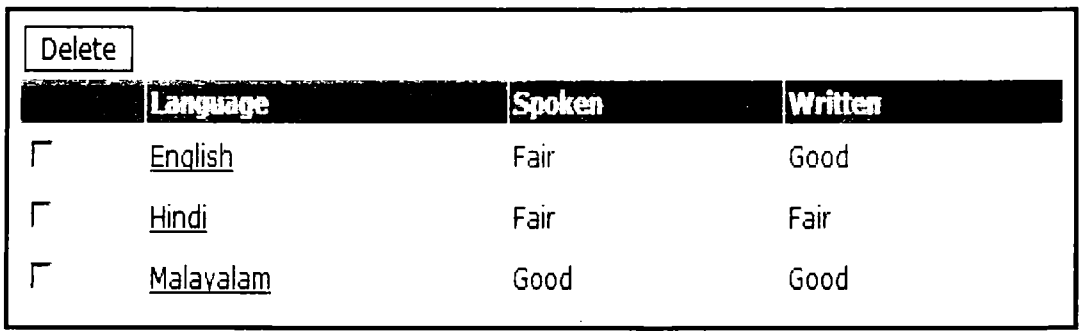


Diagram 18.31.2

- Finally the **footer.php** file is included to render the **copyright and the developer's information** on the **cand_lang.php** page. Refer Diagram 18.31.3

Copyright © Silicon Chip Technologies 2005
Crafted & Cared for by Silicon Chip Technologies
Developed & Designed by Hansel Colaco & Sharanam Shah

Diagram 18.31.3

7. **chng_login.php** – Allows modifications to the password, hint question and answer. Control is shifted to this page when the user selects **Modify Login Information** from the **cand_main.php** page. This page is available under **/candidates/chng_login.php**

Layout - *Change Login Page Of The CANDIDATE Module (chng_login.php)*

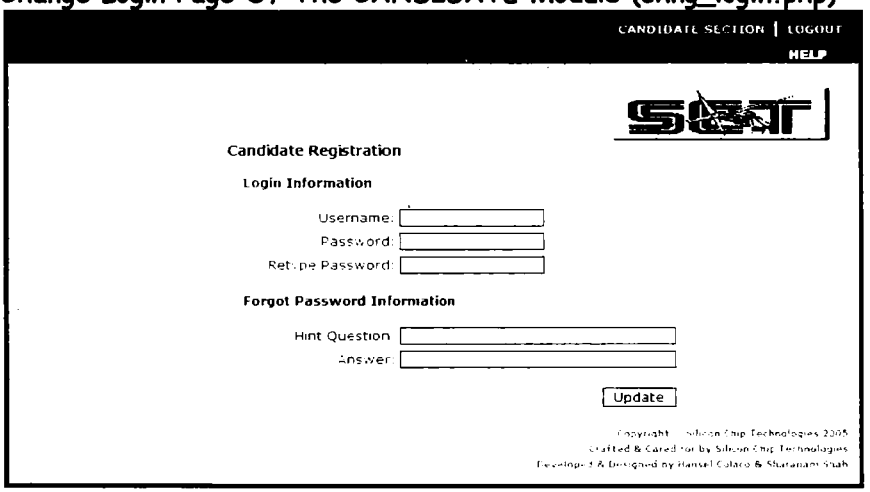


Diagram 18.32

Purpose

- Allows modifying the login information such as password and hint question/answer

Processing - Change Login Page Of The CANDIDATE Module (chng_login.php)

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A variable named **helppage** is defined to store the name of the HTML page that will serve help on demand
- A variable named **auth_reqd** is defined holding value **YES** indicating that no user can access this page directly without having logged in as a candidate

- The following two files are included:
  - /includes/functions.php
    - This file is included to make available the functions defined
  - /includes/header.php
    - This file is included to make the top bar available in the **cand_reg.php**. Refer diagram 18.33.1

Diagram 18.33.1

- An SQL query is formed and executed to retrieve the **user_ID** and the **user_username** from the USERS table based on the **candidate's identity** and the **candidate's user identity** of the candidate table. This data is used to populate the user name textbox and also to populate a hidden variable **hidUserID**
- Java script code-spec follows to validate password (including the retyped password) captured by this page. This is done using a function named **frmValidate**. This function is called when the information captured by this page is submitted using the **Update** button. This means a client side validation is performed before validating the password (including the retyped password) against the database at the server end
- Initial HTML code-spec corresponding to the template begins
- HTML code-spec for designing the layout of the form in terms of textboxes and update button responsible to capture the password, retyped password, Hint question and answer follows
- This includes a FORM tag definition wherein the **ACTION** attribute of the FORM tag is pointed to the **dosql.php** file and the **OnSubmit** attribute is pointed to the **frmValidate** java script function
  *The **dosql.php** file (explained later) does the actual processing bit for this page in terms of insertion of the data captured by this page*
- Finally the **footer.php** file is included to render the **copyright and the developer's information** on the **chng_login.php** page. Refer Diagram 18.33.2

Diagram 18.33.2

8. **dosql.php** – This file performs the Insert, Update and Delete operations for all the data entry forms of the candidate module and is available under **/candidates/dosql.php**

Purpose

- Transfers data captured by the d/e form to the database
- Allows deleting data from the database based on the instructions given by the d/e form
- Allows modifications to the data based on the changes made via the d/e form

Processing - DOSQL.PHP

- A variable named **author** is defined to store the name of the programmer who designed/coded the page
- A variable named **func_incl** is defined to keep track of the **functions.php** file inclusion
- The following three files are included:
  - /includes/config.php
    - This file is included to make the variables declared available in the dosql.php
  - /includes/db_connect.php
    - This file is included to establish a connection with the database and thus make available the variable **rcq** which is the database handle
- Following checks are made:
  - If the variable **hidPage** holds the value **cand_reg**. This means control had been passed by the **cand_reg.php** file for **I**nsert (in this case), **U**pdate or **D**elete operations. If it holds **cand_reg** then:
    - An SQL query is formed and executed to retrieve information from the USERS table based on value held by **txtUsername** form field. This is done to verify if the username chosen/captured by the **cand_reg.php** page is being duplicated. If it is being duplicated then an error message indicates the same. The control is passed backed to the **cand_reg.php** file which actually displays the error message
    - The username and password captured are validated. If the validation fails a message is displayed when the candidate Registration page is re-displayed on the screen
    - If both the above validations pass through, the data is inserted using the following processes:
      - The **functions.php** file is included to make available the functions defined
      - A unique **user identity** is generated and stored in a variable using the **gen_serial** function defined in the **functions.php** file. The value generated will be stored into the database as a **primary key** value for the **user_ID** field of the USERS table
      - An SQL query is formed and executed to perform the actual insert operation for the data captured bound to the USERS table by the **cand_reg.php** file

- A unique **user identity** is generated and stored in a variable using the **gen_serial** function defined in the **functions.php** file. The value generated will be stored into the database as a **primary key** value for the **cand_ID** field of the CANDIDATE table
- An SQL query is formed and executed to perform the actual insert operation for the data captured bound to the CANDIDATE table by the **cand_reg.php** file. The CANDIDATE table now holds one row with the following three values:
  - **CAND_USER_ID** – Holds the newly generated **user_ID**
  - **CAND_ID** – Holds the newly generated **cand_ID**
  - **IS_COMPLETE** – Holds the value **0**. This indicates that the user has reached the **cand_pers.php** page by submitting the candidate registration information

- Session variables are registered and populated. These variables are used to store the user identity, Login date and time and the type of user (in this case candidate)
- Control is passed to the **cand_pers.php** file along with the newly generated candidate identity via the **hidEditID** variable. This file will allow the user to proceed with the capturing of candidate's personal information

- If the variable **hidPage** holds the value **cand_pers**. This means control had been passed by the **cand_pers.php** file for **I**nsert, **U**pdate or **D**elete operations. If it holds **cand_pers** then:
  - The **functions.php** file is included to make available the functions defined
  - The following date fields on the **cand_pers.php** page are concatenated and stored in appropriate variables
    - txtDOB, txtDOB1, txtDOB2
    - txtDOI, txtDOI1, txtDOI2
    - txtDOE, txtDOE1, txtDOE2
    - txtLDOI, txtLDOI1, txtLDOI2
    - txtLDOE, txtLDOE1, txtLDOE2
  - A variable named **alter_query** is defined and initialized. This variable will be used as a part of the SQL query formed later

- If the user has specified the path to the photograph it is uploaded to the server using the function **move_uploaded_file**. This function accepts two parameters i.e. the source and the destination. The **txtPhoto** acts as the source holding the path to the photograph and the variable **photoURL** (defined in **config.php** file) concatenated with the contents of the **hidEditID** variable holding the candidate identity. This simply means the photograph will be renamed to the candidate's identity of the candidate holding it. Further the **alter_query** variable is appended to hold the assignment of the **cand_pic** field.
  - This functionality requires a folder named **candpics** under **pms** i.e. the root of virtual directory. The **candpics** folder should be assigned **read write permission** as: chmod 777 candpics. The **owner** of the candpics folder should be changed to **nobody** as: chown nobody.nobody candpics
- If the value held by the **IS_COMPLETE** variable is 0 then it is changed to **1** to indicate that the user has passed through the **cand_pers.php** file. The **alter_query** variable is appended to indicate the same
- An SQL query is formed and executed to store the data captured bound to the CANDIDATE table via the **cand_pers.php** file. This query is concatenated with the **alter_query** variable for an extension to the assignments made earlier
- A check is made on the contents of the hidden variable (**hidPreAddrID**) holding the Present Address Identity. If it holds an identity number then an update operation is performed. Otherwise an insert operation is performed. This variable is populated by the **cand_pers.php** file just when the form loads i.e. before any changes are attempted to the form contents
- A check is made on the contents of the hidden variable (**hidPerAddrID**) holding the Permanent Address Identity. If it holds an identity number then an update operation is performed. Otherwise an insert operation is performed. This variable is populated by the **cand_pers.php** file just when the form loads i.e. before any changes are attempted to the form contents
- A check is made on the contents of the hidden variable (**hidPass**) holding an indication to availability of passport details. If it holds YES (means the details were captured and inserted earlier) then an update operation is performed. Otherwise an insert operation is performed. This variable is populated by the **cand_pers.php** file just when the form loads i.e. before any changes are attempted to the form contents
- A check is made on the contents of the hidden variable (**hidLic**) holding an indication to availability of driving license details. If it holds YES (means the details were captured and inserted earlier) then an update operation is performed. Otherwise an insert operation is performed. This variable is populated by the **cand_pers.php** file just when the form loads i.e. before any changes are attempted to the form contents
- Control is passed to the **cand_qual.php** file along with the candidate identity via the **hidEditID** variable. This file will allow the user to proceed with the capturing of candidate's qualification information

- If the variable **hidPage** holds the value **cand_qual**. This means control had been passed by the **cand_qual.php** file for **I**nsert, **U**pdate or **D**elete operations. If it holds **cand_qual** then:
  - **functions.php** file is included to make available the functions defined. A variable named **func_incl** is set to hold the value **YES**. This is to intimate the **cand_qual.php** file that the **functions.php** file has already been included and thus avoid its inclusion again
  - A check is made on the value held by the variable **hidMode**. If it holds the value **I** it means that an **Insert** operation is desired and processes further as:
    - Based on the data captured by the **cand_qual.php** an SQL query is formed and executed to check for data duplication in terms of **exam name**. If the exam name is found to be duplicating then the control is passed back to the data entry form i.e. **cand_qual.php** along with an appropriate error message number and the exam name being duplicated for display. This error is populated by the **cand_qual.php** file based on the error number and the exam name
    - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual insertion of data in the database. During this insert operation care is taken to generate the next primary key value using the **gen_serial** function defined in the **functions.php** file
  - A check is made on the value held by the variable **hidMode**. If it holds the value **U** it means that an **Update** operation is desired and processes further as:
    - A check is made if the **Exam Name** has been modified. Comparing the form field with a hidden variable, which holds the value prior modifications, does this check. If there are no modifications in the Exam Name then an SQL query is formed and executed to update the remaining fields of the CAND_QUAL table
    - If the Exam Name is modified then an SQL query is formed and executed to check for duplication after the Exam Name is modified.
    - If the Exam Name modification creates duplication then control is passed back to the **cand_qual.php** file along with an error number and the Exam Name being duplicated. The **cand_qual.php** file based on the error number generates and renders the error message
    - If the Exam Name modification does not create any duplicates then an SQL query does the update operation
  - A check is made on the value held by the variable **hidMode**. If it holds the value **D** it means that a **Delete** operation is desired and processes further as:

- In this mode a variable **hidDelRcrdLst** holds comma separated string of the identities (in this case Qual_ID) to be deleted. This comma separated string is created by a java-script function **formDeleteValues** available in the **rollover.js** file explained earlier. This comma separated string is transformed into an array using the split function
- A counter (**ctr**) is initialized based on the size of the array. This is required to traverse a specific number of times (based on the value held by the **ctr** variable) using a loop
- A while loop is used to perform the following operations:
  - An SQL query is formed and executed to perform the actual delete operation
  - Counter (**ctr**) is decremented by **1**

- An SQL query is formed and executed to retrieve the qualification identities of the candidate currently logged in.
  - If no records are retrieved it means that all the records are deleted. If all the records are deleted the candidate has to complete the resume by starting from the **cand_qual.php** page the next time the candidate logs in. To provision this the IS_COMPLETE field of the CANDIDATE table is updated to hold **1** which simply indicates that the resume is complete only till the personal information page
  - If records are retrieved by the above SQL query then:
    - A check is carried out which determines the registration complete status. If the registration is complete then the IS_COMPLETE field of the CANDIDATE table is updated to hold NULL. This means that the candidate has a complete resume
    - Otherwise the IS_COMPLETE field of the CANDIDATE table is updated to hold **2**. This means that the candidate has completed the qualification details page and is proceeding towards candidate's employment details page or has the candidate's employment details incomplete when the candidate logs in the next time
- Control is passed back to the **cand_qual.php** page

o If the variable **hidPage** holds the value **cand_empl**. This means control had been passed by the **cand_empl.php** file for **I**nsert, **U**pdate or **D**elete operations. If it holds **cand_empl** then:

- **functions.php** file is included to make available the functions defined. A variable named **func_incl** is set to hold the value **YES**. This is to intimate the **cand_qual.php** file that the **functions.php** file has already been included and thus avoid its inclusion again
- A check is made to verify that the form is not in delete mode. If the form is not in delete mode then following processes are undertaken:

- The currency details are validated
- The dates captured by the form being in a separated format are concatenated and stored in variables as a complete date

- A check is made on the value held by the variable **hidMode**. If it holds the value **I** it means that an **Insert** operation is desired and processes further as:
  - Based on the data captured by the **cand_empl.php** an SQL query is formed and executed to check for data duplication in terms of **employer's name** and **employment location**. If these are found to be duplicating then the control is passed back to the data entry form i.e. **cand_empl.php** along with an appropriate error message number and the employer's name and location being duplicated for display. This error is populated by the **cand_empl.php** file based on the error number and the employer's name and location
  - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual insertion of data in the database. During this insert operation care is taken to generate the next primary key value using the **gen_serial** function defined in the **functions.php** file
- A check is made on the value held by the variable **hidMode**. If it holds the value **U** it means that an **Update** operation is desired and processes further as:
  - A check is made if the **Employer's Name** and **Location** has been modified. Comparing the form fields with hidden variables, which hold the values prior modifications, does this check. If there are no modifications in the Employer's Name and Location then an SQL query is formed and executed to update the remaining fields of the CAND_EMPL table
  - If the Employer's Name and Location is modified then an SQL query is formed and executed to check for duplication after modification
  - If the modification creates duplication then control is passed back to the **cand_empl.php** file along with an error number and the Employer's Name and Location being duplicated. The **cand_empl.php** file based on the error number generates and renders the error message
  - If the modification does not create any duplicates then an SQL query does the update operation
- A check is made on the value held by the variable **hidMode**. If it holds the value **D** it means that a **Delete** operation is desired and processes further as:
  - In this mode a variable **hidDelRcrdLst** holds comma separated string of the identities (in this case Empl_ID) to be deleted. This comma separated string is created by a java-script function **formDeleteValues** available in the **rollover.js** file explained earlier. This comma separated string is transformed into an array using the split function

- A counter (**ctr**) is initialized based on the size of the array. This is required to traverse a specific number of times (based on the value held by the **ctr** variable) using a loop
- A while loop is used to perform the following operations:
  - An SQL query is formed and executed to perform the actual delete operation
  - Counter (**ctr**) is decremented by **1**

- Code-spec to validate the dates captured follow. This code-spec will first retrieve the already existing records in the CAND_EMPL table and compare the dates captured by the form with ones retrieved by the SQL query. This is done using a while loop. If dates conflict then the same is indicated by an error message
- Code-spec to calculate the total number of experience based on the records retrieved by the SQL query follow. This code-spec will then update the CANDIDATE table's cand_tot_exp field with the calculated value
- An SQL query is formed and executed to retrieve the employment identities of the candidate currently logged in
  - If no records are retrieved it means that all the records are deleted. If all the records are deleted the candidate has to complete the resume by starting from the **cand_empl.php** page the next time the candidate logs in. To provision this the IS_COMPLETE field of the CANDIDATE table is updated to hold **2** which simply indicates that the resume is complete only till the qualification details page
  - If records are retrieved by the above SQL query then:
    - A check is carried out which determines the registration complete status. If the registration is complete then the IS_COMPLETE field of the CANDIDATE table is updated to hold NULL. This means that the candidate has a complete resume
    - Otherwise the IS_COMPLETE field of the CANDIDATE table is updated to hold **3**. This means that the candidate has completed the employment details page and is proceeding towards candidate's language proficiency details page or has the candidate's language proficiency details incomplete when the candidate logs in the next time
- Control is passed back to the **cand_empl.php** page

If the variable **hidPage** holds the value **cand_lang**. This means control had been passed by the **cand_lang.php** file for **I**nsert, **U**pdate or **D**elete operations. If it holds **cand_lang** then:

- **functions.php** file is included to make available the functions defined. A variable named **func_incl** is set to hold the value **YES**. This is to intimate the **cand_qual.php** file that the **functions.php** file has already been included and thus avoid its inclusion again

- A check is made on the value held by the variable **hidMode**. If it holds the value **I** it means that an **Insert** operation is desired and processes further as:
  - If a new language was entered in the language text box. This means that existing language was not selected using the select box then:
    - An SQL query is formed and executed to verify that the language name captured already exists in the database
    - If it doesn't exist then a unique language identity is generated and the language name captured is inserted into the LANGUAGE table
  - Based on the data captured by the **cand_lang.php** an SQL query is formed and executed to check for data duplication in terms of **language identity**. If found to be duplicating then the control is passed back to the data entry form i.e. **cand_lang.php** along with an appropriate error message number. This error is rendered by the **cand_lang.php** file
  - If the data captured is free of duplicates then an SQL query is formed and executed to perform the actual insertion of data in the database
- A check is made on the value held by the variable **hidMode**. If it holds the value **U** it means that an **Update** operation is desired and processes further as:
  - An SQL query is formed and executed to update the fields in the CAND_LANG table
- A check is made on the value held by the variable **hidMode**. If it holds the value **D** it means that a **Delete** operation is desired and processes further as:
  - In this mode a variable **hidDelRcrdLst** holds comma separated string of the identities (in this case LANG_ID) to be deleted. This comma separated string is created by a java-script function **formDeleteValues** available in the **rollover.js** file explained earlier
  - Based on the value held in the hidDelRcrdLst variable an SQL query is formed and executed to delete the records accordingly
- An SQL query is formed and executed to retrieve the language proficiency identities of the candidate currently logged in
  - If no records are retrieved it means that all the records are deleted. If all the records are deleted the candidate has to complete the resume by starting from the **cand_lang.php** page the next time the candidate logs in. To provision this the IS_COMPLETE field of the CANDIDATE table is updated to hold **3** which simply indicates that the resume is complete till the candidate's employment details page
  - If records are retrieved by the above SQL query then:
    - A check is carried out which determines the registration complete status. If the registration is complete then the IS_COMPLETE field of the CANDIDATE table is updated to hold NULL. This means that the candidate has a complete resume

- Otherwise the IS_COMPLETE field of the CANDIDATE table is updated to hold **4**. This means that the candidate has completed the language proficiency details page and is proceeding towards completion

- Control is passed back to the **cand_lang.php** page

- If the variable **hidPage** holds the value **cand_final**. This means control had been passed using the **Finish** button on the **cand_lang.php**. If it holds **cand_final** then:
  - **functions.php** file is included to make available the functions defined
  - The IS_COMPLETE is set to NULL which indicates that all the pages are complete and the RGST_COMPLETE is set to YES which indicates that the candidate has completed the entire registration process
  - Control is passed to the **cand_main.php** page
- If the variable **hidPage** holds the value **chng_login_cand**. This means control had been passed from the **chng_login.php** file for update operation. If it holds **chng_login_cand** then:
  - An SQL query is formed and executed to update the data captured in terms of password, hint questions and the answer
  - Control is passed to the **cand_main.php** page

The CLIENT Module

Since this module is under development there is no processing undertaken when the user accesses this module page as seen in diagram 18.34 appears.

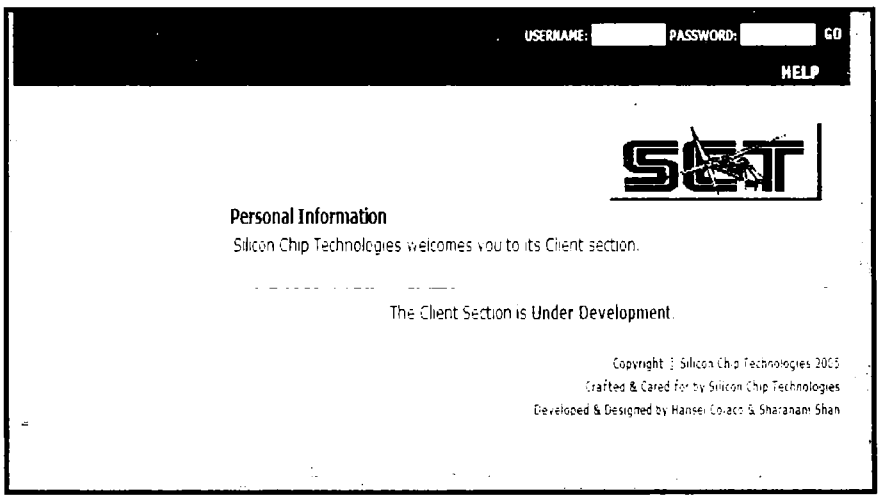


Diagram 18.34

SECTION V: APPENDIX

Setting UP The Red Hat AS 3.0 Operating System

Configuring A Dual Boot System

Most people load Linux on a computer along with another operating system, usually some flavor of M.S. Windows such as Windows 95, 98, ME, XP and so on. Generally this is because most users are still experimenting with Linux and haven't yet made up their minds whether they should adopt Linux as their favorite operating system or not.

Another factor that forces this type of arrangement is that Linux still does not have a large number of software products that make an individual as productive as M.S. Windows does. Hence most first time adopters of Linux try and keep a M.S. Windows presence along with Linux. Their computers have **two** operating systems loaded on their hard disk with some flavor of M.S. Windows and Linux.

If any computer has two operating systems loaded, it requires some sort of **dual bootstrap** program loaded as well. This dual bootstrap program allows an operating system to be chosen **just prior** actual O/s boot up into the computer's memory.

REMINDER

Two operating systems **cannot run** in the memory of the same computer at the same time. Each operating system must boot from and use its own hard disk drive **or** hard disk partition.

This material explains how to configure a computer to boot to Red Hat AS 3.0 Linux **or** another operating system. For clarity, the other operating system is assumed to be some flavor of Microsoft Windows™. Having said this, the general procedure is similar for any other operating system as well.

REMINDER

During O/s installation, when using hard disk partitions set the proper partition for Linux using **fdisk** (or any other hard disk partitioning utility).

If **no other** operating system is installed on the computer, **install M.S. Windows first** and then install Red Hat AS 3.0 Linux.

HINT

When installing Windows (versions from 9*x* to ME), **partitions cannot** be defined during Windows installation. Partitions must be **defined prior** the installation of these flavors of Windows.

When installing Windows NT or Windows 2000, partitions of a specific size can be created for Windows during installation. Leave enough free space (space that is not partitioned or formatted) on the hard drive to install Red Hat AS 3.0 Linux. A minimum of 10GB is good enough for a full install of Linux with some space left over for experimenting.

Allocating Disk Space for Linux

WARNING

Backup all important information before reconfiguring a hard drive. Reconfiguring a hard drive can result **in complete loss of data**.

Additionally, be sure to create a boot disk for both operating systems in case the boot loader fails to recognize either of them.

To install Red Hat AS 3.0 the choices are as follows:

- Add a new hard drive
- Use an existing hard drive or partition
- Restructuring an existing partition to create space for loading Linux

REMINDER

For all three options, be aware that the BIOS in some older systems cannot access more than the first 1024 cylinders on a hard drive. If this is the case, the `/boot` Linux partition must be located on **the first 1024 cylinders** of the hard drive to boot Linux.

Add A New Hard Drive

The simplest way to make room for Red Hat AS 3.0 is to add a new hard disk drive to the computer and then install Red Hat AS 3.0 on that drive.

If a second IDE hard drive is added to the computer, the Red Hat AS 3.0 installation program will recognize it as **hdb** and the existing drive (the one used by M.S. Windows) as **hda**.

Start the Red Hat AS 3.0 installation program and make sure Linux is being installed on the newly installed hard drive (**i.e.** `hdb`) rather than the hard drive used by Windows.

Use An Existing Hard Drive Or Partition

Another way to make room for Linux is to use a hard drive that is currently being used by M.S. Windows, which has an additional disk partition.

If Windows Explorer shows two hard drives, `C:` and `D:` this could indicate either that the computer has two hard drives, **or** a **single** hard drive with **two partitions.** In either case (assuming the hard drive has enough disk space), Red Hat AS 3.0 can be installed on the hard drive **or** disk partition that Windows currently recognizes as `D:`

REMINDER

Windows uses letters to refer to removable drives (for example, a ZIP drive or a CD-ROM writer) and network storage (virtual drives) **as well as** for local hard drive space. Linux **cannot** be installed on a removable **or** network drive.

If a local Windows partition is available in which Linux must be installed, complete the following**:**

1. Copy all data from the selected hard drive **or** partition (**D:** in this example) to another location. This data will be **completely lost** during Linux installation.
2. Start the Red Hat AS 3.0 Linux installation program and instruct it to install Linux in the designated drive or partition (**i.e.** the partition that M.S. Windows has designated as `D:`)

REMINDER

Linux distinguishes between hard drives and disk partitions as indicated below.

If `C:` and `D:` on the computer refer to two separate hard drives, the installation program will recognize them as **`hda`** and **`hdb`** (IDE) **or** **`sda`** and **`sdb`** (SCSI).

Tell the installation program to install on **`hdb`** (or **`sdb`**).

If `C:` and `D:` refer to **partitions** on a single drive, the installation program will recognize them as **`hda1`** and **`hda2`** (**or** **`sda1`** and **`sda2`** i.e. SCSI partitions).

During the partitioning phase of the Linux installation, if **Disk Druid** is used, **delete** the second partition (**`hda2`** or **`sda2`**), **and then** reallocate that free space to Linux.

To use **Auto Partition** Option the second partition should be deleted (unallocated) prior the Linux installation.

Restructuring An Existing Partition To Create Space For Loading Linux

The third way to make room for Linux is to create a new partition for Red Hat AS 3.0 Linux on the hard drive being used by M.S. Windows. If Windows Explorer shows only one hard drive (C:), then C: must be partitioned such that one partition will continue to hold M.S. Windows while the other will hold Red Hat AS 3.0 Linux.

After partitioning drive C:, Windows Explorer will see a smaller C: drive. When the Red Hat AS 3.0 Linux installation program is run the **Auto Partition** option can be selected to use the remainder (unallocated space) of the drive for Linux.

A destructive partitioning program, such as **fdisk**, can be used to partition the hard drive. Doing so will require the re-installation of Windows.

A number of non-destructive, third party partitioning programs, are available for M.S. Windows (**Partition Magic** being one such, *search for Windows based, hard disk partitionining programs on the Internet using Google*).

Using fdisk To Partition A Hard Disk

Create a Boot Up floppy disk from any Windows 9X computer. Using this floppy disk **boot** the computer on which two operating systems will be loaded.

REMINDER

The Boot Up floppy disk created by Windows 9X provides CD-ROM support. Thus after booting from this floppy, CD-ROM support is available if required.

After the computer boots up from the floppy, the DOS prompt (A:\>) appears. Key in **fdisk** and press the **ENTER key**.

```
A:\> fdisk ↵
```

The FDISK utility is invoked from the floppy. It scans the Hard Disk connected to the computer. If the HDD has a capacity greater than 2048 Mega bytes, a message indicating this is displayed.

In addition the following is prompted:

```
Do you wish to enable large disk support (Y/N) ....? [Y]
```

Press the key **Y** to continue. **fdisk** loads into memory. Its Main menu options are displayed.

Viewing The Current Partitions

The Main menu option for fdisk is as shown below:

1. Create DOS partition or Logical DOS Drive
2. Set active partition
3. Delete partition or Logical DOS Drive
4. Display partition information

To view the partitions on the HDD (if any), type **4** and press **ENTER**. fdisk scans the HDD and lists all existing partition(s).

Deleting Partitions

If existing partitions have to be deleted, type **3** and press **ENTER**. The menu options for deleting partitions will be displayed as listed below:

1. Delete Primary DOS partition
2. Delete Extended DOS partition
3. Delete Logical DOS Drive(s)
4. Delete NON-DOS partition

REMINDER

The approach is to delete the **logical DOS Drive(s) first**. Then delete the Extended DOS partition. Finally delete the Primary DOS partition if required.

The Extended DOS partition **cannot be deleted** before deleting the Logical DOS Drive(s) defined in it.

If **all** the current partition(s) are deleted, it is time to create new partitions. Press the Escape (**Esc**) key several times to return to the fdisk Main menu.

Creating New Partitions

To create partition(s), type **1** and press **ENTER**. The following sub-options are displayed:

1. Create Primary DOS partition
2. Create Extended DOS partition
3. Create Logical DOS Drive(s) in the Extended DOS partition

Type **1** and press **ENTER** to create the Primary DOS partition. When this is done fdisk scans the HDD for errors. After the HDD is verified for errors, an option to allocate the total HDD capacity to the primary partition is displayed. Type **N** and press **ENTER**. This tells fdisk that the entire HDD is **not being** treated as a primary DOS partition.

Next, enter the capacity of the Primary DOS partition, either as a percentage or in megabytes, and press **ENTER**. Having allocated the space for the primary partition, fdisk will return to the Main Menu.

HINT

If required an Extended DOS partition can be created. The Extended partition should occupy the balance space on the HDD. Several logical drives can share the Extended DOS partition or a single logical drive can occupy the entire Extended DOS partition, as required.

Having partitioned the HDD as required, **exit from fdisk** by pressing the Escape (**Esc**) key until the system prompt re-appears.

WARNING

Ensure that the Primary DOS partition is **marked as active** before exiting to the DOS prompt. A warning is displayed by FDISK if an attempt to exit to the system prompt is made **without marking** the Primary DOS partition as **active**.

When the DOS prompt appears restart the computer and once more boot up using the same Boot Up floppy Disk.

Before using the Primary DOS partition it needs to be formatted. At the system prompt key in:

```
A:\> format C: /s/v ↵
```

This formats the primary DOS partition, transfers the operating system to it, (which is currently resident in memory) and finally verifies that the transfer has been done correctly. Remove the floppy from the drive and reboot.

This time let the O/S load from the HDD. This validates that the operating system was transferred to the HDD successfully.

Installing M.S. Windows From the HDD

The following provides minimal guidelines for installing Windows XP on a computer.

- Set the System BIOS to boot from CD-ROM
- Insert Windows XP CD and reboot the machine
- A Message **`Press any key to boot from CD`** is prompted. Press any key.
- A blue screen comes up and the drivers required by the installer are loaded.
- Next, a screen with the following options are shown:
  - To set up Windows XP now, press ENTER
  - To repair a Windows XP installation using Recovery Console, press R
  - To quit setup without installing Windows XP, press F3

  Press Enter to continue.
- On pressing enter The **E**nd **U**ser **L**icense **A**greement is displayed. Press F8 to agree.
- After agreeing to the EULA a screen which displays the available drives/partitions appears with the following options:
  - To set up Windows XP on the selected item, press ENTER.
  - To create a partition in the unpartitioned space, press C.
  - To delete the selected partition press D

  Select the desired partition and press ENTER.
- Select the type of file system required:
  1. Format the partition using the NTFS file system (Quick)
  2. Format the partition using the FAT file system (Quick)
  3. Format the partition using the NTFS file system
  4. Format the partition using the FAT file system
  5. Convert the partition to NTFS
  6. Leave the current file system intact (No Changes)

  Select 4th Option i.e. Format the partition using the FAT file system.

  Press Enter to continue.

REMINDER

If the selected partition is greater than 2 GB a message is displayed stating that The Partition will be formatted using FAT32.

A. SETTING UP THE RED HAT AS 3.0 OPERATING SYSTEM

- A screen showing the following options appears:
  - To format the drive press F
  - To select a different partition for Windows XP press Esc
- The formatting starts followed by copying of files to their installation folders.
- The computer reboots and the installation configuration starts.
- During installation the following will be prompted for:
  - Regional Setting just click **Next**.
  - Ownership Details

    Type in the desired Owner and the Organization Name.
  - Serial Number

    Type in the Serial Number received along with the Windows XP CDROM.
  - Computer Name

    Type the desired Computer name and administrator password, if required.
  - Date Time Settings

    Select the current date and time.
  - Network Settings (Optional)

    Enter appropriate network TCP/IP settings (obtained from the network administrator)
  - Workgroup/Domain Settings

    Enter appropriate Workgroup/Domain names based on the current network setup.
- Copying Continues.
- The System Reboots
- A wizard, which asks about user creation and Internet connectivity, runs on first boot. *Use the wizard and pass appropriate values for users and their passwords. Internet connectivity can be put off for later, if required.*

Windows XP is now loaded and ready for use.

REMINDER

The **only reason** that Windows XP was installed was to have a M.S. Windows presence while actually using the computer as a Linux server. *This is to emulate how most first time Linux users tend to setup their computers.*

It is strongly recommended that there is **no M.S. Windows presence on the computer at all**. Linux being the only operating system on the computer.

Choosing The Appropriate OS AT bootup

Since there are going to be two OS loaded on the HDD, **i.e.** Windows **XP** and **Linux**, some software must run when the computer is switched on (**or** restarted), this software must permit a user to choose the operating system that should be loaded in memory. If the user cannot (*or fails to*) make a choice, then within a pre-set amount of time the **OS set as default**, will automatically load into memory.

HINT

A Linux based software called a **boot loader** takes care of this. There are two boot loaders that can be used with Linux, they are **GRUB** or **LILO**.

Either software will display each OS loaded on the HDD and allow the user to choose a specific OS to load into memory. Red Hat Linux recommends GRUB. Generally XP is set as the **default OS** (see the *Note* above).

The installation of Red Hat AS 3.0 Linux on the free space left on the HDD can now commence.

Installing Red Hat AS 3.0

Red Hat AS 3.0, (from now on referred to as **Linux**), comes on several CD's. Set the computer to boot from its CD-ROM drive via an appropriate change in its **BIOS Setup**. Place the **first** Red Hat AS 3.0 installation CD into the CD-ROM drive and restart the computer. Since the computer has been instructed to boot from the CD-ROM it will pick up its OS from the first CD inserted in the CD-ROM drive instead of the HDD.

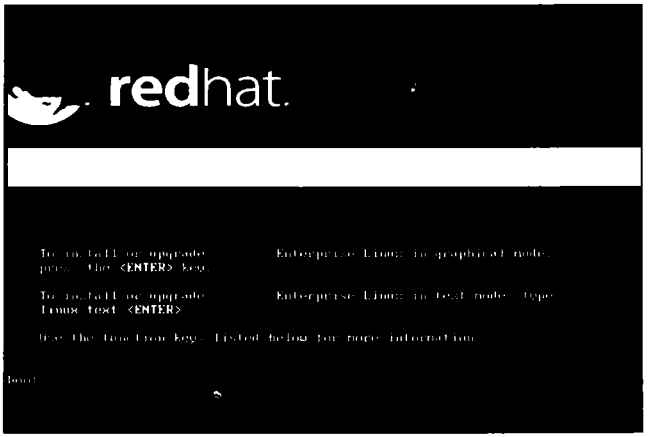


Diagram A.1.1: The CD-ROM boot up screen.

Having booted the OS from the first CD, the computer's VDU will display a screen as shown in diagram A.1.1.

The system will wait for the user to either press the Enter key if desired to run the installation in graphical mode, or to type in a command at the boot: prompt if required. Some machines are particular when it comes to installing Linux, and require the installation to be run with one or more parameters. This is where the user would enter those parameters. However for now the installation process can be commenced by simply pressing the **Enter key** at the boot: prompt. This will be followed by a screen prompting to check, the CD inserted for errors if any, in the drive. This step can be skipped if the user is sure about the CDs being used for installation, by selecting the **skip** option. Refer diagram A.1.2

REMINDER

If no command is executed within an interval of **30** seconds, Linux installation automatically commences in **graphical mode**.

Diagram A.1.2: The CDROM Testing screen.

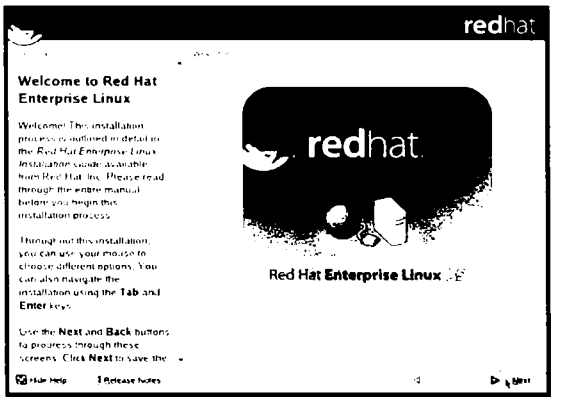


Diagram A.2: The Welcome to Red Hat AS 3.0 screen.

The Linux welcome screen is displayed to start off the installation wizard. It consists of a two-pane dialog. Help text is in the left pane and tasks to be performed are displayed in the right pane, as in diagram A.2.

The panel at the bottom of the screen contains the buttons described within table A.1:

Hide Help	Clicking it will close the installation notes appearing in the left hand of the screen
Release Notes	Clicking it will display the Release notes that have accompanied the setup files being used
Back	Clicking it will retrace one step backward in the installation process
Next	Clicking it will proceed one step forward in the installation process

Table A.1

Click Next to get the **Language Selection** screen as shown in diagram A.3.

Select an appropriate language for the Operating System and click Next to continue.

In the **Keyboard Configuration** screen (as shown in diagram A.4.1), select the appropriate keyboard type and click Next to continue.

The next screen accepts the hardware settings for the Mouse attached to the computer. Refer diagram A.4.2.

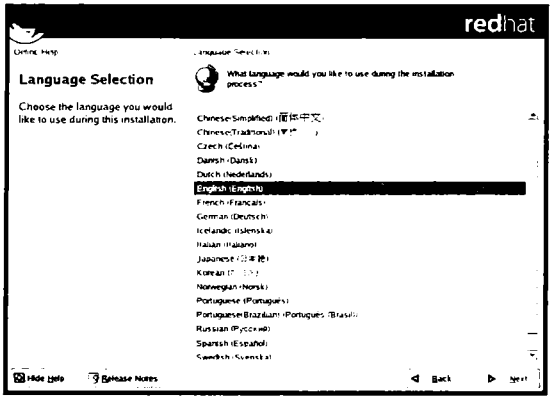


Diagram A.3: The Language Selection screen.

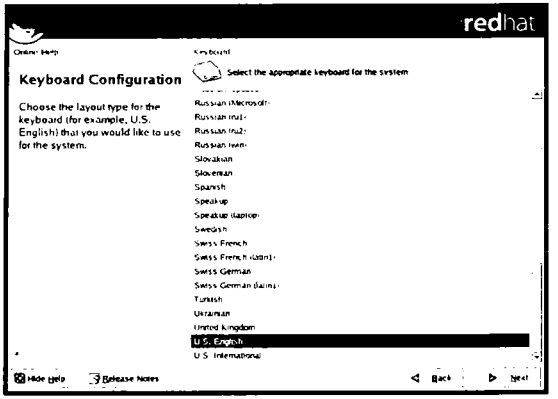


Diagram A.4.1: The Keyboard Configuration screen.

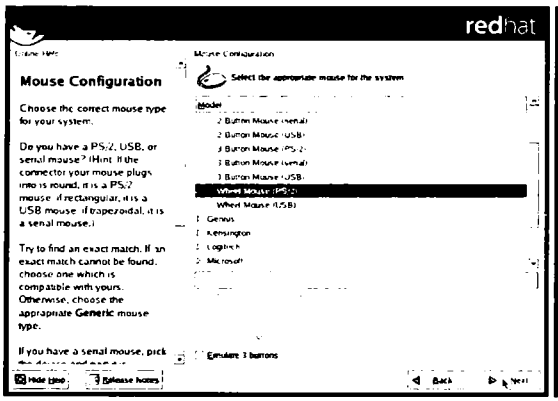


Diagram A.4.2: The Mouse Configuration screen.

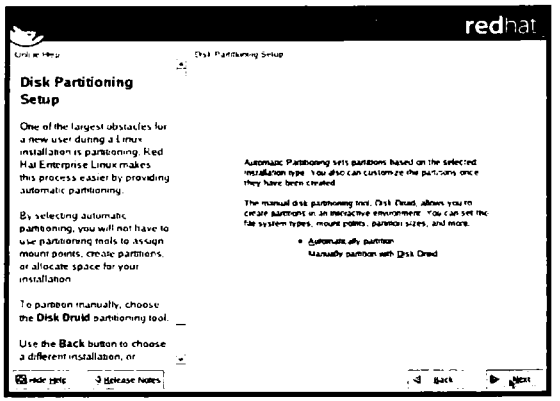
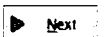


Diagram A.5.1: The Disk Partitioning screen.

After selecting the hardware features related to the mouse click  to continue.

The options selected in following steps are **critical**. The first of these critical steps is selecting a partitioning strategy. The **Disk Partitioning Setup** screen will be displayed as shown in diagram A.5.1.

The options provided here are as follows:

1. Automatically partition
2. Manually partition with **Disk Druid**

Select the **first option** (**i.e.** Automatically partition), to allow the installer itself will decide how to install and will require very few inputs from the user.

REMINDER

If the other option is selected (i.e. Manually partition with disk druid), the user has to decide the manner in which the HDD will be partitioned.

Click Next to allow the installer to scan the HDD for compatibility. The installer will proceed to the Automatic Partitioning Screen as shown in diagram A.5.2.

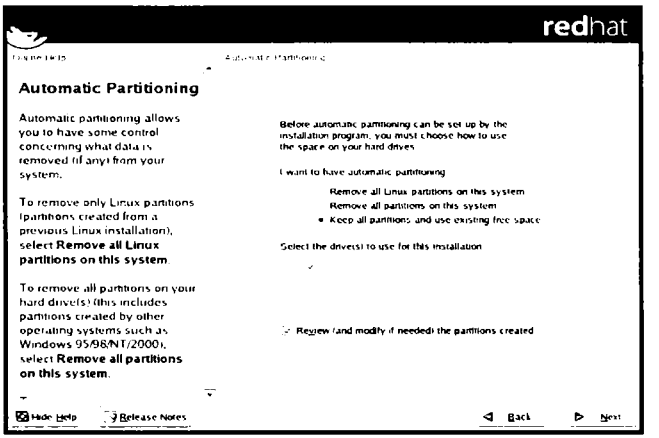


Diagram A.5.2: The Automatic Partitioning screen.

When creating partitions automatically the following options are available:

- Remove all Linux Partitions on this system
- Remove all partitions on this system
- Keep all partitions and use existing free space

1. If Linux had been installed previously on the same HDD, select the first option to reuse the space previously allocated to Linux
2. If the entire HDD has to be allocated to Linux, select the second option

REMINDER

If option 2 is chosen and any other O/S was previously installed, such as M.S. Windows it will be **completely destroyed**.

3. If Linux has to be installed on a **dual boot** system, **select the third option**. *In this case the free space is the unallocated space in the extended partition on the HDD*

The field below the three options is used to list the various HDD media connected to the computer. Select option **3** and click Next.

Since **Keep all the partitions and use the existing free space** option was selected the installer scans the HDD and generates appropriate partition information. This information is displayed in the Disk Setup screen. Refer diagram A.6.

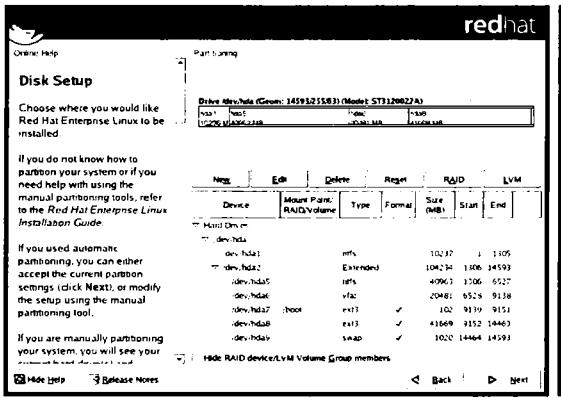


Diagram A.6: HDD's Partition Information.

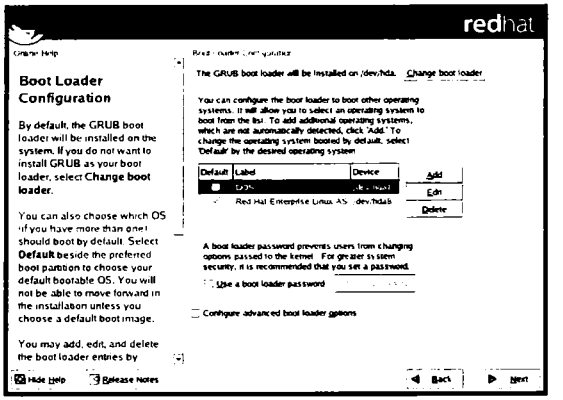


Diagram A.7.1: The Boot Loader Configuration screen.

In general on a dual boot system, the installer creates the following partitions in the free space available on the HDD:

Device	Size (Mb)	Type	Mount Point	Format
/dev/hda1	(variable)	vfat		
/dev/hda2	43 to 100	ext3	/boot	Yes
/dev/hda3	(balance)	ext3	/	Yes
/dev/had4	RAM * 2	Extended		
/dev/hda5	RAM * 2	swap		Yes

Click Next to continue. The **Boot Loader Configuration** screen as shown in diagram A.7.1 is displayed.

Choose the Linux based, **boot loader** software. This software is required on a **dual boot** machine. When the computer first starts up (**or** is restarted) it is the boot loader that allows a user to choose an operating system to be loaded into memory.

REMINDER

Clicking the Change boot loader displays the **Change Boot Loader** dialog box. This dialog box provides the following options while installing **Red Hat AS 3.0 Linux:**

- Use GRUB as the boot loader
- Use LILO as the boot loader
- Do not install a boot loader

By default, GRUB is select as the boot loader software.

The next section of this screen provides a list of O/s choices that will be available via the boot loader. Since the computer is a dual boot system with Windows and Linux, the two will be listed here. Refer diagram A.7.1.

To edit the label attached to the O/s choices, select the label and then click Edit. The Image dialog box will appear as shown in diagram A.7.2.

Type in the appropriate **Label** and click OK.

Set the default operating system as required by clicking the check box besides the options for O/s choices.

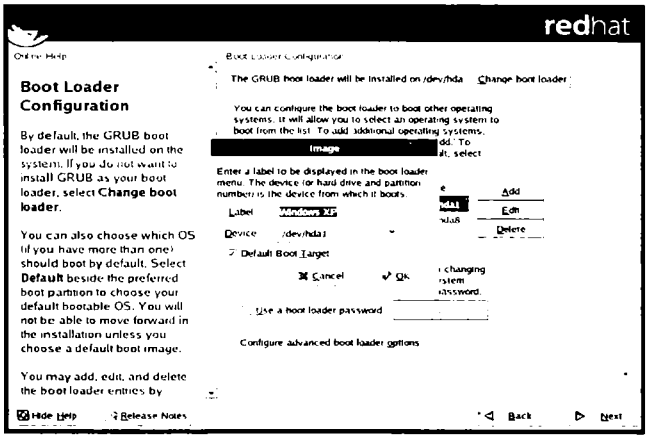


Diagram A.7.2: The Image dialog box.

REMINDER

The third section is for providing a password for the selected boot loader. Clicking on the **Use a boot loader password** option will prompt the **Enter Boot Loader Password** dialog box. This dialog box accepts a password for the boot loader. It recommends that the length of the password be **six** characters or more. If acceptable, this step can be ignored.

The fourth and the last section in the **Boot Loader Configuration** screen is the **Configure advance boot loader options**. *Experts should attempt this*.

Click Next the **Network Configuration** screen is displayed as shown in diagram A.8.

Set the properties of an Ethernet card by selecting the checkbox in the **Active on Boot** option and then click Edit.

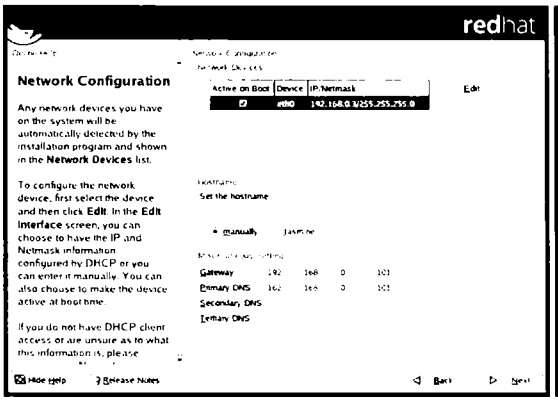


Diagram A.8: The Network Configuration screen.

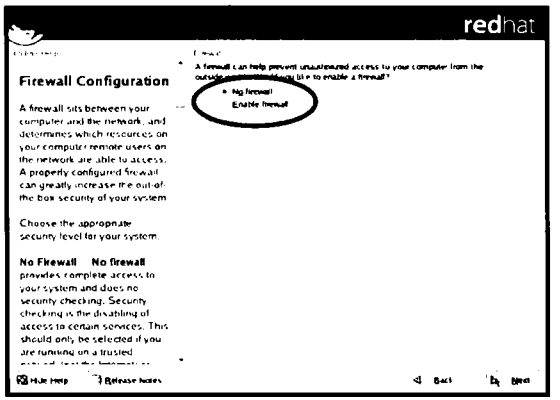


Diagram A.9: The Firewall Configuration screen.

Modify the settings to suit the networking in use. Click Next to setup Linux's Firewall security, as shown in diagram A.9.

A firewall is a vehicle for controlling what types of traffic are allowed to access the machine from the network. Red Hat includes a software firewall that is turned on by default, via the "Enable firewall" option button. The user can optionally permit selected services to pass through the firewall by checking the appropriate check boxes in the middle of this screen.

Currently, select the **No firewall** option.

Click Next to proceed to the **Language Selection** screen as shown in diagram A.10.

A list of languages supported in Linux is displayed in the right hand side of the screen. Select an appropriate language for the Operating System and click Next to continue.

Set the **default language** in the drop down list box above the list of languages.

Click Next to continue to the **Time Zone Selection** screen. Select the appropriate time zone as shown in diagram A.11. (**i.e.** Asia/Calcutta, for Linux computers in India).

Click Next to get the **Set Root Password** screen as shown in diagram A.12.

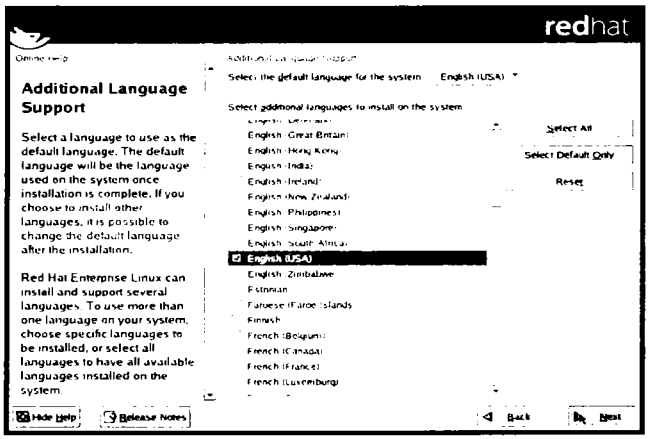


Diagram A.10: The Language Selection screen.

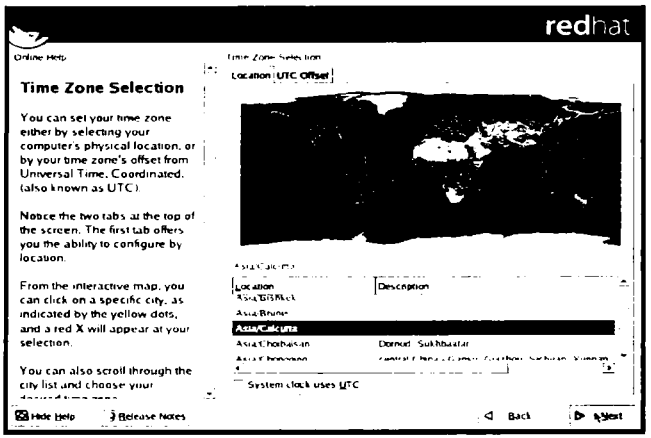


Diagram A.11: The Time Zone Selection screen.

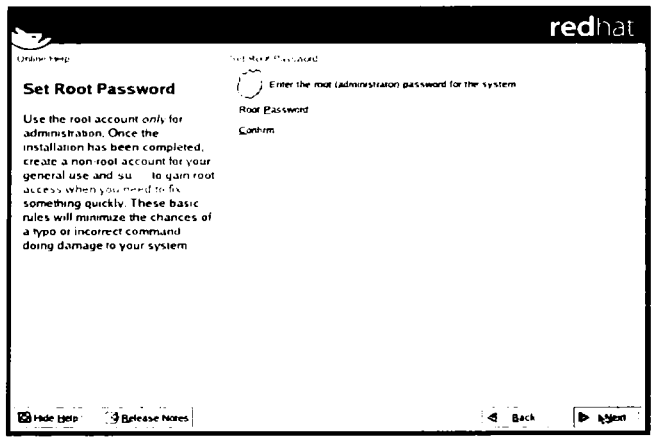


Diagram A.12: The Set Root Password screen.

REMINDER

The root password provides access to the root user's account. Root is analogous to Administrator on a Windows system. It's important to set a very strong password for the Root user.

This screen accepts the **password** for the **root** login. After keying in the password for **root**, click Next.

The **Package Installation Defaults** screen is displayed as shown in diagram A.13. At this stage of installation, a choose can be made about the packages groups which will be installed on the system. There are two options provided:

- Accept the current package list (i.e default list)
- Customize the set of packages to be installed

Select the **Customize the set of packages to be installed** option and click Next

The **Package Group Selection** screen is displayed, as shown in diagram A.14.

Check boxes allow the selection of various components and utilities available for installation.

Should there be sufficient HDD space, simply choose the last option, **Everything** and Click Next. The installer will install all packages available.

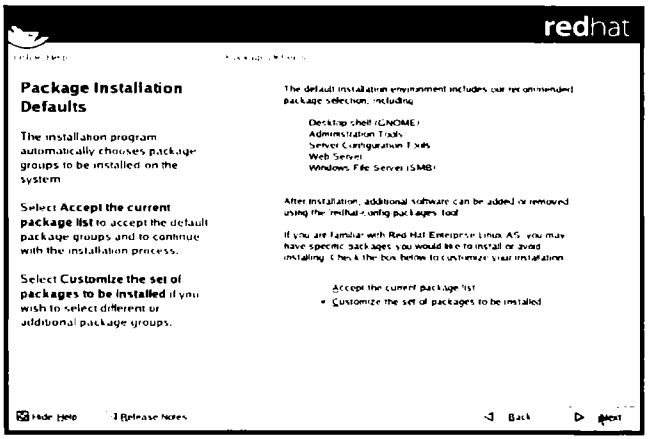


Diagram A.13: The Package Installation Default

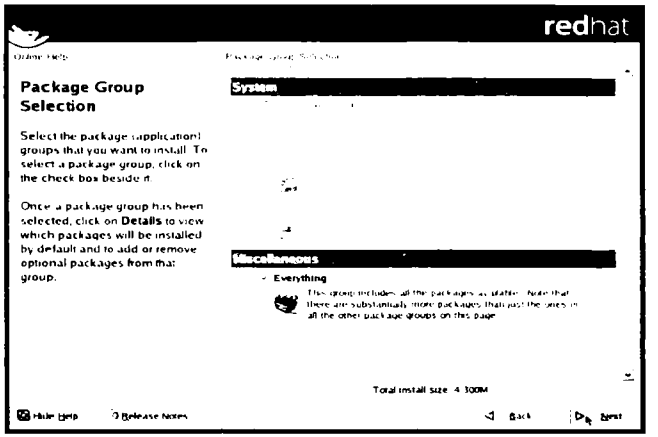


Diagram A.14: The Package Group Selection screen.

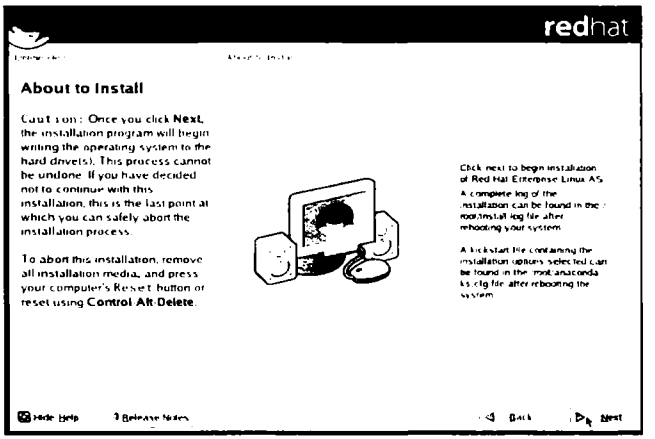


Diagram A.15: The About to Install screen.

Click Next to get the **About to Install** screen as shown in diagram A.15. This screen indicates that the installer has collected all the data required for installing Red Hat AS 3.0 Linux.

REMINDER

Once installation begins, notes regarding the progress are stored in a text file named **install.log** and stored in the **/root** directory. If anything goes wrong during installation, the user can examine this file for clues about what might have happened.

Click Next to start installation.

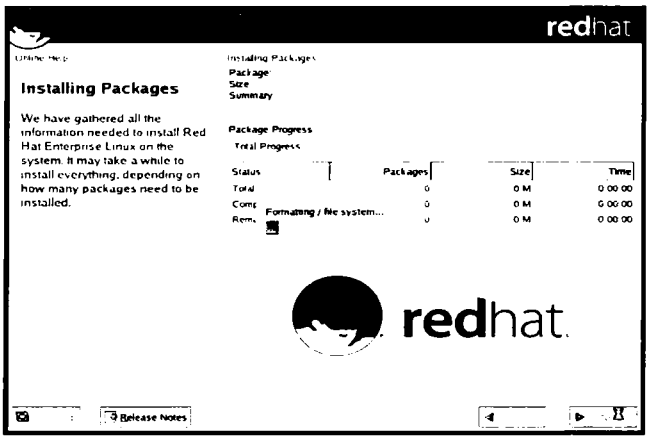


Diagram A.16.1: Formatting the HDD.

The Linux installer will first partition and format the HDD (as shown in diagram A.16.1) based on the information captured in the **Partitioning Strategy** step. Refer diagrams A.5.1 to A.6

The installer will then transfer the install image to the hard disc drive, as shown in diagram A.16.2.

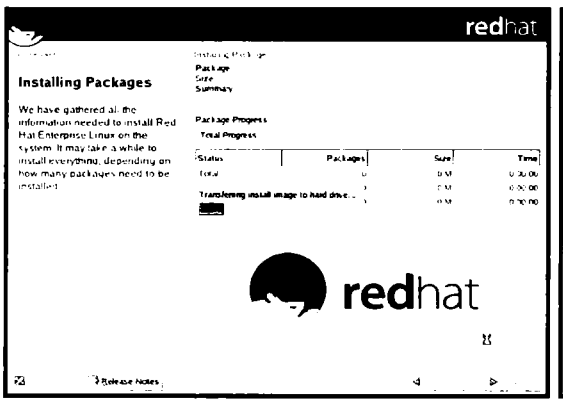


Diagram A.16.2: Install image transferred to HDD.

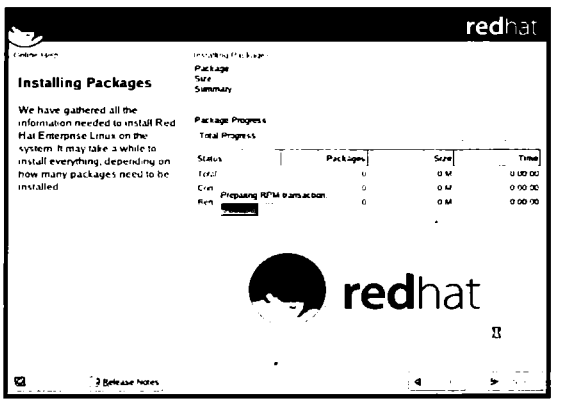


Diagram A.16.3: Preparing the RPM transaction.

Additionally, the installer will prepare the RPM transaction (i.e. the order in which Linux based RMPs will be loaded). Refer diagram A.16.3.

Next the Linux installer transfers all the packages from the Linux installation CD's to the HDD. This is a time consuming process and requires the changing of CD's when prompted. Refer diagram A.17.1 and A.17.4.

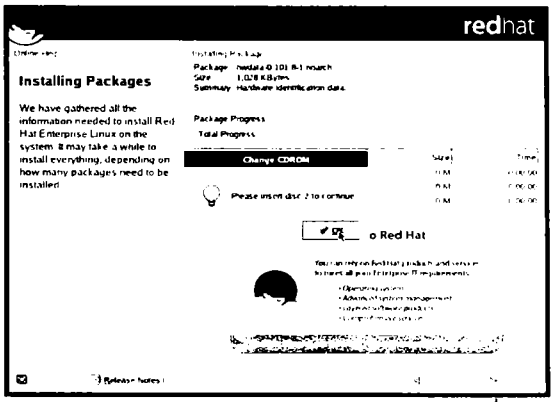


Diagram A.17.1: Prompt to insert the 2nd CD.

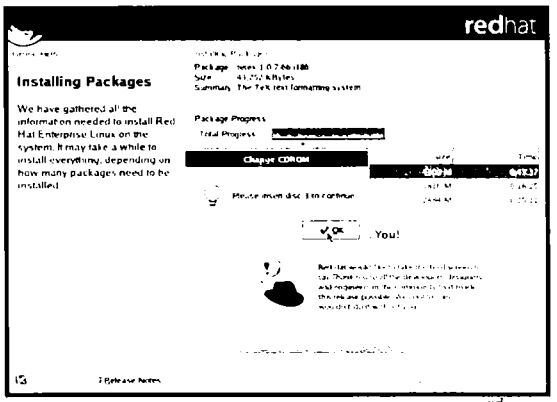


Diagram A.17.2: Prompt to insert the 3rd CD.

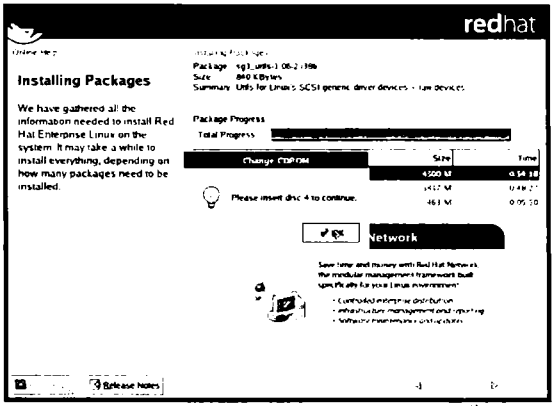


Diagram A.17.3: Prompt to insert the 4th CD.

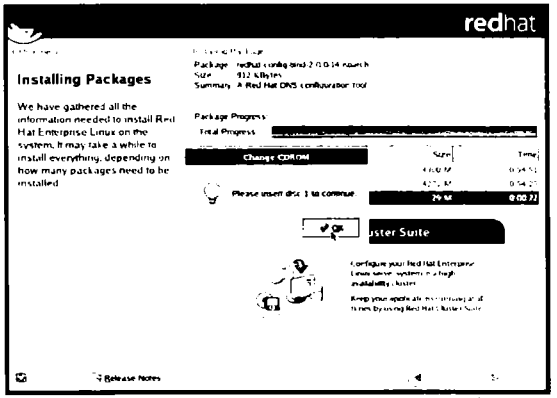
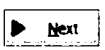


Diagram A.17.4: Prompt to insert the 1st CD.

Having copied all the required components, installation automatically proceeds with the post installation configuration before going to the next screen as shown in diagram A.18.

In the **Graphical Interface (X) Configuration** screen, as shown in diagram A.18, the installer will attempt to automatically detect the video display card in use and recommend the appropriate driver software for it.

Click  to continue.

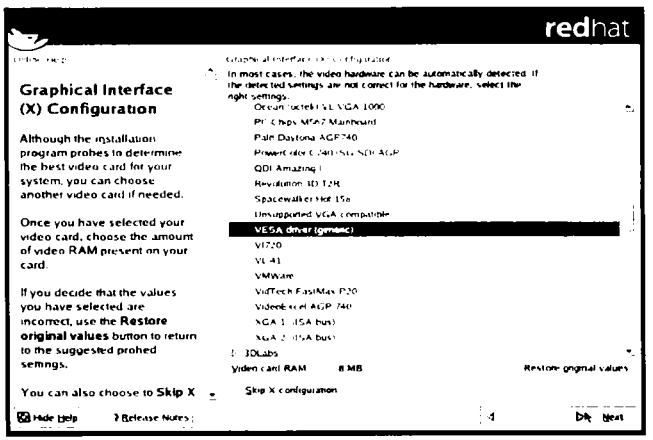


Diagram A.18: The Graphical Interface (X) Configuration screen.

The **Monitor Configuration** screen appears as shown in diagram A.19. The installer probes the VDU and recommends an appropriate device driver for it.

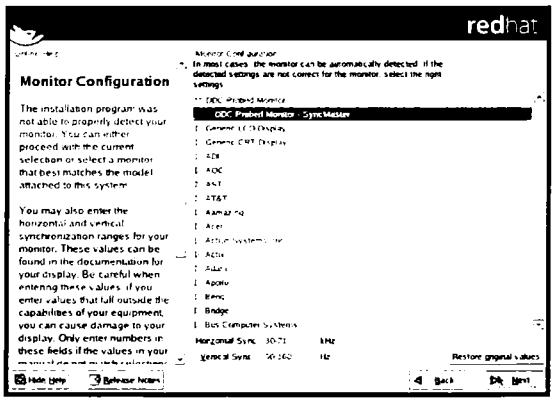


Diagram A.19: The Monitor Configuration screen.

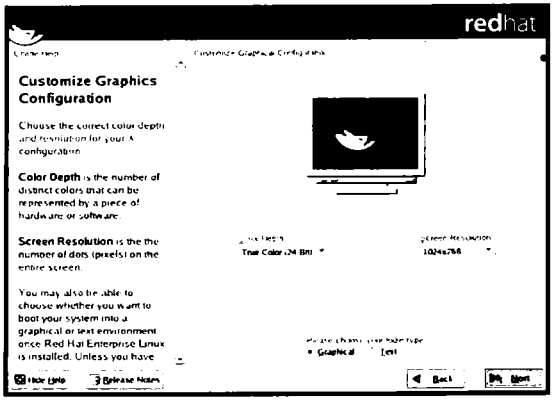


Diagram A.20: The Customize Graphics Configuration screen.

Click Next. The **Customize Graphics Configuration** screen appears as shown in diagram A.20. This screen allows the setting of the resolution and color depth of the VDU.

The last screen is the **Congratulations** screen as shown in diagram A.21. Click Exit to complete the installation of Red Hat AS 3.0 Linux.

After Linux installation is complete the computer automatically restarts.

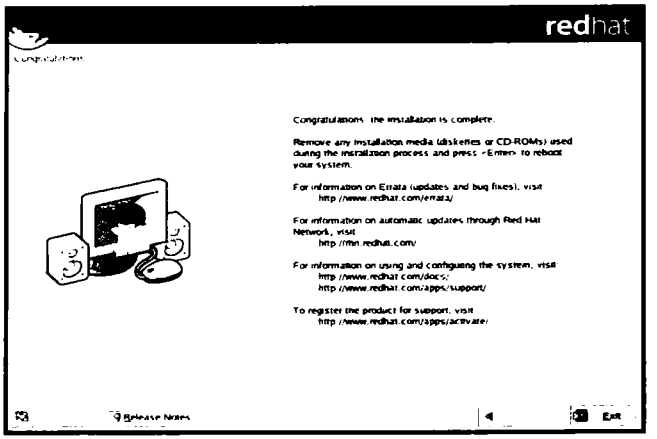


Diagram A.21: Installation completed screen.

REMINDER

Remove the CD from the computer's CD-ROM. Use **Setup** and instruct the CPU to boot from the HDD. Save these changes and Exit from Setup.

Using The Boot Loader Software

On boot up the Boot Loader software activates. This software provides the option of selecting an operating system. To select **Red Hat AS 3.0 Linux** use the down arrow key. Then press Enter. Refer diagram A.22.

REMINDER

The Boot Loader will automatically load the operating system **set as default** within **10 seconds** of displaying these option(s).

Diagram A.22: The Boot Loader screen.

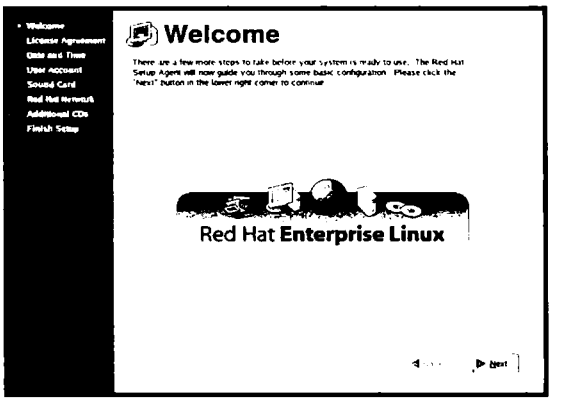


Diagram A.23.1: The Welcome screen.

When the Linux O/s is loaded, the **Welcome** screen for Red Hat AS 3.0 Linux appears. Refer diagram A.23.1. *This setting appears only when Red Hat AS 3.0 Linux starts for the first time.*

Click **Next** to get to the **License Agreement** screen as shown in diagram A.23.2.

The **Date and Time** screen will appear as shown in diagram A.23.3. Set the system calendar and clock, if required and click **Next**.

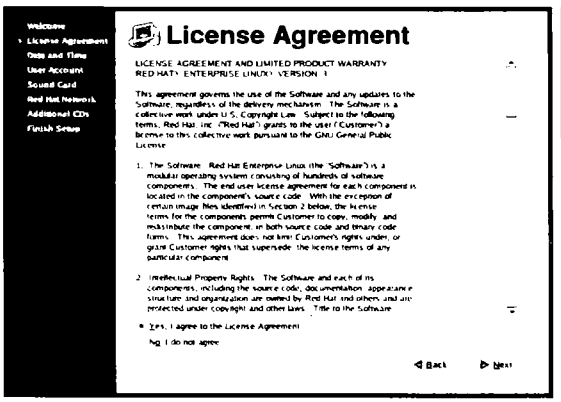


Diagram A.23.2: The License Agreement screen.

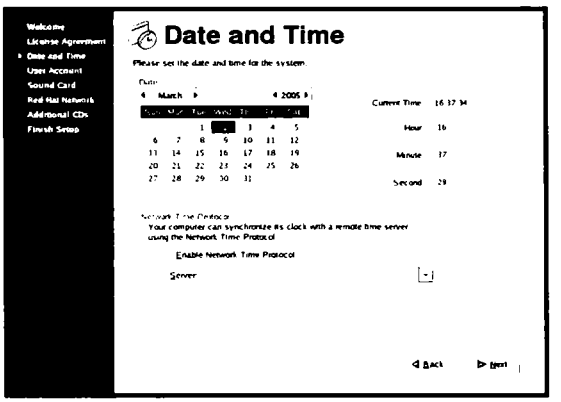


Diagram A.23.3: The Date and Time screen.

Click **Next** to get to the **User Account** screen as shown in diagram A.23.4.

Enter user information (**i.e.** the Login name, Full name and Password) to create a user under Linux. When done click **Next** to continue.

If a sound card is present on the computer the screen as shown in diagram A.23.5 will appear. This screen detects/tests the Sound Card on the computer.

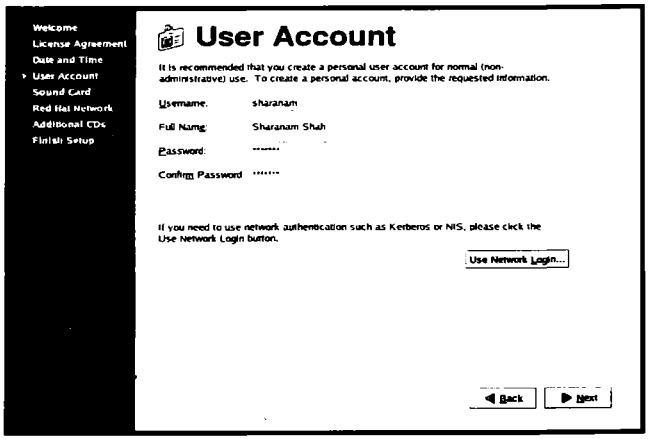


Diagram A.23.4: The User Account screen.

Clicking **Next** displays the screen as shown in diagram A.23.6. This screen allows the user to register the system with the Red Hat Network. Select the desired option and click **Next.**

The next screen is the **Additional CDs** screen. If required, Linux documentation and additional software installation from the CDs can be done here.

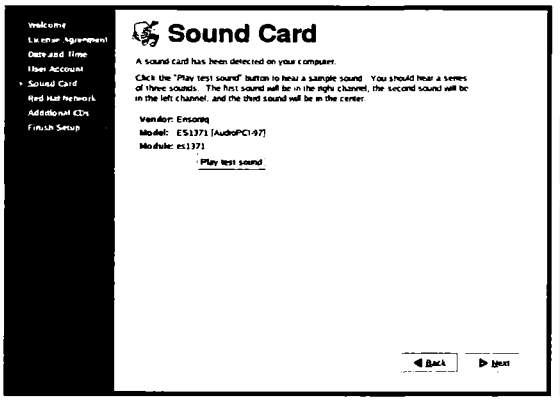


Diagram A.23.5: The Sound Card screen.

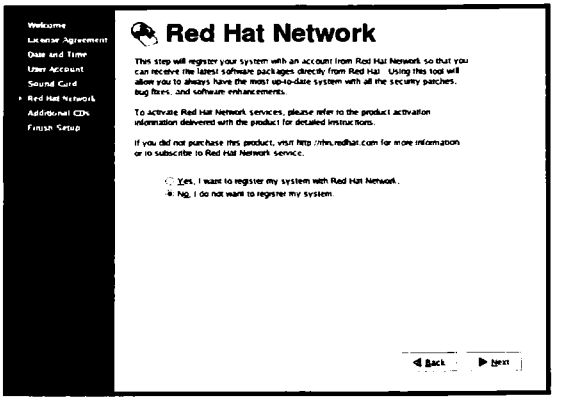


Diagram A.23.6: The Red Hat Network screen

To continue without installing any further components click **Next.**

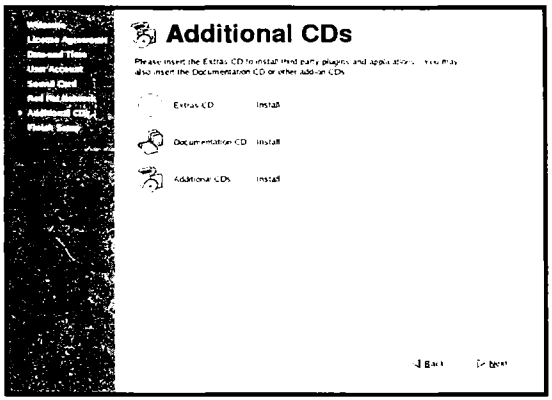


Diagram A.23.7: The Additional CDs screen.

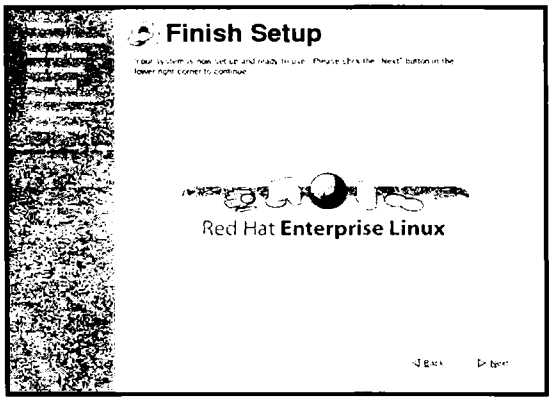


Diagram A.23.8: The Finish Setup screen.

The last screen is the **Finish Setup** screen. Click **Next** to complete the setup.

The Linux startup process will continue for a few minutes and its Graphical Login screen will be displayed. Refer diagram A.24.1.

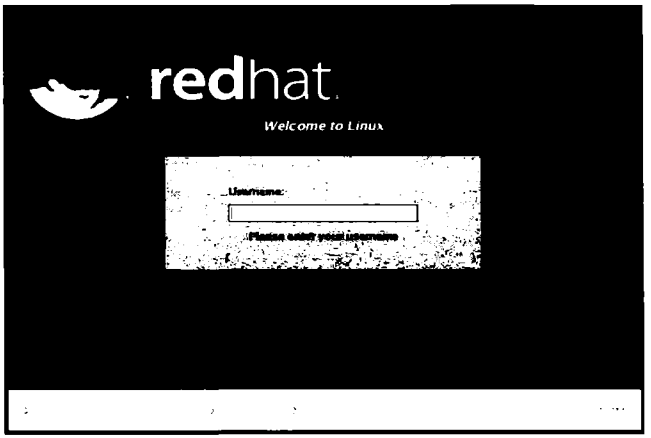


Diagram A.24.1: The Graphical login screen.

The menu bar as seen in bottom section of the screen provides the following options:

- **Language** – Change the default language for running Linux
- **Session** – Change the session manager (i.e. **GNOME** or **KDE**)
- **Reboot** – To close Linux and restart the computer
- **Shutdown** – To close Linux and switch off the computer

Click **Session** to get a popup menu as shown in diagram A.24.2. The pop up menu box permits a change in the current session.

Select **KDE** and click OK to return to the login dialog box.

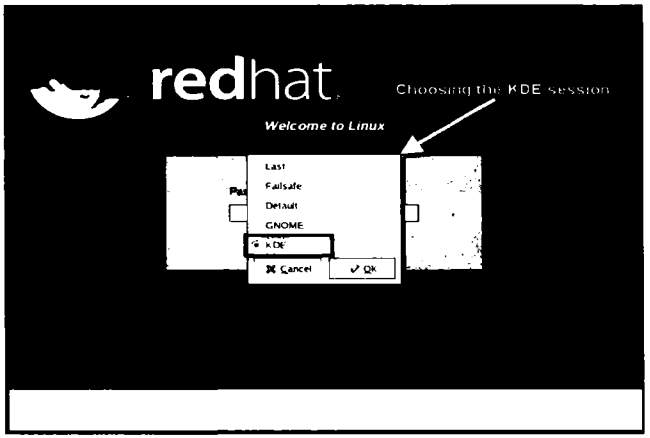


Diagram A.24.2: Selecting the Linux Session manager.

Enter **root** as the username and press **Enter**. Refer diagram A.24.3.

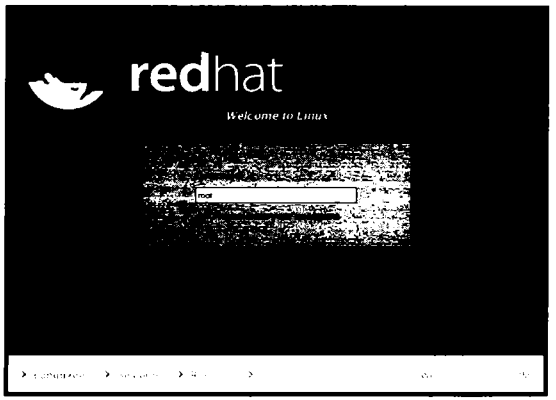


Diagram A.24.3: Enter the user name.

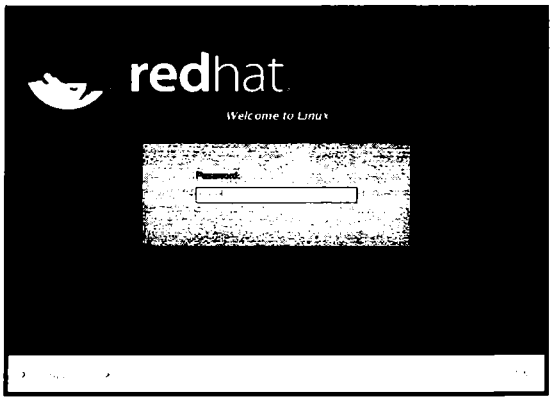


Diagram A.24.4: Enter the password.

The login dialog box will change to accept a password (previously bound to the user name). Refer diagram A.24.4.

Key-in an authentic password and press **Enter**. After authentication the KDE Session Manager is activated as shown in diagram A.25.1.

Diagram A.25.1: KDE starting up.

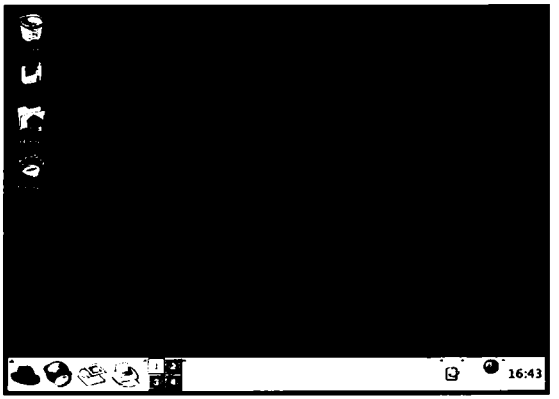


Diagram A.25.2: The Linux desktop.

Finally, the Linux desktop will appear as shown in diagram A.25.2.

Linux is now loaded in memory and is ready for use.

D

M

N

O

P

Q

R

S

T

Aviation

8173660867	Acceptable Methods, Techniques, & Practices - Aircraft Inspection & Repair, *656 Pages*	FAA/SPD	425.00
8181479661	**Aerodynamics for Engineering Students, 5/ed, *606 Pages***	**Houghton**	**400.00**
0534393845	Air Transporation: A Management Perspective, 5/ed, *648 Pages*	Wells	950.00
0632040696	Air Travel: How Safe is it?, 2/ed, *304 Pages*	Taylor	1,080.00
8175980419	**Aircraft Construction Repair & Inspection, *236 Pages***	**Christy**	**150.00**
817598001X	Aircraft Instruments & Integrated Systems, *456 Pages*	Pallett	450.00
_____	Aircraft Maintenance Engineer's (Personal Log Book), *108 Pages*	Sterling	100.00
817598046X	**Aircraft Manual (India) Volume - I: Bare Act with Short Notes 2005, *290 Pages***	**SBH**	**150.00**
8175980478	**Aircraft Manual (India) Volume - II: Bare Act with Short Notes 2005, *396 Pages***	**SBH**	**250.00**
8175980486	**Aircraft Manual (India) Volume - I & II: Bare Act with Short Notes 2005**	**SBH**	**350.00**
8175980133	Aircraft Materials & Processes, 5/ed, *406 Pages*	Titterton	225.00
	Airframe & Powerplant Mechanics		
8173660101	Airframe Handbook - AC 65 -15A, *644 Pages*	FAA	250.00
8173662835	General Handbook - AC 65 - 9A: 2000 Edition, *576 Pages*	FAA	250.00
8173660093	Powerplant Handbook - AC 65 - 12A, *532 Pages*	FAA	250.00
1405135417	**Automatic Flight Control, 4/ed, *336 Pages***	**Pallett**	**400.00**
8175980451	**Avionics Vol.1: Every Pilot's Guide to Aviation Electronics, *284 Pages***	**Ferrara**	**200.00**
8175980257	Basic Synchros & Servomechanism Parts 1 & 2, *250 Pages*	Valkenburg	150.00
0632052953	Climatology for Airline Pilots, *296 Pages*	Quantick	1,150.00
8175980494	**COMPACT: Civil Aviation Requirements Section II (Airworthiness & Aircraft Rules & AAC, *870 Pages***	**SBH**	**675.00**
8175980508	**Director General of Civil Aviation: Civil Aviation Requirements Section - 2, (Part I & II) Airworthiness, *1,310 Pages***	**SBH**	**1,000.00**
9122262355	Flight Crew Log Book, *202 Pages*	Sterling	95.00
8175980001	Flight Without Formulae, 5/ed, *320 Pages*	Kermode	175.00
0813808545	**Fly The Wing 3/ed (B / CD-ROM), *256 Pages***	**Webb**	**700.00**
0534393888	Fundamentals of Air Traffic Control 4/ed, *596 Pages*	Nolan	900.00
0632055731	Ground Studies for Pilots: Radio Aids 6/ed, *312 Pages*	Underdown	1,100.00
0632059397	Ground Studies for Pilots: Flight Planning 6/ed, *264 Pages*	Underdown	1,100.00
063205333X	Ground Studies for Pilots: Navigation, 6/ed, *371 Pages*	Underdown	1,100.00
0632054840	Ground Studies for Pilots: Meteorology, 3/ed, *294 Pages*	Underdown	1,100.00
0632059516	Ground Studies for Pilots: Flight Instruments & Automatic Flight Control Systems, 6/ed, *232 Pages*	Harris	1,100.00
8175980435	**Helicopter Aerodynamics, *178 Pages***	**R.W.Prouty**	**150.00**
0534393756	An Invitation to Fly: Basics For the Private Pilot, 7/ed, *666 Pages*	Glaeser	1,200.00
8175980427	**Light Aircraft Maintenance, *216 Pages***	**Heywood**	**100.00**
0632034726	Manual of Avionics, 3/ed, *304 Pages*	Kendal	1,600.00
0222631572	Personal Flying Logbook (Aircraft Operating Crew) - Deluxe Binding, *344 Pages*	Sterling	350.00
8175980222	RotorCraft Flying Handbook, *201 Pages*	FAA	250.00

Biological Terrorism

1401809871	Preparing for Biological Terrorism, *395 Pages*	Buck	1,050.00

Business, Management & Finance

8175980311	45 Years in Wall Street, *160 Pages*	Gann	240.00

0471430463	**101 Reasons to Own the World's Greatest Investment: Warren Buffett's Berkshire Hathaway, *272 Pages***	**Miles**	**$ 14.95**
047146256X	**Advanced Swing Trading: Strategic to Predict, Identify, and Trade Future Market Swings, *240 Pages***	**Crane**	**$ 69.95**
0471295639	**Against the Gods: The Remarkable Story of Risk, *400 Pages***	**Bernstein**	**$ 19.95**
0471445495	**The Alchemy of Finance, *416 Pages***	**Soros**	**$ 19.95**
9812530118	The Analysis and Use of Financial Statements, 3/ed **(B / CD-ROM)**, *794 Pages*	White	750.00
0631215905	Applied Derivatives: Options, Futures and Swaps, *400 Pages*	Rendleman	1,500.00
1403916578	The Asian Insider: Unconventional Wisdom for Asian Business, *306 Pages*	Backman	£ 25.00
9812549676	Bank Management, 5/ed, *916 Pages*	Koch	$ 10.50
1403933154	Big In Asia: 25 Strategies for Business Success, *352 Pages*	Backman	£ 19.99
0471718890	**The Black Book of Outsourcing: How to Manage the Changes, Challenges, and Opportunites, *390 Pages***	**Brown**	**$ 29.95**
0324015658	Business Analysis & Valuation Using Financial Statement, 2/ed, *1,107 Pages*	Palepu	2,000.00
0324024207	Capital Markets, *621 Pages*	Liaw	1,100.00
0471445509	Common Stocks and Uncommon Profits and Other Writing Markets, *320 Pages*	Fisher	$ 19.95
0324200544	Credit Scoring for Risk Managers: The Handbook for Leaders, *270 Pgs*	Mays	650.00
817598029X	The Crowd: A Study of The Popular Mind, *244 Pages*	Le Bon	260.00
0471304654	Damodaran on Valuation, *454 Pages*	Damodaran	$ 60.00
0471446912	**Dawn Right! : Behind the Scenes with Berkshire Hathaway Billionaire Charlie Munger, *294 Pages***	**Lowe**	**$ 16.95**
8173660999	Doing Business with the French, *150 Pages*	Jhangiani	150.00
8175980338	The Dow Theory, *266 Pages*	Rhea	290.00
0471249483	**The Education of a Speculator, *444 Pages***	**Niederhoffer**	**$ 19.95**
0470820780	**The Essays of Warren Buffett: Lessons for Investos and Managers Revised Edition, *292 Pages***	**Cunningham**	**$ 19.95**
047140179X	**Event Marketing : How to Successfully Promote Events, Festivals, Conventions, and Expositions, *256 Pages***	**Hoyle**	**$ 50.00**
0932750435	Elliott Wave Principles, 20th Anniversary Edition, *248 Pages*	Frost	$ 34.95
047122703X	**The Essential Buffett: Timeless Principles for the New Economy, *270 Pages***	**Hagstrom**	**$ 19.95**
8173665982	Ethics, Indian Ethos and Management, *252 Pages*	Balachandran	125.00
0324178174	Financial Institutions, Investments, and Management: An Introduction, 8/ed , *648 Pages*	Mayo	750.00
0471267686	Financial Modeling Using Excel & VBA **(B / CD-ROM)**, *672 Pages*	Sengupta	$ 65.00
0471686174	**The Five Rules for Successful Stock Investing : Morningstar's Guide to Building Wealth and Winning in the Market, *364 Pages***	**Dorsey**	**$ 16.95**
158799190X	Fooled by Randomness: The Hidden Role of Chance in Life and in the Markets, 2/ed, *310 Pages*	Taleb	$ 27.95
0631232400	Futures, Options and Swaps, 4/ed **(B / CD-ROM)**, *910 Pages*	Kolb	900.00
0470014989	**Hot Commodities: How Anyone Can Invest Profitably in the World's Best Market, *276 Pages***	**Rogers**	**$ 20.50**
0471416177	**How to be a Billionaire: Proven Strategies From the Titans of Wealth, *320 Pages***	**Fridson**	**$ 18.95**
0471710490	**How to Make Money Selling Stocks Short, *195 Pages***	**O'Neil**	**$ 19.95**
0934380759	How to Trade in Stocks, *256 Pages*	Livermore	$ 29.95
9812438955	Introduction to Derivatives & Risk Management, 6/ed, *696 Pages*	Chance	500.00
9812406409	Investment Analysis & Portfolio Management, 7/ed, *1,276 Pages*	Reilly	750.00
0030339294	Investment, 6/ed, *818 Pages*	Reilly	1,200.00
0324180071	Investments: An Introduction, 7/ed, **(B / CD-ROM)**, *1,182 Pages*	Mayo	1,700.00
817366885X	Logistics in International Business, *334 Pages*	Aserkar	325.00
0471397741	**J.K. Lasser's Pick Stocks Like Warren Buffett, *304 Pages***	**Boroson**	**$ 16.95**
817598032X	The Japanese Chart of Charts, *228 Pages*	Shimizu	285.00
0471389455	**Manias, Panics, and Crashes: A History of Financial Crises, 4/ed, *304 Pages***	**Kindleberger**	**$ 19.95**

8173669252	**Master Scheduling in the 21st Century, *228 Pages***	**Wallace**	**250.00**
8175980303	The Plunderers, *344 Pages*	LeFevre	300.00
0471677744	**Practical Speculation, *398 Pages***	**Niederhoffer**	**$ 18.95**
0471003786	**The Power of Gold : The History of an Obsession, *448 Pages***	**Bernstein**	**$ 16.95**
0471678767	**Reminiscences of a Stock Operator Illustrated, *254 Pages***	**Lefevre**	**$ 29.95**
0538861010	Risk Management & Derivatives, *708 Pages*	Stulz	1,400.00
9812403736	Risk Management & Insurance, 11/ed, *652 Pages*	Trieschmann	500.00
8173661987	**Sales & Operations Planning: The How - To Handbook, 2/ed, *200 Pages***	**Wallace**	**250.00**
8173669996	**Sales Forecasting: A New Approach, *188 Pages***	**Wallace**	**250.00**
0471450375	**Special Events : Event Leadership for a New World, 4/ed, *528 Pages***	**Goldblatt**	**$ 65.00**
8173668817	**Strategic Bidding: A Successful Approach (for IT / Consultancy), *192 Pages***	**Garg**	**250.00**
0470821094	Swaps & Financial Derivatives: Products, Pricing, Application & Risk Management, 3/ed, **(B / CD-ROM)**, (4 Volumes) *4,700 Pages*	Das	11,000.00
0471655856	**Trade Like Jesse Livermore, *236 Pages***	**Smitten**	**$ 40.00**
0471655848	**Trade Like Warren Buffett, *246 Pages***	**Altucher**	**$ 49.95**
0471463396	**Value Investing: From Graham to Buffett & Beyond, *320 Pages***	**Greenwald**	**$ 19.95**
0471361917	Valuation Measuring & Managing the Value of Companies, 3/ed, *512 Pages*	Copeland	$ 60.00
0471702196	**Valuation with (B / CD-ROM): Measuring & Managing the Value of Companies, 4/ed, *772 Pages***	**Koller**	**$ 200.00**
0471430455	**The Warrent Buffett CEO: Secrets from the Berkshire Hathaway Managers, *432 Pages***	**Miles**	**$ 16.95**
0471392642	**The Warrent Buffett Portfolio: Mastering the Power of the Focus Investment Strategy, *256 Pages***	**Hagstrom**	**$ 16.95**
0470821531	**Warren Buffett: An Illustrated Biography of the World's Most Successful Investor, *120 Pages***	**Morio**	**$ 14.95**
0471271144	**What I Learned Before I Sold to Warrent Buffett: An Entrepreneur's Guide to Developing a Highly Successful Company, *272 Pages***	**Helzberg Jr.**	**$ 24.95**
0471119784	**Where Are the Customers' Yachts? or A Good Hard Look at Wall Street, *256 Pages***	**Schwed**	**$ 19.95**

FORTHCOMING TITLES

8173666792	Rules of Origin in International Trade, *000 Pages*	Dr.Sathpathy	TBA
8173668531	Futures & Options, *000 Pages*	Sridhar	TBA

Catering & Hotel Management

0471359262	**The Advanced Professional Pastry Chef, 4/ed, *800 Pages***	**Friberg**	**$ 65.00**
047145785X	**The Book of Yields : Accuracy in Food Costing and Purchasing, 6/ed, *288 Pages***	**Lynch**	**$ 37.65**
8173668736	Careers in Hospitality - Hotel Management Entrance Exam Guide, *332 Pages.*	Rego	200.00
0766826864	Dining Room & Banquet Management, 3/ed, *440 Pages*	Strianese	400.00
818147810X	**Escoffier: The Complete Guide to the Art of Modern Cookery, *672 Pages***	**Craknell**	**950.00**
9812406174	Front Office Operations & Management, *376 Pages*	Ismail	260.00
0471468495	**Garde Manger, The Art and Craft of the Cold Kitchen, *592 Pages***	**CIA**	**$ 65.00**
8175980214	Hering's Dictionary of Classical & Modern Cookery, 13/ed, *866 Pages*	Bickel	450.00
9812402101	Introduction to Catering, *416 Pages*	Shiring	295.00
8181478045	**The Larder Chef: Food Preparation and Presentation, 3/ed, *322 Pages***	**M. J. Leto**	**300.00**
9812431160	Management Accounting for Hospitality & Tourism, 3/ed, *340 Pages*	Kotas	325.00
1861528825	Practical Professional Catering Management, 2/ed, *560 Pages*	Cracknell	650.00
0471382574	**The Professional Chef, 7/ed, *1,056 Pages***	**CIA**	**$ 70.00**
0471359254	**The Professional Pastry Chef: Fundamentals of Baking and Pastry, 4/ed, *1,040 Pages***	**Friberg**	**$ 65.00**
0471572284	**The Sauce Bible : Guide to the Saucier's Craft, *400 Pages***	**Larousse**	**$ 54.95**
1401819826	Selling Destinations Geography for the Travel Professional, 4/ed, *566 Pages*	Mancini	800.00
0766845931	Training Desiging for the Hospitality Industry, *216 Pages*	Jaszay	350.00

8175980265	The Waiter, *224 Pages*	Fuller	200.00
9812402071	Welcome to Hospitality An Introduction, 2/ed, *480 Pages*	Chon	320.00

Civil Engineering

8173669570	**Concrete Bridge Practice: Analysis, Design and Economics, 2/ed, *802 Pages* [Size 8.5 x 11, 8 Full Color Inserts, Hardbound]**	**Dr. V. K. Raina**	**800.00**
8173669597	**Concrete Bridge Practice: Construction, Maintenance and Rehabiliation, *464 Pages* [Size 8.5 x 11, 14 Full Color Inserts, Hardbound]**	**Dr. V. K. Raina**	**525.00**
8173669767	**Concrete Bridge: Inspection, Repair, Strengthening, Testing, Load Capacity Evaluation, *546 Pages* [Size 8.5 x 11, 32 Full Color Inserts, Hardbound]**	**Dr. V. K. Raina**	**675.00**
8173669775	**Concrete for Construction: Facts and Practice, 272 Pages [Size 7 x 9.5]**	**Dr. V. K. Raina**	**250.00**
8173669783	**Construction Management Practice: Inside Story, *136 Pages* [Size 7 x 9.5]**	**Dr. V. K. Raina**	**150.00**

FORTHCOMING TITLE

8173666806	Fields Manual for Highway and Bridge Engineers, *000 Pages*	Dr. V. K. Raina	TBA

Computers

Shroff Original Titles
Launching The X Team Series
(An Imprint of Shroff Publishers)

PUBLISHED TITLE

8173666849	**Application Development with Oracle & PHP on Linux for Beginners (B / CD-ROM), *828 Pages***	**Bayross**	**550.00**

FORTHCOMING TITLES

8173670048	Eclipse 3.1 for Beginners, *600 Pages*	Bayross	TBA
8173670056	J2EE 5 Server Programming for Professionals, *1,400 Pages*	Bayross	TBA
8173670064	Linux Programming for Beginners, *800 Pages*	Bayross	TBA
8173670153	MySQL 5 for Beginners, *600 Pages*	Bayross	TBA
8173670188	Oracle 10g for Beginners, *1,000 Pages*	Bayross	TBA
8173670218	Oracle 10g for Professionals, *1,200 Pages*	Bayross	TBA
8173670242	PHP 5.1 for Beginners, *600 Pages*	Bayross	TBA
8173670404	PHP 5.1 for Professionals, *800 Pages*	Bayross	TBA

Coming Soon under The X Team Series

Microsoft Platform

8173670773	.NET 2.0 Framework for Professionals, *000 Pages*	TBA	TBA
8173670781	.NET 2.0 Remoting for Professionals, *000 Pages*	TBA	TBA
8173670498	ASP .NET 2.0 with VB 2005 & Visual C# 2005 for Beginners, *000 Pgs*	TBA	TBA
8173670501	ASP .NET 2.0 with VB 2005 & Visual C# 2005 for Professionals, *000 Pages*	TBA	TBA
817367079X	SQL Server 2005 for Beginners, *000 Pages*	TBA	TBA
8173670803	SQL Server 2005 for Professionals, *000 Pages*	TBA	TBA
8173670811	SQL Server 2005 Reporting Services for Professionals, *000 Pages*	TBA	TBA
8173670757	Visual C# 2005 for Beginners, *000 Pages*	TBA	TBA
8173670765	Visual C# 2005 for Professionals, *000 Pages*	TBA	TBA
8173670730	Visual Basic 2005 for Beginners, *000 Pages*	TBA	TBA
8173670749	Visual Basic 2005 for Professionals, *000 Pages*	TBA	TBA
817367082X	Visual Studio 2005 Team Systems for Professionals, *000 Pages*	TBA	TBA

Java Platform

8173670838	EJB 3 for Professionals, *000 Pages*	TBA	TBA
8173670846	J2EE 5 for Beginners, *000 Pages*	TBA	TBA
8173670854	JSP 3 for Programmers, *000 Pages*	TBA	TBA

8173670862	Spring 1.2 for Beginners, *000 Pages*	TBA	TBA
Open Sources			
8173670870	Apache Tomcat 5.5 for Professionals, *000 Pages*	TBA	TBA
8173670889	XML for Professionals, *000 Pages*	TBA	TBA

Other Titles

8173669600	**The Art of Creative Destruction: Illustrated Software Testing & Test Automation, *280 Pages***	**Puranik**	**250.00**
8173669988	**Database Concepts and Systems, 2/ed, *292 Pages***	**Bayross**	**275.00**
8173667942	Operating System Concepts and Principles, *346 Pages*	George	275.00
8173668019	Software Defect Prevention Concepts and Implementation, *180 Pages* (Hardbound)	Kane & Bajaj	300.00
8173668817	Strategic Bidding: A Successful Approach (for IT / Consultancy), *192 Pages*	Garg	250.00
FORTHCOMING TITLES			
8173666806	First Encounter with Java Including BlueJ, *000 Pages*	Shefali Bhutta	TBA
8173670897	Database Managment Systems - Core Concept, *000 Pages*	Ghosh	TBA

Shroff / Apress

(Apress Original Titles in Indian Reprints)

PUBLISHED TITLES			
8181281101	.NET Game Programming with DirectX 9.0, VB .NET Edition **(B / CD-ROM)**, *654 Pages*	Lobao	550.00
8181280563	.NET System Management Service, *442 Pages*	Golomshok	400.00
8181280539	Advanced .NET Remoting, C# Edition, *424 Pages*	Rammer	375.00
8181280555	Applied ADO .NET: Building Data Driven Solutions, *956 Pages*	Chand	675.00
8181281063	ASP .NET 2.0 Revealed, *408 Pages*	Lorenz	375.00
8181280547	ATL Server: High Performance C++ on .NET, *520 Pages*	Kumar	450.00
8181280903	Automating Unix and Linux Administration, *600 Pages*	Bauer	500.00
8181281713	Beginning ASP .NET 1.1 in VB .NET from Novice to Professional,*1,008 Pages*	MacDonald	675.00
8181280989	C # Programmer's Handbook, *528 Pages*	Pearce	500.00
8181281314	Code Generation in Microsoft .NET, *760 Pages*	Dollard	525.00
8181280598	Comprehensive VB .NET Debugging, *528 Pages*	Pearce	450.00
1590590937	Database Programming with C#, *696 Pages*	Thomsen	400.00
8181280784	Debugging Strategies for .NET Developers, *272 Pages*	Dillon	250.00
818128108X	Designing Scalable .NET Applications, *568 Pages*	Rossberg	475.00
8181280741	Developing .NET Enterprise Applications, *528 Pages*	Kanalakis	425.00
8181280660	Enterprise Java for SAP, *336 Pages*	Sincock	325.00
8181281705	Essential PHP Tools: Modules, Extensions, and Accelerators, *368 Pages*	Sklar	350.00
8181281055	Introduction to 3D Game Engine Design Using DirectX 9 and C#, *424 Pages*	Harrison	375.00
8181281667	Ivor Horton's Beginning ANSI C++: The Complete Language, 3/ed, *1,118 Pages*	Horton	750.00
8181281675	Java Regular Expressions, *276 Pages*	Habibi	325.00
8181281098	JBoss 3.2 Deployment and Administration, *368 Pages*	Kunnumpurath	325.00
8181280768	Learn VB .NET Through Game Programming, *400 Pages*	Tagliaferri	350.00
8181280806	Managed C++ and .NET Development, *976 Pages*	Fraser	675.00
8181281330	Mastering Oracle PL/SQL Pracatical Solutions (Preview 10g), *648 Pages*	McDonald	475.00
8181281047	Maximizing .NET Performance, *304 Pages*	Wienholt	275.00
8181281209	Maximizing Performance Scalability with IBM WebSphere (covers Version 4 & 5), *576 Pages*	Neat	500.00
8181281322	Oracle JDeveloper 10g Empowering J2EE Development, *294 Pages*	Oak	325.00
8181280687	Peer - to - Peer with VB .NET, *440 Pages*	MacDonald	400.00
8181280776	Performance Tuning and Optimizing ASP .NET Applications, *392 Pages*	Hasan	375.00
8181280946	Practical Python, *648 Pages*	Hetland	525.00
8181281071	Pro JSP, 3/ed (covers 2.0 and Servlet 2.4), *624 Pages*	Brown	350.00

8181280954	The Qmail Handbook, *512 Pages*	Sill	425.00
8181280792	Real World .NET Applications, *624 Pages*	Kurniawan	500.00
8181280679	Real World ASP .NET Best Practices, *224 Pages*	Muhammad	225.00
818128058X	Real World Enterprise Reports Using VB6 and VB .NET, *692 Pages*	Ganz, Jr	550.00
8181280210	Software Excorcism: A Handbook for Debugging and Optimizing Legacy Code, *376 Pages*	Blunden	325.00
1590590759	The Sun Certified Java Developer Exam with J2SE 1.4, *384 Pages*	Habibi	300.00
8181280571	Web Services Patterns: Java Edition, *356 Pages*	Monday	325.00
8181280601	WebSphere Studio Application Developers 5.0: Practical J2EE Development, *656 Pages*	Livshin	525.00

Shroff / Apress

(Formerly under Wrox Press)

TITLES AT REDUCED PRICES

8173661448	ADO 2.6 Programmer's Reference, *720 Pages*	Sussman	200.00
8173661464	ASP 3.0 Programmers Reference, *1,344 Pages*	Anderson	250.00
8173661502	Beginning ASP Databases, *868 Pages*	Kauffan	250.00
8173661529	Beginning Components for ASP, *868 Pages*	Anderson	250.00
817366143X	Beginning E-Commerce with VB, ASP, SQL Server 7.0 & MTS, *802 Pages*	Reynolds	250.00
8173661545	Beginning Java 2 - JDK 1.3 Edition, *1,272 Pages*	Horton	375.00
817366160X	Beginning Java Object: From Concepts to Code, *696 Pages*	Barker	200.00
8173661642	Beginning XHTML, *768 Pages*	Boumphrey	250.00
8173663238	Data-Centric .NET Programing with C#, *812 Pages*	Ferracchiati	325.00
8173661707	Enterprise Application Architecture with VB, ASP, MTS, *816 Pages*	Moniz	200.00
8173661685	Java Programmer's Reference, *1.252 Pages*	Palmer	325.00
8173661855	JavaScript Programmer's Reference **(B / CD-ROM)**, *1,004 Pages*	Wootton	275.00
817366157X	Professional Application Centre 2000, *432 Pages*	Homer	175.00
8173661634	Professional ASP Data Access, *1,340 Pages*	Esposito	375.00
8173662959	Professional ASP XML, *920 Pages*	Baartse	250.00
8173662444	Professional BizTalk, *732 Pages*	Mohr	200.00
8173662967	Professional ColdFusion 5, *1,240 Pages*	Ambrose-Haynes	350.00
8173663505	Professional Commerce Server 2000 Programming, *1,126 Pages*	Wrox	350.00
8173661898	Professional DCOM Application Development, *496 Pages*	Pinnock	150.00
8173661901	Professional DCOM Programming, *592 Pages*	Grimes	150.00
8173662452	Professional J2EE Programming with BEA Web Logic Server, *538 Pages*	Gomez	100.00
8173661863	Professional Java Programming, *1,144 Pages*	Spell	275.00
8173662029	Professional Site Server 3.0 Commerce Edition, *736 Pages*	Tabini	250.00
8173663084	Professional Windows DNA: Building Distributed Web Applications with Visual Basic, COM+, MSMQ, SOAP & ASP, *1,016 Pages*	Blexrud	350.00

PUBLISHED TITLES

8173664692	.NET Compact Framework, *188 Pages*	Morris	150.00
8173661456	.NET Enterprise Development in C#: From Design to Deployment, *474 Pages*	Reynolds	325.00
8173661553	.NET Enterprise Development in VB .NET: From Design to Deployment, *470 Pages*	Reynolds	350.00
8173666342	A C# Application from Inspiration to Implementation, *382 Pages*	Dunn	350.00
8173666946	Apache Tomcat Security Handbook, *244 Pages*	Chopra	250.00
8173666407	ASP To ASP .NET Migration Handbook, *312 Pages*	Conway	275.00
8173664862	ASP .NET 1.0 with C#: Namespace Reference, *966 Pages*	Kalani	650.00
8173664870	ASP .NET 1.0 with VB .NET: Namespace Reference, *924 Pages*	Kalani	650.00
8173662355	ASP .NET Distributed Data Applications, *680 Pages*	Homer	500.00
8173666415	ASP .NET E-Commerce Programming; Problem – Design - Solution: C# Edition, *372 Pages*	Hoffman	350.00

8173662436	Axis: The Next Generation of Java SOAP, *294 Pages*	Irani	225.00
8173666318	BEA WebLogic Server 7.0 Handbook: Deployment and Administration, *456 Pages*	Mulder	375.00
8173666466	Beginning .NET Web Services using C#, *368 Pages*	Bustos	325.00
8173664951	Beginning .NET Web Services using VB .NET, *360 Pages*	Bustos	300.00
8173664889	Beginning ASP .NET 1.0 Databases using C#, *476 Pages*	Kauffman	350.00
8173661510	Beginning ATL 3 COM Programming, *548 Pages*	Grimes	450.00
8173664897	Beginning C# Databases, *664 Pages*	Allen	500.00
8173666350	Beginning C# Web Applications with Visual Studio .NET, *628 Pages*	Cazzulino	550.00
8173662827	Beginning Databases with MySQL, *600 Pages*	Matthew	500.00
8173664137	Beginning Databases with PostgreSQL, *592 Pages*	Stones	400.00
8173666970	Beginning J2EE 1.4, *740 Pages*	Mukhar	475.00
8173663998	Beginning Java Networking, *832 Pages*	Darby	550.00
8173666458	Beginning Java Web Services, *500 Pages*	Bequet	400.00
8173666385	Beginning JSP 2.0: Build Web Applications Using JSP, Java, and Struts, *866 Pages*	Galbraith	500.00
8173662738	Beginning Oracle Programming, *1,132 Pages*	Dillon	675.00
8173661774	Beginning Perl, *704 Pages*	Cozens	500.00
8173661669	Beginning SQL Programming **(B / CD-ROM)**, *752 Pages*	Kauffman	575.00
8173663181	Beginning SQL Server 2000 for Visual Basic Developers, *888 Pages*	Willis	600.00
8173663467	Beginning SQL Server 2000 Programming **(B / CD-ROM)**, *796 Pages*	Dewson	600.00
817366420X	Beginning VB .NET Databases, *716 Pages*	Forgey	350.00
8173660417	Beginning Visual Basic *6,920 Pages*	Wright's	500.00
8173661626	Beginning Visual Basic 6 Database Programming, *888 Pages*	Connell	600.00
8173662088	Beginning WAP, WML & WMLScript, *676 Pages*	Lee	425.00
8173665958	Beginning Web Programming using VB.NET & Visual Studio .NET, *472 Pages*	Bowes	400.00
817366644X	Building an ASP .NET Intranet, *484 Pages*	Ardestani	425.00
8173666180	C# Class Design Handbook, *388 Pages*	Conway	350.00
8173666121	C# Data Security Handbook, *300 Pages*	MacDonald	325.00
8173661383	C# Programmer's Reference, *582 Pages*	Palmer	475.00
8173666199	C# Threading Handbook, *296 Pages*	Titus	275.00
8173663041	C# Web Services: Building Web Services with .NET Remoting and ASP .NET, *628 Pages*	Banerjee	450.00
8173666296	Dissecting a C# Application: Inside SharpDevelop, *544 Pages*	Holm	475.00
8173664080	Early Adopter HailStorm (.NET My Services), *244 Pages*	Maharry	175.00
8173664048	Early Adopter J2SE 1.4, *218 Pages*	Hart	175.00
817366238X	Early Adopter JSP Standard Tag Library, *322 Pages*	Falkner	250.00
8173662940	Early Adopter JXTA: Peer-to-Peer Computing with Java, *312 Pages*	Li	225.00
8173664005	Early Adopter VoiceXML, *328 Pages*	Andersson	250.00
8173662304	Early Adopter XQuery, *256 Pages*	Cagle	225.00
8173664714	Effective Visual Studio .NET, *600 Pages*	DeLoveh	450.00
8173666016	Expert One-on-One Advanced .NET Programming, *500 Pages*	Robinson	450.00
8173663874	Expert One-on-One Oracle, *1,328 Pages*	Kyte	850.00
8173667020	Expert One-on-One Visual Basic .NET Business Objects, *714 Pages*	Lhotka	600.00
817366496X	Fast Track ADO .NET: C# Edition, *322 Pages*	Ardestani	300.00
8173661928	Fast Track ASP .NET: C# Edition, *356 Pages*	Bellinaso	250.00
8173663033	Fast Track C#, *450 Pages*	Allen	350.00
8173664730	Fast Track VB .NET, *502 Pages*	Hollis	375.00
8173666393	IBM WebSphere 4.0 Application Server Deployment & Administration Handbook, *450 Pages*	Sharman	400.00
817366174X	Implementing LDAP, *518 Pages*	Wilcox	375.00
8173661758	Instant UML, *368 Pages*	Muller	300.00
8173664803	J2EE Design Patterns Applied, *450 Pages*	Berry	300.00
817366322X	Oracle 9i Java Programming: Solutions for Developers Using PL/SQL and Java, *884 Pages*	Holm	575.00

8173666504	PHP MySQL Website Programming: Problem–Design–Solution, *630 Pages*	Lea	500.00
8173666148	Professional .NET for Java Developers with C#, *448 Pages*	Lunn	350.00
8173664013	Professional .NET Framework, *760 Pages*	Hoffman	600.00
8173666369	Professional .NET Network Programming, *508 Pages*	Krowczyk	450.00
8173661790	Professional Active Server Pages 3.0, *1,316 Pages*	Homer	750.00
8173664056	Professional ADO .NET Programming, *746 Pages*	Joshi	600.00
817366613X	Professional ADO .NET with VB .NET, *614 Pages*	Dickinson	525.00
8173664757	Professional Apache 2.0, *902 Pages*	Wainwright	675.00
8173666482	Professional Apache Security, *386 Pages*	Mobily	375.00
8173661154	Professional ASP .NET Security, *472 Pages*	Basiura	375.00
817366451X	Professional ASP .NET Server Controls, *462 Pages*	Butler	350.00
8173664242	Professional ASP .NET Web Services, *792 Pages*	Basiura	600.00
8173664986	Professional ASP .NET Web Services with VB .NET, *808 Pages*	Basiura	550.00
8173666997	Professional C# Design Patterns Applied, *374 Pages*	Fischer	350.00
8173665567	Professional Design Patterns in VB .NET - Building Adaptable Application, *366 Pages*	Fischer	300.00
8173664064	Professional ebXML Foundations, *724 Pages*	Chappell	550.00
8173663513	Professional EJB, *1,200 Pages*	Adatia	750.00
8173663157	Professional J2EE EAI, *958 Pages*	Juric	600.00
8173661022	Professional Java E-Commerce, *1,034 Pages*	Allamaraju	400.00
8173662932	Professional Java Security, *544 Pages*	Garms	350.00
8173660212	Professional Java Server Programming 3/ed - 1.3 J2EE Edition Version, *1,284 Pages*	Allamaraju	675.00
817366255X	Professional Java Servlets 2.3, *740 Pages*	Bell	600.00
8173662908	Professional Java SOAP, *546 Pages*	Bequet	350.00
8173662614	Professional Java Web Services, *514 Pages*	Hendricks	450.00
8173661804	Professional Java XML, *1,188 Pages*	Ahmed	750.00
8173664536	Professional JavaScript, 2/ed, *1,192 Pages*	Baartse	675.00
8173662975	Professional Jini, *928 Pages*	Li	575.00
8173661723	Professional JMS Programming, *800 Pages*	Giotta	350.00
8173662460	Professional JSP Tag Libraries, *492 Pages*	Brown	350.00
8173661820	Professional JSP, 2/ed, *1,228 Pages*	Brown	750.00
8173661960	Professional Linux Deployment, *1,000 Pages*	Prasad	500.00
8173662002	Professional Linux Programming, *1,196 Pages*	Matthew	850.00
8173661979	Professional MFC with Visual C++ 6, *1,248 Pages*	Blaszczak	850.00
8173331812	Professional Oracle8i Application Prog with Java, PL/SQL & XML, *1,248 Pages*	Carnell	575.00
8173663696	Professional Perl Development, *750 Pages*	Kobes	450.00
817366305X	Professional Perl Programming, *1,248 Pages*	Wainwright	850.00
817366630X	Professional PHP Web Services, *502 Pages*	Fuller	500.00
8173666008	Professional SCWCD Certification **(B / CD-ROM)**, *480 Pages*	Dalton	400.00
8173664099	Professional SQL Server 2000 Data Warehousing with Analysis Services, *708 Pages*	Bain	450.00
8173663572	Professional SQL Server 2000 Database Design, *628 Pages*	Davidson	575.00
8173663556	Professional SQL Server 2000 XML, *622 Pages*	Burke	400.00
8173666210	Professional Visual Basic 6: A 2003 Programmer's Resource **(B / CD-ROM)**, *674 Pages*	Ablan	600.00
8173662134	Professional VB .NET Transactions, *552 Pages*	Bortniker	350.00
8173661839	Professional Visual Basic Interoperability - COM & VB6 to .NET, *446 Pages*	Hollis	325.00
817366210X	Professional WAP, *832 Pages*	Wrox	600.00
8173666091	Professional Web Services Security, *630 Pages*	Galbraith	550.00
8173663351	Professional Windows Forms, *708 Pages*	Bell	500.00
8173662037	Professional XML 2/ed, *1,000 Pages*	Birbeck	675.00
8173663122	Professional XML for .NET Developers, *754 Pages*	Dalvi	550.00
8173663076	Professional XSL, *800 Pages*	Cagle	500.00
8173666490	SQL Server 2000 Stored Procedures Handbook, *292 Pages*	Bain	250.00

8173665370	Understanding the .NET Framework, *280 Pages*	Tegels	225.00
817366482X	VB .NET & SQL Server 2000 - Building an Effective Data Layer, *582 Pages*	Bain	475.00
8173661693	Visual Basic .NET Class Design Handbook, *396 Pages*	Olsen	325.00
8173665028	Visual Basic .NET Code Security Handbook, *312 Pages*	Lippert	225.00
8173666431	Visual Basic .NET Debugging Handbook, *332 Pages*	Narkiewicz	300.00
8173665036	Visual Basic .NET Deployment Handbook, *320 Pages*	Delorme	250.00
8173666059	Visual Basic .NET Reflection Handbook, *264 Pages*	Hart	225.00
8173664943	Visual Basic .NET Remoting Handbook, *278 Pages*	Curran	250.00
8173666075	Visual Basic .NET Serialization Handbook, *338 Pages*	Olsen	300.00
8173665583	Visual Basic .NET Solutions Toolkit: 30 Pratical Components for .NET, *466 Pages*	Lhotka	375.00
8173664838	Visual Basic .NET Text Manipulation Handbook, *294 Pages*	Liger	225.00
8173664846	Visual Basic .NET Threading Handbook, *278 Pages*	Ardestani	225.00
8173665974	Visual Basic .NET Windows Services Handbook, *218 Pages*	Conway	175.00
8173665540	Visual C# .NET: A Guide for VB6 Developers, *556 Pages*	Maiani	450.00
8173664765	Visual C++ .NET: A Primer for C++ Developers, *486 Pages*	Corera	350.00
8173664706	Web Services Faceplates, *304 Pages*	Mohr	225.00
8173662290	XML Applications, *600 Pages*	Boumphrey	400.00
8173666474	XML Design Handbook, *332 Pages*	Bonneau	350.00

Auerbach Publication

0849319978	**Information Security Management Handbook, 5/ed, *2,088 Pages***	**Tipton**	**£ 68.00**
0849320321	**Information Technology Control & Audit, 2/ed, *888 Pages***	**Gallegos**	**£ 26.00**
084931707X	Official (ISC)2 Guide to the CISSP Exam **(B / CD-ROM)**, *930 Pages*	Hansche	3,000.00
0849314798	**Software Engineering Handbook, *896 Pages***	**Keyes**	**£ 63.00**
0849325242	**Software Testing and Continuous Quality Improvement (B / CD-ROM), 2/ed, *560 Pages***	**Lewis**	**£ 38.00**

Shroff / Brainysoftware.com

8173664501	How Tomcat Works (Covers 4 & 5), *486 Pages*	Kurniawan	375.00

Shroff / Charles River Media

8173660875	Applied Software Engineering Using Apache Jakarta Commons **(B / CD-ROM)**, *436 Pages*	Gross	400.00
8173660905	Code Hacking: A Developer's Guide to Network Security **B / CD-ROM)**, *404 Pages*	Conway	400.00
8173665346	**Linux Network Security (B / CD-ROM), *564 Pages***	**Smith**	**475.00**
8173660913	The Linux TCP/IP Stack: Networking for Embedded Systems **(B / CD-ROM)**, *604 Pages*	Herbert	450.00
8173669759	Software Testing Techniques: Finding the Defects that Matter, *388 Pages*	Loveland	350.00

FORTHCOMING TITLES

August 2005

8173661049	Creating Fractals **(B / CD-ROM)**, *300 Pages*	Stevens	350.00
8173661472	Crystal Report 11 for Developer's **(B / CD-ROM)**, *600 Pages*	McAmis	450.00
8173663963	Mathematics and Physics for Programmers **(B / CD-ROM)**, *608 Pgs*	Kodicek	450.00
8173662533	Microprocessors: From Assembly Language to C Using the PIC18Fxx2 **(B / CD-ROM)**, *450 Pages*	Reese	300.00
8173664579	The OpenGL Extensions Guide **(B / CD-ROM)**, *670 Pages*	Lengyel	525.00

September 2005

8173670056	C++ Standard Library Practical Tips **(B / CD-ROM)**, *300 Pages*	Reese	200.00
8173670102	Cryptographic Libraries for Developer's **(B / CD-ROM)**, *400 Pages*	Kelly	400.00
8173670099	Java Messaging, *500 Pages*	Bruno	400.00
8173670072	Enterprise Web Services Security **(B / CD-ROM)**, *400 Pages*	Hollar	400.00
8173670110	Programming Mobile Business Solutions Using Visual Studio 2005 **(B / CD-ROM)**, *400 Pages*	Robbins	400.00
8173670080	SQL Server 2005 for Developer's **(B / CD-ROM)**, *400 Pages*	Ericsson	400.00

CRC Press

0849308097	**Software Testing: A Craftman's Approach, 2/ed, *384 Pages***	**Jorgensen**	**£ 46.00**
0849323665	**Software Engineering Processess: Principles & Application, *752 Pages***	**Wang**	**£ 60.00**

Shroff / Curlingstone

(Curlingstone now an imprint of Apress)

8173666326	The Art and Science of Oracle Performance Tuning, *484 Pages*	Lawson	325.00
8173666989	Data Modeling for Everyone, *524 Pages*	Allen	400.00

Shroff / DVT Press

8173669171	The J2EE Architect's Handbook, *304 Pages*	Ashmore	300.00

Shroff / Endorf

8173667004	CISSP Study Guide 2/ed (2002-03 Updated ed), *568 Pages*	Endorf	300.00
8173667012	SSCP Study Guide Updated Edition, *292 Pages*	Endorf	225.00

Shroff / Friends of ED

(friends of ED now an imprint of Apress)

TITLES AT REDUCED PRICES

8173664196	Director 8.5 Studio **(B / CD-ROM)**, *836 Pages*	Bauman	100.00
817366336X	Flash 5 Studio **(B / CD-ROM)**, *798 Pages*	Adnani	100.00
8173663939	Flash & Director StudioLab, *516 Pages*	Gould	75.00
8173663416	Flash XML StudioLab, *512 Pages*	Tindale	75.00
8173663408	Foundation Director 8.5, *456 Pages*	Utian	75.00
8173663386	Foundation Dreamweaver UltraDev, *480 Pages*	Paddock	50.00
8173662126	Foundation Flash 5, *646 Pages*	Bhangal	75.00
8173663459	Foundation Illustrator 10, *548 Pages*	Loader	75.00
8173664552	Foundation Macromedia Flash MX, *704 Pages*	Besley	75.00
8173664188	Foundation Photoshop 6 (Includes 8 Full Color Pages), *672 Pages*	Smith	75.00
8173664595	Photoshop Elements Express, *250 Pages*	Beckley	50.00
8173664609	Revolutionary After Effects 5.5 **(B / CD-ROM)**, *500 Pages*	Kingsnorth	50.00

PUBLISHED TITLES

8181281594	Extending Macromedia Flash MX 2004: Complete Guide and Reference to JavaScript Flash, *472 Pages*	Peters	425.00
8173662185	Foundation ActionScript, *512 Pages*	Bhangal	300.00
8181281624	Foundation Macromedia Flash MX 2004, *488 Pages*	Besley	425.00
1903450039	New Masters of Flash **(B / CD-ROM)**, *544 Pages* [Size 9x10]	McIntyre	850.00
1903450365	New Masters of Flash: Annual 2002 **(B / CD-ROM)**, *544 Pages* [Size 9x10]	McIntyre	850.00
1903450624	New Masters of Photoshop **(B / CD-ROM)**, *552 Pages* [Size 9x10]	McIntyre	850.00

Shroff / John Wiley

0471469122	Art of Software Testing, 2/ed, *252 Pages*	Meyers	$ 75.00
0471718890	**The Black Book of Outsourcing: How to Manage the Changes, Challenges, and Opportunites, *390 Pages***	**Brown**	**$ 29.95**
8173660387	Client/Server Programming with Java & Corba 2/ed **(B / CD-ROM)**, *1,072 Pages*	Orfali	650.00
9971513714	Client/Server Survival Guide 3/ed, *795 Pages*	Orfali	500.00
9971514036	The CISSP Prep Guide: Mastering the 10 Domain of Computer Security, *576 Pages*	Krutz	200.00
997151415X	The Data Warehouse Lifecycle Toolkit **(B / CD-ROM)**, *792 Pages*	Kimball	650.00
9814126144	The Data Warehouse Toolkit 2/ed, *460 Pages*	Kimball	450.00
0471267686	Financial Modeling Using Excel & VBA **(B / CD-ROM)**, *672 Pages*	Sengupta	$ 65.00
9971513579	Software Engineering: Principles & Practice 2/ed, *748 Pages*	Van Vliet	500.00

9971514281	System Analysis & Design: An Object Oriented Approach with UML, *534 Pages*	Dennis	450.00

Shroff / Labyrinth

8173665443	Computer Concepts & Windows, *130 Pages*	Stolins	125.00
817366546X	Microsoft Excel 2002 - Level 1 **(B / DISK)**, *252 Pages*	Favro	250.00
8173665478	Microsoft Office XP Comprehensive Course **(B / DISK)**, *1,202 Pgs*	Favro/Stolins	675.00
8173665451	Microsoft Word 2002 - Level 1 **(B / DISK)**, *250 Pages*	Favro/Stolins	250.00

Shroff / Manning

8173665389	J2EE & XML Development, *300 Pages*	Gabrick	200.00
8173665397	JDK 1.4 Tutorial, *408 Pages*	Travis	300.00
8173665400	Microsoft .NET for Programmers, *380 Pages*	Grimes	275.00
8173665419	Web Development with Apache & Perl, *432 Pages*	Petersen	300.00
8173665427	Web Development with JavaServer Pages, 2/ed, *804 Pages*	Fields	450.00

Shroff / Many Worlds Productions

0970683081	Model, Rig, Animate! with 3ds max 6 **(B / CD-ROM)**, *240 Pages*	Bousquet	$ 35.00

Shroff / Mike Murach

8173669228	Murach's ASP .NET Web Programming with VB .NET, *736 Pages*	Lowe	575.00
8173665532	**Murach's Beginning Java 2 JDK 5, *800 Pages***	**Doug**	**450.00**
8173669244	Murach's C#, *770 Pages*	Murach	550.00
8173669236	Murach's JavaServlets & JSP: Training & Reference **(B / CD-ROM)**, *642 Pages*	Steelman	525.00
817366921X	Murach's VB .NET Database Programming with ADO. NET, *610 Pages*	Prince	525.00

FORTHCOMING TITLES

8173668620	Murach's ASP.NET 2.0 Upgrader's Guide C# Edition, *546 Pages*	Lowe	400.00
8173668698	Murach's ASP.NET 2.0 Upgrader's Guide VB Edition, *546 Pages*	Lowe	400.00

US EDITIONS

1890774022	DB2 for the COBOL Programmer, Part 1 (2/ed), *431 Pages*	Garvin	$ 45.00
1890774030	DB2 for the COBOL Programmer, Part 2 (2/ed), *402 Pages*	Garvin	$ 45.00
0911625313	DOS/VSE Assembler Language, *492 Pages*	McQuilen	$ 45.00
0911625364	DOS/VSE ICCF, *372 Pages*	Eckols	$ 31.00
091162550X	DOS/VSE JCL, 2/ed, *445 Pages*	Eckols	$ 45.00
0911625291	IMS for the COBOL Programmer, Part 1, *333 Pages*	Eckols	$ 45.00
0911625305	IMS for the COBOL Programmer, Part 2, *398 Pages*	Eckols	$ 45.00
1890774170	Murach's CICS Desk Reference, *592 Pages*	Lowe	$ 49.50
189077409X	Murach's CICS for the COBOL Programmer, *633 Pages*	Menendez	$ 54.00
1890774146	Murach's OS/390 and z/OS JCL, *559 Pages*	Lowe	$ 62.50
1890774243	Murach's Mainframe COBOL, *687 Pages*	Murach	$ 59.50
1890774057	Murach's Structured COBOL **(B / CD-ROM)**, *800 Pages*	Murach	$ 62.50
0911625569	MVS TSO, Part 1: Concepts and ISPF, *467 Pages*	Lowe	$ 42.50
0911625577	MVS TSO, Part 2: Commands, CLIST, and REXX, *450 Pages*	Lowe	$ 42.50
0911625348	MVS Assembler Language, *528 Pages*	McQuillen	$ 45.00
0911625941	Structured COBOL Methods, *208 Pages*	Noll	$ 25.00
091162547X	VM / CMS: XEDIT Commands and Features, *225 Pages*	Eckols	$ 25.00
091162533X	VSAM: AMS and Application Programming, *260 Pages*	Lowe	$ 32.50
0911625463	VSAM for the COBOL Programmer, *187 Pages*	Lowe	$ 27.50

Shroff / No Starch Press

8173668078	The Art of Assembly Language **(B / CD-ROM)**, *928 Pages*	Hyde	525.00
8173667365	The Book of Nero 6 Ultra Edition: CD and DVD Burning Made Easy, *216 Pages*	Ross	200.00

8173663130	**The Book of Postfix: State-of-the-Art Message Transport , *362 Pages***	**Hildebrandt**	**425.00**
8173667993	The Book of Wi-Fi, *288 Pages*	Ross	250.00
8173668566	The Book of VMWare, *268 Pages*	Ward	250.00
817366952X	Cisco Routers for the Desperate: Router Management, The Easy Way, *148 Pages*	Lucas	150.00
8173668388	Hacking: The Art of Exploitation, *256 Pages*	Erickson	225.00
8173668337	**How Linux Works, *392 Pages***	**Ward**	**300.00**
8173667977	How Not To Program in C++, *272 Pages*	Ouanlline	175.00
8173668345	The Linux Cookbook, 2/ed, *826 Pages*	Stutz	600.00
8173668353	**The Linux Enterprise Cluster: Build a Highly Available Cluster with Commodity Hardware and Free Software (B / CD-ROM), *460 Pages***	**Kopper**	**450.00**
8173668825	Linux for Non-Geeks **(B / CD-ROM)**, *336 Pages*	Grant	300.00
8173668876	The Official GNOME 2 Developer's Guide, *520 Pages*	Warkus	475.00
8173667985	Steal This Computer Book 3, *356 Pages*	Wang	350.00
8173669554	Stealing this File Sharing Book, *296 Pages*	Wang	300.00
8173668795	**Wicked Cool Shell Scripts, *368 Pages***	**Taylor**	**350.00**
8173668329	Write Great Code: Volume 1: Understanding the Machine, *464 Pgs*	Hyde	400.00
8173661871	**Writing Portable Code: A Guide to Developing Software for Multiple Platforms, *276 Pages***	**Hook**	**275.00**

FORTHCOMING TITLES

September 2005

8173670021	The Book of™ MaxDB: MySQL for the Enterprise **(B / CD-ROM)**, *600 Pages*	Singleton	500.00
8173669392	Computer Security 101: What You Need to Know to Secure Your Computer or Network **(B / CD-ROM)**, *250 Pages*	Bradley	250.00
8173670013	Linux Made Easy: The Official Guide to Xandros 3 for Everyday Users **(B / CD-ROM)**, *496 Pages*	Grant	400.00
8173670129	Now Playing: Visual Basic 2005 Express **(B / CD-ROM)**, *760 Pages*	Wang	500.00
8173664641	The TCP/IP Guide, *1,550 Pages*	Kozierok	850.00
8173670137	Wicked Cool Java: Code Bits, Open-Source Libraries and Project Ideas, *772 Pages*	Eubanks	550.00
8173670048	Wicked Cool™ Perl Scripts Useful Perl Scripts That Solve Difficult Problems, *304 Pages*	Oualline	300.00
8173670145	Write Great Code: Volume 2: Thinking Low - Level, Writing High - Level, *272 Pages*	Hyde	275.00

Shroff / O'Reilly

TITLES AT REDUCED PRICES

8173663661	.NET Framework Essentials, *324 Pages*	Thai	75.00
8173664277	Access Cookbook for 97, 2000 & 2002 **(B / CD-ROM)**, *724 Pages*	Getz	100.00
8173662231	ActionScriprt: The Definitive Guide, *726 Pages*	Moock	150.00
8173662886	Building Oracle XML Applications **(B / CD-ROM)**, *824 Pages*	Muench	100.00
8173664250	Building Wireless Community Networks, *144 Pages*	Flickenger	50.00
8173662584	Cascading Style Sheets: The Definitive Guide, *476 Pages*	Meyer	75.00
8173662878	Developing ASP Components 2/ed, *864 Pages*	Powers	100.00
817366272X	DHCP for Windows 2000, *288 Pages*	Alcott	75.00
8173663599	Exim: The Mail Transport Agent, *638 Pages*	Hazel	75.00
8173660573	Java Threads, 2/ed, *336 Pages*	Oaks	100.00
8173664145	Java Cookbook, *890 Pages*	Darwin	150.00
8173660557	Java Foundation Classes in a Nutshell, *752 Pages*	Flanagan	75.00
8173662711	Java Internationalization, *456 Pages*	Deitsch	75.00
8173663807	Java Programming with Oracle SQLJ, *404 Pages*	Price	75.00
8173661103	JavaScript Application Cookbook, *512 Pages*	Bradenbaugh	75.00
8173661111	JavaScript Pocket Reference, *96 Pages*	Flanagan	25.00
8173662509	Jini in a Nutshell, *420 Pages*	Oaks	75.00
8173662320	Learning Java **(B / CD-ROM)**, *732 Pages*	Niemeyer	75.00

8173660603	Learning Perl/Tk, *380 Pages*	Walsh	75.00
8173661677	Learning Web Design: A Beginner's Guide to HTML, Graphics & Web Environment (2 Color Book), *432 Pages*	Niederst	100.00
8173663173	Learning WML & WMLScript, *204 Pages*	Frost	50.00
8173663149	Learning XML, *376 Pages*	Ray	75.00
8173662193	MCSD in a Nutshell: The Visual Basic Exams, *636 Pages*	Foxall	75.00
8173662398	MCSE in a Nutshell: The Windows 2000 Exams, *504 Pages*	Moncur	75.00
8173662517	MP3: The Definitive Guide, *408 Pages*	Hacker	75.00
8173660697	Oracle PL/SQL Language Pocket Reference, *80 Pages*	Feuerstein	25.00
8173661243	Perl/Tk Pocket Reference, *104 Pages*	Lidie	25.00
817366269X	PHP Pocket Reference, *124 Pages*	Lerdorf	25.00
8173661251	PNG: The Definitive Guide, *344 Pages*	Roelofs	50.00
817366126X	Practical Internet Groupware, *520 Pages*	Udell	75.00
8173660751	Practical UNIX & Internet Security, 2/ed, *1,008 Pages*	Garfinkel	150.00
8173660190	QuarkXPress in a Nutshell, *552 Pages*	O'Quinn	50.00
8173662487	sed & awk Pocket Reference, *60 Pages*	Robbins	25.00
8173661294	sendmail Desktop Reference, *74 Pages*	Costales	25.00
8173662606	Tcl/Tk Pocket Reference, *100 Pages*	Raines	25.00
8173663734	VB .NET Language in a Nutshell, *662 Pages*	Roman	100.00
8173663475	Web Database Applications with PHP & MySQL, *600 Pages*	Williams	75.00
8173661359	Webmaster in a Nutshell, 2/ed, *540 Pages*	Spainhour	75.00
8173662630	Windows 2000 Active Directory, *624 Pages*	Lowe-Norris	75.00
8173663327	Windows 2000 Performance Guide, *650 Pages*	Friedman	75.00
8173663289	XML in a Nutshell, *400 Pages*	Harold	100.00
8173662096	XSLT, *486 Pages*	Tidwell	100.00
PUBLISHED TITLES			
8173666539	.NET and XML, *484 Pages*	Bornstein	425.00
8173669201	.NET Compact Framework Pocket Guide, *122 Pages*	Lee	125.00
8173666547	.NET Framework Essentials, 3/ed, *392 Pages*	Thai	325.00
8173665729	**.NET Gotchas, *402 Pages***	**Subramaniam**	**300.00**
8173665753	802.11 Security, *200 Pages*	Potter	175.00
8173664420	802.11 Wireless Networks: The Definitive Guide, *474 Pages*	Gast	400.00
8173664285	Access Database Design & Programming, 3/ed, *454 Pages*	Roman	325.00
8173666377	**Access Hacks: Tips & Tools for Wrangling Your Data, *362 Pgs***	**Bluttman**	**325.00**
8173667233	ActionScript Cookbook, *904 Pages*	Lott	675.00
8173666555	ActionScript for Flash MX Pocket Reference, *152 Pages*	Moock	125.00
8173665761	ActionScript for Flash MX: The Definitive Guide, 2/ed, *904 Pages*	Colin Moock	750.00
8173666563	Active Directory for Windows Server 2003, 2/ed, *692 Pages*	Allen	575.00
8173667357	Active Directory Cookbook for Windows Server 2003 and Windows 2000, *632 Pages*	Allen	525.00
817366725X	ADO .NET Cookbook, *636 Pages*	Hamilton	500.00
8173665222	ADO .NET for Visual Basic .NET 2003 in a Nutshell: A Desktop Quick Reference **(B / CD-ROM)**, *626 Pages*	Hamilton	500.00
0596006004	Adobe Encore DVD: In the Studio, *336 Pages*	Dixon	1,225.00
0596007361	Adobe InDesign CS One-on-One **(B / CD-ROM)**, *400 Pages*	McClelland	1,375.00
0596006187	Adobe Photoshop CS One-on-One **(B / CD-ROM)**, *488 Pages*	McClelland	1,225.00
8173668302	**Advanced Perl Programming, 2/ed, *308 Pages***	**Cozens**	**325.00**
8173668205	AI for Game Developers, *400 Pages*	Bourg	375.00
8173669384	**AspectJ Cookbook, *364 Pages***	**Miles**	**350.00**
8173667292	Amazon Hacks: 100 Industrial Strength Tips & Tricks, *312 Pages*	Bausch	250.00
8173665524	Ant: The Definitive Guide, *296 Pages*	Tilly	275.00
8173665133	Apache: The Definitive Guide (Covers Apache 2.0 & 1.3) 3/ed, *594 Pages*	Laurie	500.00

8173667446	Apache Cookbook (Covers Apache 2.0 & 1.3), *264 Pages*	Coar	250.00
8173662525	Apache Pocket Reference, *112 Pages*	Ford	75.00
8173662274	**Apache Security, *428 Pages***	**Ristic**	**400.00**
8173666571	ASP .NET Cookbook (Cover ASP .NET 1.1), *846 Pages*	Kittel	700.00
8173667306	ASP .NET in a Nutshell: A Desktop Quick Reference, 2/ed, *1,008 Pages*	Duthie	600.00
8173667837	**The Art of Project Management, *512 Pages***	**Berkun**	**350.00**
8173662347	AutoCAD 2000 In a Nutshell: A Command Reference Guide, *592 Pages*	Kent	325.00
8173663955	Beginning Perl for Bioinformatics, *390 Pages*	Tisdall	300.00
8173668981	Better, Faster, Lighter Java, *270 Pages*	Tate	250.00
817366658X	BGP, *292 Pages*	van Beijnum	275.00
8173665125	BLAST, *372 Pages*	Korf	325.00
817366899X	BSD Hacks:100 Industrial-Strength Tips & Tools, *458 Pages*	Lavigne	375.00
8173666598	Building Embedded Linux Systems, *422 Pages*	Yaghmour	350.00
8173661014	Building Internet Firewalls 2/ed, *904 Pages*	Chapman	650.00
8173662282	Building Java Enterprise Applications: Vol. 1 - Architecture, *324 Pages*	McLaughlin	225.00
8173661391	Building Linux Clusters **(B / CD-ROM)**, *360 Pages*	Spector	350.00
8173665621	Building Secure Servers with Linux, *454 Pages*	Bauer	375.00
8173664749	**Building the Perfect PC, *360 Pages***	**Thompson**	**350.00**
8173666601	**C Pocket Reference, *142 Pages***	**Prinz**	**100.00**
8173664439	C# & VB .NET Conversion Pocket Reference, *156 Pages*	Mojica	75.00
8173665885	C# Cookbook, *876 Pages*	Teilhet	600.00
8173664293	C# Essentials, 2/ed, *224 Pages*	Albahari	175.00
8173665192	C# Language Pocket Reference, *132 Pages*	Drayton	100.00
8173663726	C# in a Nutshell: A Desktop Quick Reference, *864 Pages*	Drayton	600.00
8173666628	C++ in a Nutshell, (Cover ISO/IEC 14882 STD) *816 Pages*	Lischner	500.00
8173667101	C++ Pocket Reference, *148 Pages*	Loudon	100.00
0596001088	The Cathedral & The Bazaar: Musings On Linux and Open Source by an Accidental Revolutionary, Revised & Expanded, *256 Pages*	Raymond	525.00
8173669066	Cascading Style Sheets: The Definitive Guide (Covers CSS2 & CSS 2.1), 2/ed, *538 Pages*	Meyer	475.00
8173662266	CDO & MAPI Programming with Visual Basic, *388 Pages*	Grundgeiger	175.00
8173664269	CGI Programming on the World Wide Web, *454 Pages*	Gundavaram	300.00
817366045X	CGI Programming with Perl 2/ed, *476 Pages*	Gundavaram	325.00
8173668469	**Classic Shell Scripting, *568 Pages***	**Robbins**	**425.00**
8173667241	Cisco Cookbook, *918 Pages*	Dooley	700.00
8173663653	COM & .NET Component Services, *390 Pages*	Lowy	250.00
8173663645	COM+ Programming with Visual Basic, *372 Pages*	Mojica	225.00
8173660638	Creating Effective JavaHelp, *196 Pages*	Lewis	125.00
8173669163	CSS Cookbook (Cover CSS 2.1), *280 Pages*	Schmitt	275.00
8173667802	**Database In Depth: The Relational Model for Practitioners, *250 Pgs***	**C.J.Date**	**225.00**
8173662363	Database Nation: The Death of Privacy in the 21st Century, *336 Pages*	Garfinkel	235.00
8173662894	Database Programming with JDBC & Java 2/ed, *348 Pages*	Reese	150.00
8173665664	Designing Embedded Hardware, *324 Pages*	Catsoulis	250.00
8173663882	Designing Large Scale LANs, *408 Pages*	Dooley	275.00
8173662428	Developing Bio-informatics Computer Skills, *504 Pages*	Gibas	225.00
8173669473	**Developing Feeds with RSS and Atom, *280 Pages***	**Hammersley**	**300.00**
0596005474	Digital Photography: Expert Techniques (Covers Photoshop CS), *496 Pages*	Milburn	1,375.00
0596006667	Digital Photography Hacks: 100 Industrial - Strength Tips & Tools, *336 Pages*	Story	800.00
0596006276	Digital Photography Pocket Guide, 2/ed, *128 Pages*	Story	500.00
0596009461	**Digital Video Hacks: Tips & Tools for Shooting, Editing & Sharing, *426 Pages***	**Paul**	**800.00**
0596005237	Digital Video Pocket Guide, *474 Pages*	Story	500.00

8173662983	DNS & BIND (Covers BIND 9), 4/ed, *642 Pages*	Albitz	500.00
8173665672	DNS & BIND Cookbook, *248 Pages*	Liu	250.00
8173663785	DNS on Windows 2000, 2/ed, *376 Pages*	Larson	275.00
8173660506	DNS on Windows NT, *352 Pages*	Albitz	195.00
8173662991	Dreamweaver MX: The Missing Manual, *750 Pages*	McFarland	600.00
8173665281	Dynamic HTML: The Definitive Reference, 2/ed, *1,428 Pages*	Goodman	650.00
8173669015	Eclipse (Coverage of 3.0), *344 Pages*	Holzner	325.00
8173669309	Eclipse Cookbook (Cover 3.0), *372 Pages*	Holzner	350.00
8173663017	Effective awk Programming, 3/ed, *454 Pages*	Robbins	325.00
8173667268	Enterprise JavaBeans (Covers EJB 2.1 & EJB 2.0 Includes workbook for JBoss 4.0), 4/ed, *798 Pages*	Monson - Haefel	500.00
8173668167	Enterprise Service Architecture - O'Reilly SAP Series, *236 Pages*	Woods	225.00
8173669317	Essential ActionScript 2.0, *528 Pages*	Moock	400.00
8173666784	**Enterprise Service Bus, *284 Pages***	**Chappell**	**300.00**
8173669430	**Essentail SharePoint, *338 Pages***	**Webb**	**325.00**
8173663580	Essential SNMP, *338 Pages*	Mauro	300.00
817366529X	Essential System Administration, 3/ed, *1,178 Pages*	Frisch	525.00
8173666644	Essential System Administration Pocket Reference, *152 Pages*	Frisch	125.00
8173660255	Essential Windows NT System Administration, *488 Pages*	Frisch	225.00
8173662495	Ethernet: The Definitive Guide, *528 Pages*	Spurgeon	300.00
8173662754	Excel 2000 In a Nutshell: A Power User's Quick Reference, *560 Pages*	Simon	300.00
8173669619	**Excel 2003 Personal Trainer (B / CD-ROM), (2 Color book) *490 Pages***	**CustomGuide**	**550.00**
8173668299	Excel 2003 Programming: A Developer's Notebook, *330 Pages*	Webb	325.00
8173668035	**Excel: The Missing Manual, *802 Pages***	**MacDonald**	**550.00**
8173669406	**Excel Annoyances: How to Fix the Most Annoying Things about Your Favorite Spreadsheet , *266 Pages***	**Frye**	**275.00**
8173668612	**Excel Hacks: 100 Industrial Strength Tips & Tools, *316 Pages***	**David**	**275.00**
8173665257	Exploring the JDS Linux Desktop **(B / CD-ROM)**, *418 Pages*	Adelstein	400.00
8173667470	**Flash Hacks: 100 Industrial Strength Tips & Tools, *504 Pages***	**Bhangal**	**400.00**
8173667314	Flash Remoting MX: The Definitive Guide, *652 Pages*	Muck	550.00
8173668590	**Flash Out of the Box: A User-Centric Beginner's Guide to Flash (B / CD-ROM), *264 Pages***	**Hoekman**	**300.00**
0596002874	Free As In Freedom: Richard Stallman's Crusade for Free Software, *243 Pages*	Williams	750.00
817366868X	**GDB Pocket Reference, *78 Pages***	**Robbins**	**75.00**
817366949X	**Google Hacks: Tips & Tools for Smarter Searching 2/ed, *496 Pages***	**Calishain**	**350.00**
8173667136	Google Pocket Guide, *144 Pages*	Calishain	125.00
8173669325	Google: The Missing Manual, *312 Pages*	Milstein	325.00
0596006624	Hackers & Painters: Big Ideas from the Computer Age, *272 Pages*	Graham	700.00
8173668256	Hardcore Java, *354 Pages*	Simmons	300.00
8173663424	Hardening Cisco Routers, *196 Pages*	Akin	150.00
8173668213	Hardware Hacking Projects for Geeks, *358 Pages*	Fullam	350.00
8173664668	**Head First Design Patterns, *688 Pages***	**Sierra**	**500.00**
8173665265	Head First EJB: Passing the Sun Certified Business Component Developer Exam, *744 Pages*	Sierra	450.00
8173666024	**Head First Java (Covers Java 5.0) 2/ed, *730 Pages***	**Sierra**	**450.00**
817366403X	**Head First Servlets & JSP: Passing the Sun Certified Web Component Developer Exam, *666 Pages***	**Sierra**	**600.00**
8173669341	Hibernate: A Developer's Notebook, *190 Pages*	Elliott	200.00
8173669260	High Performance Linux Cluster with OSCAR, Rocks, OpenMosix, and MPI, *380 Pages*	Sloan	350.00
8173669023	High Performance MySQL, *304 Pages*	Zawodny	300.00
8173665141	HTML & XHTML: The Definitive Guide, 5/ed, *676 Pages*	Musciano	500.00

8173664323	HTML Pocket Reference, 2/ed, *100 Pages*	Niederst	70.00
8173663165	**Home Hacking Projects for Geeks, *346 Pages***	**Faulkner**	**325.00**
8173669449	**Home Networking Annoyances: How to Fix the Most Annoying Things About Your Home Network, *234 Pages***	**Ivens**	**250.00**
0596008597	Illustrations with Photoshop: A Designer's Notebook, *96 Pages*	Rodarmor	775.00
8173665109	Information Architecture for the World Wide Web, 2/ed, *492 Pages*	Rosenfeld	400.00
8173669414	**Internet Annoyances: How to Fix the Most Annoying Things about Going Online, *266 Pages***	**Gralla**	**250.00**
8173661057	Internet Core Protocols: The Definitive Guide **(B / CD-ROM)**, *476 Pages*	Hall	375.00
8173660158	Internet in a Nutshell, *456 Pages*	Quercia	215.00
8173663378	IP Routing, *244 Pages*	Malhotra	200.00
817366448X	IPv6 Essentials, *362 Pages*	Hagen	300.00
8173663025	**IPv6 Network Administration, *316 Pages***	**Murphy**	**325.00**
8173667489	IRC Hack: 100 Industrial-Strength Tips & Tools, *442 Pages*	Mutton	400.00
8173667373	J2EE Design Patterns, *390 Pages*	Crawford	325.00
8173663432	J2ME in a Nutshell: A Desktop Quick Reference, *474 Pages*	Topley	400.00
8173669295	Jakarta Commons Cookbook, *412 Pages*	O'Brien	375.00
8173669481	**Jakarta Struts Cookbook, *536 Pages***	**Siggelkow**	**400.00**
8173667144	Jakarta Struts Pocket Reference, *142 Pages*	Cavaness	125.00
8173668477	**Java 1.5 Tiger: A Developer's Notebook, *210 Pages***	**McLaughlin**	**175.00**
8173664471	Java & SOAP, *286 Pages*	Englander	275.00
817366367X	Java & XML, 2/ed, *534 Pages*	McLaughlin	450.00
8173664498	Java & XML Data Binding, *224 Pages*	McLaughlin	225.00
8173663793	Java & XSLT, *534 Pages*	Burke	350.00
8173669368	Java Cookbook (Coverage of 1.5), 2/ed, *872 Pages*	Darwin	600.00
8173666660	Java Database Best Practices, *304 Pages*	Eckstein	275.00
8173666679	Java Data Objects, *568 Pages*	Jordan	350.00
817366577X	Java Enterprise Best Practices, *296 Pages*	Eckstein	275.00
8173664625	Java Enterprise in a Nutshell: A Desktop Quick Reference, 2/ed, *1,004 Pages*	Farley	600.00
8173662843	Java Examples in a Nutshell 2/ed, *592 Pages*	Flanagan	225.00
8173668639	Java Examples in a Nutshell: A Tutorial Companion to Java in a Nutshell, 3/ed, *728 Pages*	Flanagan	400.00
8173666687	Java Extreme Programming Cookbook, *296 Pages*	Burke	225.00
817366434X	Java in a Nutshell: A Desktop Quick Reference, 4/ed, *1,000 Pages*	Flanagan	600.00
8173664404	Java Management Extensions, *318 Pages*	Perry	275.00
8173663211	Java Message Service, *300 Pages*	Monson-Haefel	225.00
817366353X	Java Network Programming, 3/ed, *770 Pages*	Harold	500.00
8173665117	Java NIO, *308 Pages*	Hitchens	275.00
8173665788	Java Performance Tuning, 2/ed, *600 Pages*	Shirazi	450.00
8173663904	Java Programming with Oracle JDBC, *504 Pages*	Bales	300.00
8173663815	Java RMI, *578 Pages*	Grosso	400.00
8173664129	Java Security, 2/ed, *624 Pages*	Oaks	500.00
8173668221	Java Servlet & JSP Cookbook, *756 Pages*	Perry	500.00
8173662851	Java Servlet Programming 2/ed, *786 Pages*	Hunter	500.00
8173665680	Java Swing, 2/ed, *1,288 Pages*	Loy	750.00
8173665923	Java Threads (Covers J2SE 5.0), 3/ed, *368 Pages*	Oaks	300.00
8173663440	Java Web Services, *286 Pages*	Chappell	250.00
8173666695	Java Web Services in a Nutshell: A Desktop Quick Reference (Covers J2EE 1.4 & JWSDP), *696 Pages*	Topley	500.00
8173669465	**JBoss: A Developer's Notebook, *182 Pages***	**Richards**	**200.00**
8173666709	JavaScript and DHTML Cookbook, *546 Pages*	Goodman	475.00

8173663823	JavaScript: The Definitive Guide (Covers JavaScript 1.5), 4/ed, *942 Pages*	Flanagan	750.00
8173669031	**JavaServer Faces, *614 Pages***	**Bergsten**	**450.00**
8173665303	JavaServer Pages (Covers JSP 2.0 & JSTL 1.1), 3/ed, *762 Pages*	Bergsten	500.00
8173663831	JavaServer Pages Pocket Reference, *96 Pages*	Bergsten	65.00
8173666717	**JDBC Pocket Reference, *160 Pages***	**Bales**	**100.00**
8173668604	JUnit Pocket Guide, *100 Pages*	Beck	100.00
059600768X	Just A Geek, *296 Pages*	Wheaton	700.00
817366515X	JXTA in a Nutshell: A Desktop Quick Reference, *422 Pages*	Oaks	225.00
8173665605	Kerberos: The Definitive Guide, *280 Pages*	Garman	275.00
8173669724	**Killer Game Programming in Java, *986 Pages***	**Davison**	**675.00**
8173666725	LDAP System Administration, *318 Pages*	Carter	300.00
8173665168	Learning C#, *374 Pages*	Liberty	325.00
8173669635	**Learning GNU Emacs, 3/ed, *544 Pages***	**Cameron**	**450.00**
8173663912	Learning Oracle PL/SQL (Covers Oracle9i), *452 Pages*	Pribyl	325.00
8173663718	Learning Perl, 3/ed, *344 Pages*	Schwartz	275.00
8173667322	Learning PHP 5, *378 Pages*	Sklar	350.00
8173667381	Learning Python (Covers Python 2.3), 2/ed, *630 Pages*	Lutz	525.00
8173668051	**Learning the Bash Shell, 3/ed, *362 Pages***	**Newham**	**350.00**
8173664447	Learning the Korn Shell, 2/ed, *438 Pages*	Rosenblett	325.00
8173664234	Learning the UNIX Operating System, 5/ed, *174 Pages*	Peek	125.00
8173660611	Learning the vi Editor 6/ed, *352 Pages*	Lamb	250.00
8173666741	Learning UML, *304 Pages*	Si Alhir	150.00
817366563X	Learning Visual Basic .NET, *326 Pages*	Liberty	250.00
8173669422	**Learning Windows Server 2003, *682 Pages***	**Hassell**	**525.00**
817366062X	lex & yacc 2/ed, *392 Pages*	Levine	225.00
8173668442	**Linux Cookbook, *590 Pages***	**Schroder**	**450.00**
8173668493	**Linux Device Drivers (Covers Version 2.6.10 Linux Kernel), 3/ed, *646 Pages***	**Rubini**	**450.00**
8173667179	Linux in a Nutshell, 4/ed, *994 Pages*	Siever	500.00
8173669457	**Linux in a Windows World, *504 Pages***	**Smith**	**450.00**
8173668507	Linux iptables Pocket Reference, *106 Pages*	Purdy	100.00
8173664544	**Linux Network Administrator's Guide, 3/ed, *372 Pages***	**Kirch**	**350.00**
8173668647	**Linux Pocket Guide, *212 Pages***	**Barrett**	**150.00**
8173667187	Linux Security Cookbook, *340 Pages*	Barrett	325.00
817366675X	Linux Server Hacks: 100 Industrial - Strength Tips & Tools, *240 Pages*	Flickenger	225.00
8173668434	Linux Unwired, *322 Pages*	Weeks	300.00
817366465X	Managing & Using MySQL, 2/ed, *448 Pages*	Reese	325.00
8173662746	Managing IMAP, *412 Pages*	Mullet	200.00
8173660271	Managing IP Networks with Cisco Routers, *352 Pages*	Ballew	200.00
8173663610	Managing NFS & NIS, 2/ed, *518 Pages*	Stern	400.00
8173669589	Managing Projects with GNU Make, 3/ed, *310 Pages*	Mecklenburg	300.00
8173665230	Managing RAID on Linux, *268 Pages*	Vadala	275.00
8173668655	Managing Security with Snort & IDS Tools, *296 Pages*	Cox Ph.D.	300.00
8173662800	Managing the Windows 2000 Registry, *564 Pages*	Robicheaux	325.00
0596007035	**Mapping Hacks: Tips & Tools for Electronic Cartography, *574 Pages***	**Erie**	**900.00**
8173661162	Mastering Algorithms with C **(B / DISK)**, *572 Pages*	Loudon	400.00
8173664455	**Mastering FreeBSD and OpenBSD Security, *472 Pages***	**Korff**	**425.00**
8173664366	Mastering Oracle SQL, (Cover Oracle9i), *348 Pages*	Mishra	225.00
8173664617	**Mastering Oracle SQL (Covers Oracle Database10g), 2/ed, *504 Pages***	**Mishra**	**400.00**
8173666768	Mastering Perl for Bioinformatics, *406 Pages*	Tisdall	300.00
8173665087	Mastering Regular Expressions, 2/ed, *492 Pages*	Friedl	400.00
8173665702	Mastering Visual Studio .NET 2003, *420 Pages*	Flanders	375.00

8173667918	**Maven: A Developer's Notebook, *232 Pages***	**Massol**	**250.00**
8173667497	Mono: A Developer's Notebook, *312 Pages*	Dumbill	325.00
8173665648	MySQL Cookbook (Covers MySQL 4.0), *1,028 Pages*	DuBois	750.00
817366806X	MySQL in a Nutshell, *358 Pages*	Dyer	325.00
8173666776	**MySQL Pocket Reference, *94 Pages***	**Reese**	**100.00**
8173665249	NetBeans: The Definitive Guide, *662 Pages*	Boudreau	500.00
8173668809	Network Security Assessment, *398 Pages*	McNab	400.00
8173668396	**Network Security Tools: Writing, Hacking, and Modifying Security Tools , *350 Pages***	**Dhanjani**	**350.00**
8173663521	Networking Security Hacks: 100 Industrial-Strength Tips & Tools, *328 Pages*	Lockhart	325.00
8173663688	Network Troubleshooting Tools, *370 Pages*	Sloan	250.00
8173667926	**Nokia Smartphone Hacks, *418 Pages***	**Yuan**	**400.00**
8173667845	NUnit Pocket Reference, *100 Pages*	Hamilton	100.00
8173665613	Object-Oriented Programming with Visual Basic .NET, *306 Pages*	Hamilton	225.00
8173668264	Office 2003 XML, *596 Pages*	Lenz	450.00
8173667519	OpenOffice.org Writer **(B / CD-ROM)**, *234 Pages*	Weber	250.00
8173667403	Optimizing Oracle Performance, *426 Pages*	Milsap	375.00
8173669287	Oracle Applications Server 10g Essentials, *292 Pages*	Greenwald	275.00
8173661170	Oracle Built-in Packages **(B / DISK)**, *956 Pages*	Feuerstein	475.00
8173667071	Oracle Data Dictionary Pocket Reference, *150 Pages*	Kreines	125.00
8173660670	Oracle Database Administration: The Essential Reference, *552 Pages*	Kreines	325.00
817366417X	Oracle DBA Checklist Pocket Reference, *88 Pages*	RevealNet	65.00
8173660689	Oracle Distributed Systems **(B / DISK)**, *552 Pages*	Dye	325.00
8173664110	Oracle Essentials: Oracle9i, Oracle8i & Oracle8, 2/ed, *382 Pages*	Greenwald	250.00
817366935X	Oracle Essentials: Oracle Database 10g, 3/ed, *368 Pages*	Greenwald	275.00
817366580X	Oracle in a Nutshell: A Desktop Quick Reference, *934 Pages*	Greenwald	600.00
8173664560	Oracle Initialization Parameters Pocket Reference (Oracle Database 10g), *128 Pages*	Kreines	125.00
8173663246	Oracle Net8 Configuration & Troubleshooting, *412 Pages*	Toledo	250.00
8173663629	Oracle PL/SQL Best Practices, *208 Pages*	Feuerstein	175.00
8173661189	Oracle PL/SQL Built-ins Pocket Reference, *78 Pages*	Feuerstein	60.00
8173665176	Oracle PL/SQL Programming, 3/ed, *1,024 Pages*	Feuerstein	600.00
8173662401	Oracle PL/SQL Programming: A Developer's Workbook, *576 Pages*	Feuerstein	300.00
8173661197	Oracle PL/SQL Programming: Guide to Oracle8i Features **(B / DISK)**, *264 Pages*	Feuerstein	235.00
8173668116	Oracle Regular Expression Pocket Reference, *74 Pages*	Burcham	75.00
8173661200	Oracle SAP Administration, *208 Pages*	Burleson	175.00
8173660298	Oracle Scripts **(B / CD-ROM)**, *208 Pages*	Lomansky	250.00
8173660719	Oracle Security, *448 Pages*	Theriault	220.00
8173663637	Oracle SQL* Loader: The Definitive Guide, *278 Pages*	Gennick	175.00
8173669333	**Oracle SQL*Plus Pocket Reference, 3/ed, *160 Pages***	**Gennick**	**125.00**
8173666067	**Oracle SQL*Plus: The Definitive Guide, 2/ed, *592 Pages***	**Gennick**	**400.00**
8173662916	Oracle SQL: The Essential Reference, *424 Pages*	Kreines	200.00
8173661847	**Oracle Utilities Pocket Reference, *136 Pages***	**Mishra**	**100.00**
8173661219	Oracle Web Applications: PL/SQL Developer's Intro, *264 Pages*	Odewahn	180.00
817366028X	Oracle8 Design Tips, *136 Pages*	Ensor	120.00
8173668027	PC Annoyances: How to Fix the Most Annoying Thingsabout your Personal Computer, *236 Pages*	Bass	225.00
8173667152	PC Hacks: 100 Industrial-Strength Tips & Tools, *316 Pages*	Aspinwall	300.00
8173669732	**PC Hardware Annoyances: How to Fix the Most ANNOYING Things About Your Computer Hardware, *276 Pages***	**Bigelow**	**275.00**
817366532X	PC Hardware in a Nutshell: A Desktop Quick Reference, 3/ed, *848 Pages*	Thompson	325.00
8173667128	PDF Hacks: 100 Industrial-Strength Tips & Tools, *308 Pages*	Steward	300.00

8173664463	Perl & XML, *224 Pages*	Ray	175.00
8173661073	Perl 5 Pocket Reference, 3/ed, *96 Pages*	Vromans	70.00
8173667195	Perl 6 Essentials, *208 Pages*	Randal	175.00
8173668280	**Perl Best Practices, *554 Pages***	**Conway**	**425.00**
8173667330	Perl Cookbook, 2/ed, *976 Pages*	Christiansen	675.00
8173668043	Perl Template Toolkit, *600 Pages*	Chamberlain	525.00
8173665710	PHP Cookbook, *638 Pages*	Sklar	550.00
0596008600	Photo Retouching with Photoshop: A Designer's Notebook, *96 Pages*	CLEC'H	775.00
817366871X	Postfix: The Definitive Guide, *288 Pages*	Dent	275.00
8173669805	**Powerpoint 2003 PersonalTrainer (B / CD-ROM), (2 Color book) *342 Pages***	**CustomGuide**	**425.00**
8173660301	Practical C Programming 3/ed, *456 Pages*	Oualline	225.00
8173666822	Practical C++ Programming, 2/ed, *582 Pages*	Oualline	225.00
817366711X	Practical mod_perl, *932 Pages*	Bekman	675.00
8173663920	Practical PostgreSQL **(B / CD-ROM)**, *642 Pages*	Command Prompt Inc.	450.00
8173666733	Practical RDF, *360 Pages*	Powers	325.00
8173666830	Practical Unix & Internet Security, 3/ed, *994 Pages*	Garfinkel	650.00
8173664390	Practical VoIP Using VOCAL, *532 Pages*	Dang	450.00
8173665818	Programming .NET 1.1 Components, *488 Pages*	Lowy	375.00
8173667209	Programming .NET Security, *704 Pages*	Freeman	550.00
8173667411	Programming .NET Windows Applications (Covers .NET 1.1, & Visual Studio .NET 2003), *1316 Pages*	Liberty	750.00
8173664382	Programming .NET Web Services, *500 Pages*	Ferrara	375.00
817366742X	Programming ASP .NET (Covers .NET 1.1, & Visual Studio .NET 2003), 2/ed, *1026 Pages*	Liberty	675.00
8173669651	**Programming C# (Covers C# 2.0, .NET 2.0 & Visual Studio 2005), 4/ed, *680 Pages***	**Liberty**	**525.00**
817366076X	Programming Embedded Systems in C & C++, *198 Pages*	Barr	150.00
8173669694	**Programming Flash Communication Server, *842 Pages***	**Lesser**	**600.00**
8173661278	Programming Internet E-mail, *384 Pages*	Wood	225.00
8173668183	Programming Jakarta Struts 2/ed, *470 Pages*	Cavaness	400.00
8173662657	Programming Perl 3/ed, *1,116 Pages*	Wall	675.00
8173663114	Programming PHP, *530 Pages*	Lerdorf	350.00
8173667500	Programming Python (Covers Python 2) **(B / CD-ROM)**, 2/ed, *1,305 Pages*	Lutzf	750.00
8173662371	Programming the Perl DBI, *372 Pages*	Descartes	200.00
8173667063	Programming Visual Basic .NET 2003, 2/ed, *564 Pages*	Liberty	425.00
8173665737	Programming Web Services with Perl, *492 Pages*	Ray	400.00
8173662045	Programming Web Services with SOAP, *268 Pages*	Snell	200.00
817366207X	Programming Web Services with XML-RPC, *240 Pages*	St.Laurent	175.00
8173668175	Programming with QT (Covers Qt 3), 2/ed, *532 Pages*	Dalheimer	450.00
817366479X	**Python Cookbook (Covers Python 2.3 & 2.4), 2/ed, *852 Pages***	**Martelli**	**600.00**
8173669708	**Python Pocket Reference (Covers Python 2.4), 3/ed, *168 Pages***	**Lutz**	**125.00**
8173666857	Python in a Nutshell: A Desktop Quick Reference (Cover Python 2.2), *662 Pages*	Martelli	500.00
8173668485	qmail, *268 Pages*	Levine	275.00
817366689X	Real World Web Services, *230 Pages*	Iverson	225.00
8173667276	Regular Expression Pocket Reference, *112 Pages*	Stubblebine	100.00
0596007191	Revolution in the Valley, *324 Pages*	Hertzfeld	775.00
8173665206	Running Linux, 4/ed, *702 Pages*	Welsh	425.00
8173667055	Samba Pocket Reference, 2/ed, *146 Pages*	Eckstein	125.00
8173664226	SAX2 (Simple API for XML), *248 Pages*	Brownell	175.00
8173667217	Secure Coding: Principles & Practices, *200 Pages*	Graff	225.00
8173667284	Secure Programming Cookbook for C and C++, *800 Pages*	Viega	600.00
817366840X	Security Warrior, *562 Pages*	Peikari	500.00
8173663262	Securing Windows NT/2000 Servers for the Internet, *200 Pages*	Norberg	125.00

8173669376	Securing Windows Server 2003, *456 Pages*	Danseglio	400.00
8173660786	sed & awk, 2/ed, *440 Pages*	Dougherty	300.00
817366918X	SELINUX NSA's Open Source Security Enhanced Linux, *264 Pages*	McCarty	275.00
8173665834	sendmail, 3/ed, *1,238 Pages*	Costales	850.00
817366823X	sendmail Cookbook, *418 Pages*	Hunt	400.00
8173666865	Sequence Analysis in a Nutshell: A Guide to Common Tools and Databases (Covers EMBOSS 2.5.0), *310 Pages*	Markel	275.00
8173664161	Server Load Balancing,*198 Pages*	Bourke	150.00
817366739X	**SharePoint User's Guide, *158 Pages***	**IDC**	**150.00**
817366983X	**SharePoint Office Pocket Guide, *94 Pages***	**Webb**	**75.00**
8173669503	**Snort Cookbook, *296 Pages***	**Orebaugh**	**300.00**
8173663866	Solaris 8 Administrator's Guide, *308 Pages*	Watters	225.00
8173669198	SpamAssassin (Covers 3.0), *232 Pages*	Schwartz	250.00
8173668191	Spidering Hacks: 100 Industrial - Strength Tips & Tools, *436 Pages*	Hemenway	350.00
817366837X	**Spring: A Developer's Notebook, *202 Pages***	**Tate**	**200.00**
8173666520	SQL in a Nutshell (Covers SQL Server, DB2, MySQL, Oracle & PostgreSQL), 2/ed, *720 Pages*	Kline	450.00
8173667438	**SQL Pocket Reference (Cover Oracle. DB2, SQL Server & MySQL), *170 Pages***	**Gennick**	**125.00**
8173668248	SQL Tunning (Covers Oracle, DB2 & SQL Server), *356 Pages*	Tow	325.00
8173668418	Squid: The Definitive Guide, *472 Pages*	Wessels	450.00
8173662924	SSH: The Secure Shell: The Definitive Guide, *564 Pages*	Barrett	325.00
8173668574	STL Pocket Reference, *136 Pages*	Lischner	100.00
8173669511	SWT: A Developer's Notebook, *330 Pages*	Hatton	325.00
8173664811	**Swing Hacks, *554 Pages***	**Marinacci**	**425.00**
8173665311	**Switching to VoIP, *514 Pages***	**Wallingford**	**450.00**
8173663254	T1: A Survival Guide, *312 Pages*	Gast	225.00
817366093X	Tcl/Tk in a Nutshell: A Desktop Quick Reference, *480 Pages*	Raines	240.00
8173664676	TCP/IP Network Administration 3/ed, *756 Pages*	Hunt	500.00
8173666512	**Test Driving Linux: A Desktop User's Guide (B / CD-ROM), *372 Pages***	**Lavigna**	**350.00**
8173666873	Tomcat: The Definitive Guide (Cover Tomcat 4), *336 Pages*	Brittain	300.00
8173660352	UML in a Nutshell: A Desktop Quick Reference, *336 Pages*	Alhir	225.00
8173667225	UML Pocket Reference, *96 Pages*	Pilone	100.00
8173665893	Understanding the Linux Kernel, 2/ed, *832 Pages*	Bovet	500.00
817366627X	**Unit Test Frameworks (B / CD-ROM), *222 Pages***	**Hamill**	**225.00**
8173661324	UNIX in a Nutshell: A Desktop Quick Reference for SVR4 & Solaris 7, 3/ed, *628 Pages*	Robbins	325.00
8173665656	Unix Power Tools, 3/ed, *1,162 Pages*	Powers	750.00
8173666202	Upgrading to PHP 5 (Covers MySQL 4.1), *358 Pages*	Trachtenberg	350.00
8173660948	Using & Managing PPP, *464 Pages*	Sun	240.00
8173665842	Using Samba, 2/ed, *570 Pages*	Eckstein	500.00
8173664374	VB .NET Core Classes in a Nutshell: A Desktop Quick Reference **(B / CD-ROM)**, *584 Pages*	Kurniawan	500.00
817366594X	VB .NET Language Pocket Reference, *160 Pages*	Roman	125.00
8173666881	VBScript in a Nutshell: A Desktop Quick Reference, 2/e, *528 Pages*	Lomax	400.00
8173663300	VBScript Pocket Reference, *118 Pages*	Childs	60.00
8173668582	**Version Control With Subversion, *332 Pages***	**Collins - Sussman**	**350.00**
8173662622	vi Editor Pocket Reference, *76 Pages*	Robbins	60.00
8173661340	Virtual Private Networks, 2/ed, *228 Pages*	Scott	150.00
8173666164	**Visual Basic 2005: A Developer's Notebook, *272 Pages***	**MacDonald**	**250.00**
8173669740	**Visual C# 2005: A Developer's Notebook, *250 Pages***	**Liberty**	**225.00**
8173660964	Visual Basic Controls in a Nutshell, *512 Pages*	Dictor	310.00
817366501X	**Visual Studio Hacks: Tips & Tools for Turbocharging the IDE, *304 Pages***	**Avery**	**375.00**

8173668094	Volume One: Xlib Programming Manual (Version 11), *824 Pages*	Nye	575.00
8173668108	Volume Two: Xlib Reference Manual (Version 11), 3/ed, *948 Pages*	Nye	600.00
8173668086	Volume Zero: X Protocol Reference Manual (Version X11), *468 Pages*	Nye	350.00
8173668310	X Windows Reference Manaul (3 Volume Set)	Nye	1,350.00
0596007337	We the Media: Grassroots Journalism by the People,for the People, *320 Pages*	Gillmor	775.00
8173663092	WebLogic 8.1: The Definitive Guide, *860 Pages*	Mountjoy	600.00
8173669058	**Web Database Application with PHP & MySQL (Covers PEAR, PHP 5 & MySQL 4.1), 2/ed, *828 Pages***	**Willaims**	**600.00**
8173663750	Web Design in a Nutshell: A Desktop Quick Reference, 2/ed, *656 Pages*	Niederst	425.00
8173664412	Web Performance Tuning, 2/ed, *488 Pages*	Killelea	350.00
8173665214	Web Privacy with P3P, *350 Pages*	Cranor	300.00
8173663947	Web Security, Privacy & Commerce, 2/ed, *768 Pages*	Garfinkel	500.00
8173663394	Web Services Essentials, *320 Pages*	Cerami	200.00
8173665931	WebLogic Server 6.1 Workbook for Enterprise JavaBeans, 3/ed, *264 Pages*	Nyberg	200.00
8173661308	The Whole Internet: The Next Generation, *576 Pages*	Conner/Krol	425.00
8173661367	Win32 API Programming with Visual Basic **(B / CD-ROM)**, *534 Pages*	Roman	400.00
8173662789	Windows 2000 Administration in a Nutshell: A Desktop Quick Reference, *1000 Pages*	Tulloch	350.00
8173663319	Windows 2000 Commands Pocket Reference, *122 Pages*	Frisch	60.00
8173660743	Windows 2000 Quick Fixes, *304 Pages*	Boyce	225.00
8173660883	Windows NT TCP/IP Network Administration, *512 Pages*	Hunt	250.00
8173666903	Windows Server 2003 in a Nutshell: A Desktop Quick Reference, *672 Pages*	Tulloch	550.00
8173668833	Windows Server Hacks: 100 Industrial-Strength Tips & Tools , *328 Pages*	Tulloch	325.00
8173667454	Windows XP Hacks: 100 Industrial-Strength Tips & Tools, *294 Pages*	Gralla	350.00
8173663564	Windows XP in a Nutshell: A Desktop Quick Reference, *640 Pages*	Karp	375.00
8173669643	**Windows XP Personal Trainer (B / CD-ROM), *480 Pages***	**CustomGuide**	**550.00**
8173666911	Windows XP Pocket Reference, *196 Pages*	Karp	125.00
8173665915	Windows XP Unwired: A Guide for Home, Office, and the Road, *316 Pages*	Lee	275.00
8173667462	Wireless Hacks: 100 Industrial-Strength Tips & Tools, *424 Pages*	Flickenger	275.00
8173662770	Word 2000 in a Nutshell: A Power User's Quick Reference, *516 Pages*	Glenn	225.00
817366692X	Word Pocket Guide, *160 Pages*	Glenn	125.00
8173665354	Writing Excel Macros with VBA, 2/ed, *580 Pages*	Roman	450.00
8173660778	Writing Word Macros, *416 Pages*	Roman	275.00
8173668450	XML in a Nutshell (Covers XML 1.1 & XInclude), 3/ed, *724 Pages*	Harold	500.00
8173666156	XML Hacks: 100 Industrial-Strength Tips & Tools, *490 Pages*	Fitzgerald	425.00
8173663343	XML Pocket Reference, 2/ed, *128 Pages*	Eckstein	100.00

FORTHCOMING TITLES

August 2005

8173664587	ASP .NET 2.0: Developer's Notebook, *358 Pages*	Lee	325.00
8173661995	Access 2003 Personal Trainer **(B / CD-ROM)**, (2 Color book), *372 Pages*	CustomGuide	400.00
0596100965	Adobe Photoshop CS2 One-on-One **(B / DVD)**, *476 Pages*	McClelland	1,900.00
8173667721	Car PC Hacks: Tips & Tools for Geeking Your Ride, *394 Pages*	Stolarz	400.00
8173667799	Computer Privacy Annoyances, *218 Pages*	Tynan	225.00
8173664633	Firefox Hacks, *398 Pages*	McFarlane	350.00
8173667861	Home Networking: The Missing Manual, *263 Pages*	Lowe	275.00
8173670161	Learning Java **(B/CD-ROM)**, 3/ed, *946 Pages*	Niemeyer	600.00
8173664684	Linux Desktop Hacks: Tips & Tools for Customizing and Optimizing your OS, *384 Pages*	Petreley	350.00
8173668272	PC Pest Control, *298 Pages*	Gralla	300.00
8173668361	Perl Testing: A Developer's Notebook, *218 Pages*	Langworth	225.00
8173668426	QuickTime for Java: A Developer's Notebook, *266 Pages*	Adamson	275.00
8173666032	TOAD Pocket Reference for Oracle, 2/ed, *126 Pages*	McGrath	125.00
8173666172	Windows Server Cookbook: For Windows Server 2003 & Windows 2000, *706 Pages*	Allen	550.00

8173669716	Word 2003 Personal Trainer **(B / CD-ROM)**, *462 Pages*	CustomGuide	450.00
8173666113	Word Annoyances, *218 Pages*	Hart	225.00
	September 2005		
817367017X	Cisco IOS in a Nutshell, 2/ed, *808 Pages*	Boney	500.00
059600849X	Commercial Photoshop Retouching: In The Studio, *410 Pgs*	Honiball	2,000.00
8173670196	Creating Microsoft SharePoint Sites, *314 Pages*	Webb	300.00
817367020X	Digital Identity, *274 Pages*	Windley	1,600.00
0596100159	Digital Photography Pocket Guide, 3/ed, *160 Pages*	Story	700.00
8173670226	Eclipse IDE Pocket Guide, *140 Pages*	Burnette	125.00
8173670234	Essential Business Process Modeling, *346 Pages*	Havey	325.00
8173670250	Open Source for the Enterprise, *246 Pages*	Woods	325.00
8173670269	Open Sources 2.0, *314 Pages*	DiBona	325.00
8173670277	Oracle DBA Pocket Guide, *128 Pages*	Kreines	125.00
8173670285	RT Essentials, *234 Pages*	Vincent	225.00
8173670293	Security and Usability, *600 Pages*	Cranor	500.00
8173670307	XSLT 1.0 Pocket Reference, *116 Pages*	Lenz	100.00
	October 2005		
8173670315	Access Annoyances: How to Avoid the Most Annoying Things About Your Favorite Database , *314 Pages*	Mitchell	325.00
8173670323	Access for Rookies: The Missing Manual, *314 Pages*	Palmer	300.00
8173670331	Asterisk: The Future of Telephony, *514 Pages*	Smith	450.00
817367034X	Beyond Java, *170 Pages*	Tate	175.00
8173670358	Designing Interfaces, *384 Pages*	Tidwell	350.00
8173670366	Essentail PHP Security, *310 Pages*	Shiflett	300.00
8173670374	Excel for Rookies: The Missing Manual, *314 Pages*	MacDonald	300.00
8173670382	Learning SQL, *410 Pages*	Beaulieu	350.00
8173670390	Linux Desktop Pocket Reference, *154 Pages*	Brickner	125.00
0596100302	Photoshop Retouching Cookbook for Digital Photographers, *176 Pages*	Huggins	1,400.00
8173670412	Prefactoring, *200 Pages*	Pugh	200.00
8173670420	Programming Apache Axis, *410 Pages*	Haddad	400.00
8173670439	Programming Visual Basic 2005, *800 Pages*	Liberty	500.00
8173670447	Visual Basic 2005 Jumpstart Pocket Guide, *164 Pages*	Lee	125.00
	November 2005		
8173670455	Blackberry Hacks, *256 Pages*	Mabe	275.00
8173670463	Integrating Excel and Access, *514 Pages*	Schmalz	425.00
8173670471	Internet Forensics, *304 Pages*	Jones	325.00
817367048X	PHP in a Nutshell, *314 Pages*	Hudson	300.00
0596100221	Photoshop Photo Effects Cookbook, *176 Pages*	Shelbournce	1,400.00
0596008511	Photoshop RAW, *266 Pages*	Aaland	1,600.00
817367051X	Powerpoint Annoyances, *266 Pages*	Swinford	250.00
8173670528	Practical Development Environments, *352 Pages*	Doar	350.00
8173670536	Programming Avalon **(B / CD - ROM)**, *410 Pages*	Sells	400.00
8173670544	Producing Free Software, *266 Pages*	Fogel	275.00
8173670552	Twisted Network Programming Essentials, *210 Pages*	Fettig	225.00
8173670560	Yahoo ! Hacks, *362 Pages*	Bausch	325.00
	December 2005		
8173670579	JBoss at Work: A Practical Guide, *256 Pages*	Marrs	250.00
8173670587	Oracle PL/SQL for DBAs, *466 Pages*	Feuerstein	425.00
8173670595	Programming SQL Server 2005, *362 Pages*	Wildermuth	325.00
	February 2006		
8173670609	Bonjour: The Definitive Guide, *256 Pages*	Steinberg	275.00

8173670617	C in a Nutshell, *400 Pages*	Prinz	275.00
8173670625	Firewall Warrior, *610 Pages*	Artymiak	500.00
8173670633	JUnit: The Definitive Guide, *310 Pages*	Lane	300.00

Shroff / Object Sources

PUBLISHED TITLES

8173668760	Struts Survival Guide: Basics to Best Practices, *224 Pages*	Shenoy	225.00

FORTHCOMING TITLES

8173668779	J2EE Project Survival Guide Volume 1
8173668787	J2EE Project Survival Guide Volume 2

Shroff / Pearson Education

8129703769	.NET - A Complete Development Cycle **(B / CD-ROM)**, *584 Pages*	Lenz	600.00
8129703882	Applying Enterprise JavaBeans: Component-Based Development for the J2EE Platform, 2/ed, *486 Pages*	Matena	375.00
8129703777	ASP .NET Solutions - 23 Case Studies, *894 Pages*	Leinecker	725.00

Shroff / Rampant TechPress

PUBLISHED TITLES

8173669112	**Conducting the J2EE Job Interview: IT Manager Guide for J2EE with Interview Questions, *244 Pages***	**Hunter**	**250.00**
8173669147	**Conducting the Java Job Interview: IT Manager Guide for Java with Interview Questions, *308 Pages***	**Hunter**	**300.00**
8173669155	Conducting the Network Administrator Job Interview: IT Manager Guide with Cisco CCNA Job Interview Questions, *146 Pages*	Haeder	150.00
8173667764	Conducting the Oracle Job Interview Tips - IT Manager Guide for Oracle Job Interviews with Interview Questions (Includes Oracle 9i), *144 Pages*	Ault	150.00
8173669139	Conducting the Programmer Job Interview: IT Manager Guide with Java, J2EE, C, C++, Unix, PHP and Oracle Interview Questions!, *308 Pages*	Burleson	300.00
8173669074	Conducting the UNIX Job Interview: IT Manager Guide with Unix Interview Questions, *180 Pages*	Haeder	200.00
8173667829	**Conducting the Web Designer Job Interview: IT Manager J2EE Job Interview Questions, *308 Pages***	**Burleson**	**300.00**
8173669082	Conducting the Web Master Job Interview: IT Manager Guide with Webmaster Interview Questions!, *372 Pages*	Burleson	350.00
817366773X	Creating a Self-Tuning Oracle Database - Automating Oracle9i Dynamic SGA Performance, *156 Pages*	Burleson	150.00
8173664927	**Easy Oracle Automatic: Oracle10g Automatic Storage, Memory and Diagnostic Features, *300 Pages***	**Dr. Kumar**	**300.00**
8173667810	Mike Ault's Oracle Internals Monitoring & Tuning Scripts - Advanced Internals & OCP Certification Insights for the Master DBA (Includes Oracle 10g), *368 Pages*	Ault	350.00
8173668523	OCP Instructor's Guide for Oracle DBA Certification: A Study Guide to Advanced Oracle Certified Professional Database Administration Techniques, *340 Pages*	Foot	325.00
817366854X	**Oracle10g Grid & Real Application Clusters: Oracle10g Grid Computing with RAC, *868 Pages***	**Ault**	**600.00**
8173668558	Oracle Database 10g New Features: Oracle10g Reference for Advanced Tuning & Administration, *548 Pages*	Ault	450.00
8173664919	**Oracle Dataguard: Standby Database Failover Handbook, *364 Pgs***	**Kumar**	**325.00**
8173669104	Oracle Disk I / O Tuning: Disk IO Performance &		

	Optimization for Oracle 10g Database, *244 Pages*	Ault	250.00
8173667748	Oracle Performance Troubleshooting – with Dictionary Internals SQL & Tuning Scripts (Includes Oracle 9i),*268 Pages*	Schumacher	250.00
8173668515	Oracle Privacy Security Auditing: Includes Federal Law Compliance with HIPAA, Sarbanes-Oxley & The Gramm-Leach-Bliley Act GLB (Includes Oracle 10g), *692 Pages*	Nanda	575.00
8173667756	Oracle9i RAC - Oracle Real Application Clusters Configuration and Internals, *628 Pages*	Ault	500.00
8173667780	Oracle Replication - Snapshot, Multi-Master & Materialized Views Scripts (Includes Oracle 10g), *244 Pages*	Garmany	250.00
8173669546	**Oracle Solid State Disk Tuning: High Performance Oracle Tuning with RAM Disk, *204 Pages***	**Burleson**	**200.00**
8173669090	**Oracle SQL Tuning & CBO Internals (Includes Oracle 10g), *354 Pages***	**Floss**	**350.00**
8173664994	**Oracle Streams: High Speed Replication & Data Sharing, *308 Pgs***	**Tumma**	**300.00**
8173667772	Oracle Utilities - Using Hidden Programs, Import / Export, SQL*Loader, Oradebug, Dbverify, Tkprof & More (Includes Oracle 9i), *276 Pages*	Moore	250.00
8173664978	**Oracle Wait Event Tuning: High Performance with Wait Event Interface Analysis (Includes 10g), *276 Pages***	**Andert**	**275.00**
8173669120	**Top Answers to Job Interview Questions: Includes questions you must ask at every interview, *148 Pages***	**Burleson**	**150.00**

FORTHCOMING TITLES

August 2005

8173664854	Conducting the Web Services Job Interview: IT Manager Guide for Web Services Job Interviews with Delphi, .Net, XML UDDI, SOAP, ASP .NET & WSL Web Services Job Interview Questions, *256 Pages*	Brown	250.00
8173664900	Conducting the Windows Advanced Server Job Interview: IT Manager Guide to Windows Advanced Server Job Interviews with Itanium, .NET & Windows Advanced Server Oracle Job Interview Questions, *276 Pages*	Burleson	275.00
8173669872	Easy Linux Commands: Working Examples of Linux Command Syntax, *250 Pages*	Clark	250.00
8173669902	Easy Oracle Jumpstart: Oracle Database Management Concepts &Administration (Includes 10g), *200 Pages*	Freeman	200.00
8173664935	Easy Oracle PL/SQL Programming: Get Started Fast with Working PL/SQL Code Examples (Includes 10g), *200 Pages*	Col. Garmany	200.00
8173669813	Easy Oracle SQL: Get Started Fast Writing SQL Reports with SQL*Plus, *200 Pages*	Col. Garmany	200.00
8173669821	Oracle Job Scheduling: Creating Robust Task Management with dbms_job and Oracle 10g dbms_Scheduler, *240 Pages*	Dr. Hall	225.00
8173669910	Oracle RAC & Grid Tuning with Solid - State Disk: Experts Secrets for High Performance Clustered Grid Computing (Includes 10g), *230 Pages*	Ault	225.00
8173669538	Oracle Silver Bullets: Real - World Oracle Performance Secrets, *200 Pages*	Burleson	200.00
8173669856	Oracle Tuning: Oracle Time - Series Optimization with the Automatic Workload Repository, *640 Pages*	Burleson	500.00
817366661X	Oracle Tuning Power Scripts with 100+ High Performance SQL Scripts, *300 Pages*	Ault	300.00
8173669937	You're Fired! Firing Computer Professionals: The IT Manager Guide for Terminating "With Cause", *240 Pages*	Papaj	250.00

September 2005

8173669848	Easy Java Debugging: Java Testing & Development for Speed & Quality, *240 Pages*	Gradner	250.00
8173669945	High Performance SQL Server DBA Tuning & Optimization Secrets (Splash "SQL Server 2005 Yukon") , *240 Pages*	Schumacher	250.00
8173669953	Super SQL Server Systems: Turbocharge Database Performance with C++ External Procedures, *450 Pages*	Gamma	400.00

November 2005

8173669880	Easy Oracle HTML - DB: Creat Dynamic Web Pages with Oracle, *400 Pages*	Cunningham	350.00
8173669929	Personal Oracle RAC Clusters: Create Oracle 10g Grid		

	Computing At - Home, *200 Pages*	Stoever	200.00
	February 2006		
8173670641	Easy Oracle Data Mining Expert Secrets for complex High Performance Programming, *230 Pages*	Dr. Hamm	225.00
8173669899	Easy Oracle PHP By Example: Easy Dynamic Web Pages with Oracle Data, *400 Pages*	Gogala	350.00
817367065X	Oracle Best Practices The Roadmap for the Oracle DBA, *230 Pages*	Freeman	225.00
8173670668	Oracle PL/SQL Tuning Expert Secrets for High Performance Programming (Covers 10g), *230 Pages*	Dr. Hall	225.00
8173670676	SQL Best Practices Design Patterns to improve coding style, *230 Pages*	Tropashko	225.00

Shroff / RDS Press

817366904X	**Loosely Coupled: The Missing Pieces of Web Services, *378 Pages***	**Kaye**	**350.00**

Shroff / Springer

8181281756	Beyond Fear, *398 Pages*	Schneier	450.00
8181280865	Enterprise Management with SAP SEM Business Analytics, *232 Pages*	Meier	225.00
8181281195	Guide to J2EE: Enterprise Java, *690 Pages*	Hunt	575.00
8181281187	Guide to the Unified Process featuring UML, Java and Design Patterns, *444 Pages*	Hunt	400.00
8181280857	SAP R/3 Implementation Methods & Tools, *195 Pages*	Hans-Jurgen	175.00

Shroff / Sybex

8173668124	Mastering Rational XDE, *636 Pages*	Boggs	500.00
8173668132	MCAD / MCSD: **(Exam70-310)** Visual Basic .NET XML Web Service and Server Components Study Guide, **(B / CD-ROM)**, *640 Pages*	Fanstill	500.00
8173668140	PMP - Project Management Professional Workbook, *303 Pages*	Baca	200.00
8173668159	Stop Staring: Facial Modeling and Animation Done Right, **(B / CD-ROM)** *353 Pages*	Osipa	500.00

Shroff / Syngress

8173665362	**Aggressive Network Self - Defence, *422 Pages***	**Archibald**	**425.00**
8173667608	The Best Damn Cisco Internetworking Book Period, *1,168 Pages*	Riley	850.00
8173665001	The Best Damn Windows Server 2003 Book Period, *1,042 Pages*	Snehaker	750.00
8173665184	**Black Hat Physical Device Security: Exploting Hardware and Software, *422 Pages***	**Miller**	**400.00**
8173665877	**Buffer OverFlow Attacks: Defect, Exploit, Prevent, *526 Pages***	**Foster**	**475.00**
8173667896	Building A Cisco Wireless LAN, *532 Pages*	Ouellet	450.00
8173667969	Building DMZS for Enterprise Network, *832 Pages*	Schmied	600.00
8173667527	Building SANs With Brocade Fabric Switches, *480 Pages*	Judd	400.00
8173667934	Check Point NG / AI:Next Generation with Application Intelligence Security Administration, *628 Pages*	Tobkin	525.00
8173667616	Cisco Security Professional's Guide to Secure Intrusion Detection Systems, *676 Pages*	Baumrucker	550.00
8173667535	Cisco Security Specialist's Guide to PIX Firewall, *608 Pages*	Osipov	525.00
8173666253	Configuring and Troubleshooting Windows XP Professional **(B / CD-ROM)**, *818 Pages*	Grasdal	600.00
8173666245	Configuring Cisco Voice Over IP, 2/ed, *578 Pages*	Flannagan	450.00
8173665826	**Configuring NetScreen Firewalls, *740 Pages***	**Cameron**	**550.00**
8173667578	Configuring Symantec AntiVirus: Corporate Edition, *756 Pages*	Hunter	600.00
8173668957	CYA Securing Exchange Server 2003 & Outlook Web Access, *340 Pages*	Walther	400.00
8173668965	CYA Securing IIS 6.0, *424 Pages*	Cheah	425.00
8173665044	Cyber Adversary Characterization: Auditing the Hacker Mind, *512 Pages*	Parker	375.00
8173669961	**Cyber Spying: Tracking Your Family's (Sometimes) Secret Online Lives, *470 Pages***	**Fair**	**475.00**
8173666237	Dr. Tom Shinder's ISA Server and Beyond **(B / CD-ROM)**, *866 Pages*	Shinder	600.00
8173665060	**Deploying Citrix MetaFrame Presentation Server 3.0 with Windows Server 2003 Terminal Services, *600 Pages***	**Wilson**	**500.00**

817366787X	Designing a Wireless Network, *410 Pages*	Wheat	350.00
8173668752	Ethereal Packet Sniffing (B / CD-ROM), *510 Pages*	Orebaugh	475.00
8173666105	**Google Hacking for Penetration Testers, *532 Pages***	**Long**	**400.00**
817366790X	Hack Proofing Your Wireless Network, *514 Pages*	Barnes	450.00
8173665850	**Hacking a Terror Network: The Silent Threat of Covert Channels (B / CD-ROM), *406 Pages***	**Rogers**	**425.00**
8173668922	Hacking the Code: ASP .NET Web Application Security, *474 Pages,*	Burnett	450.00
8173668744	Hardware Hacking, *566 Pages*	Grand	500.00
8173669562	How to Cheat at Managing Microsoft Windows Small Business Server 2003, *504 Pages*	Snedaker	500.00
8173666636	**Infosec Career Hacking: Sell Your Skillz, Not Your Soul, *480 Pages***	**Bayles**	**425.00**
8173665869	Inside the SPAM Cartel: Trade Secrets From The Dark Side, *436 Pages*	Spammer - X	425.00
8173665079	**Intrusion Prevention and Active Response: Deploying Network and Host IPS , *432 Pages***	**Rash**	**400.00**
8173668914	Managing and Securing a Cisco Structured Wireless-Aware Network, *500 Pages*	Wall	475.00
8173667888	Managing Cisco Network Security, 2/ed, *784 Pages*	Knipp	600.00
8173667705	MCSA/MCSE **(Exam 70-290)** Managing and Maintaining a Windows Server 2003 Environment: Study Guide & DVD Training System **(B / DVD)**, *1,028 Pages*	Todd	750.00
8173667659	MCSA/MCSE **(Exam 70-291)** Implementing, Managing, and Maintaining a Windows Server 2003 Network Infrastructure: Study Guide & DVD Training System **(B / DVD)**, *1,076 Pages*	Snedaker	750.00
8173667675	MCSA/MCSE **(Exam 70-292)**: Managing and Maintaining a Windows Server 2003 Environment for an MCSA Certified on Windows 2000 Study Guide & DVD Training System **(B / DVD)**, *739 Pages*	Schmied	600.00
8173667640	MCSE **(Exam 70-293)** Planning and Maintaining a Windows Server 2003 Network Infrastructure: Study Guide & DVD Training System **(B / DVD)**, *1,120 Pages*	Cross	800.00
8173667632	MCSE **(Exam 70-294)** Planning, Implementing, and Maintaining a Windows Server 2003 Active Directory Infrastructure: Study Guide & DVD Training System **(B / DVD)**, *1,068 Pages*	Cross	750.00
8173667683	MCSE **(Exam 70-296)** Planning, Implementing and Maintaining a Windows Server 2003 Environment for an MCSE Certified on Windows 2000: Study Guide & DVD Training System **(B / DVD)**, *848 Pages*	Hunter	700.00
8173667713	MCSE **(Exam 70-297)** Designing a Windows Server 2003 Active Directory and Network Infrastructure: Study Guide & DVD Training System **(B / DVD)**, *688 Pages*	Barber	600.00
8173667667	MCSE **(Exam 70-298)** Designing Security for a Windows Server 2003 Network: Study Guide & DVD Training System **(B / DVD)**, *818 Pages*	Khnaser	650.00
8173667594	MCSE / MCSA **(Exam 70-214)** Implementing & Administering Security in a Windows 2000 Network: Study Guide & DVD Training Systems **(B / DVD)**, *866 Pages*	Schmied	600.00
817366997X	**Microsoft Log Parser Toolket, *470 Pages***	**Giuseppini**	**450.00**
8173665338	Nessus Network Auditing **(B / CD-ROM)**, *548 Pages*	Beale	550.00
8173665745	Programmer's Ultimate Security DeskRef, *612 Pages*	Foster	525.00
8173667586	Rick Gallaher's MPLS Training Guide: Building Multi Protocol Label Switching Networks, *324 Pages*	Gallaher	300.00
8173668868	Security Sage's Guide to Hardening the Network Infrastructure, *546 Pages*	Andrés	450.00
8173667543	Security+ Study Guide & DVD Training System **(B / DVD)**, *866 Pages*	Cross	650.00
8173668701	Security Assessment: Case Studies for Implementing the NSA IAM, *468 Pages*	Miles	425.00
8173668949	Snort 2.1 Intrusion Detection, 2/ed, **(B / CD-ROM)** *790 Pages*	Alder	650.00
8173667551	SSCP Study Guide and DVD Training System **(B / DVD)**, *784 Pages*	Jacob	600.00
8173665095	**Sockets, Shellcode, Porting, and Coding: Reverse Engineering Exploits and Tool Coding for Security Professionals, *704 Pages***	**Foster**	**550.00**

817366756X	Stealing the Network: How to Own the Box, *330 Pages*	Russell	300.00
8173668930	Stealing the Network: How to Own a Continent, *424 Pages*	Rogers	450.00
8173668841	WarDriving: Drive, Detect, Defend: A Guide to Wireless Security, *524 Pages*	Hurley	400.00
8173665575	Windows to Linux Migration Toolkit **(B / CD-ROM)**, *532 Pages*	Allen	475.00
8173665699	Wireless Hacking: Projects for Wi - Fi Enthusiasts, *364 Pages*	Barken	375.00

FORTHCOMING TITLES

August 2005

8173666962	Cisco PIX Firewalls: Configure / Manage / Troubleshoot, *608 Pages*	Khan	500.00
8173667160	Cisco Voice Over IP Security, *608 Pages*	Dugan	500.00
8173667349	Dr. Tom Shinder's Configuring ISA Server 2004, *608 Pages*	Shinder	500.00
8173667624	Network + Study Guide & Practice Exams: The Gold Standard of Certification Prep, *848 Pages*	Honick	600.00
8173665907	Penetration Testing with Google Hacks, *304 Pages*	Long	300.00
8173667691	Stealing the Network: How to Own an Identity, *450 Pages*	Mullen	400.00

September 2005

8173670684	How to Cheat at IT Project Management: Clear, Concise, Easy-To-Read to Effective IT Project Management, *416 Pages*	Snedakar	400.00
8173670692	Nessus, Snort, & Ethereal Power Tools: Customizing Open Source Security Applications, 400 Pgs	Caswell	375.00
8173670706	Software Piracy Exposed, *400 Pages*	Honick	400.00

October 2005

8173670714	Configuring Check Point NGX VPN-1/Firewall-1, *608 Pages*	Stiefel	500.00
8173670722	RFID Security, *448 Pages*	Lindstorm	450.00
817367003X	Securing IM and P2P Applications for the Enterprise, *650 Pages*	Sachs	550.00

Thomson Learning / Autodesk Press

0766837831	3D Studio Max 4 Ground Rules **(B / CD-ROM)**, *638 Pages*	Peterson	850.00
0766812340	AutoCAD 2000: A Problem-Solving Approach **(B / CD-ROM)** , *1,262 Pages*	Tickoo	200.00
0766838471	AutoCAD 2002 for Architecture, *1,040 Pages*	Jefferis	200.00
076683851X	AutoCAD 2002: 3D Modeling - A Visual Approach **(B / CD-ROM)**, *644 Pages*	Wilson	200.00
0766838536	AutoCAD 2002: A Problem-Solving Approach, *1,127 Pages*	Tickoo	200.00
0766842533	AutoCAD 2002: Complete **(B / CD-ROM)**, *840 Pages*	Burchard	200.00
0766843696	AutoCAD 2002 Professional **(B / CD-ROM)**, *370 Pages*	Burchard	200.00
9812542604	AutoCAD 2004 for Architecture, *942 Pages*	Jefferis	675.00
0619130598	CCNA CertBlaster Studennt Edition for Cisco Certified Network Associate (CCNA) Exam # 640-607 **(CD-ROM Only)**	Thomson	525.00
0766841456	Combustion Ground Rules **(B / CD-ROM)**, *408 Pages*	Peterson	1,175.00
0766828611	Construction Scheduling with Primavera Enterprise, 2/ed, *390 Pgs*	Marchman	700.00
0766812405	Customizing AutoCAD 2000, *561 Pages*	Tickoo	200.00
0766838528	Customizing AutoCAD 2002, *484 Pages*	Tickoo	200.00
0766820084	Director 8 & Lingo (Inside Macromedia) **(B / CD-ROM)**, *656 Pages*	Wilson	800.00
076682909X	Flash 5 Graphics, Animation & Interactivity **(B / CD-ROM)**, *642 Pages*	Mohler	1,060.00
1401827551	Harnessing 3ds Max 5 **(B / CD-ROM)**, *908 Pages*	Bousquet	1,150.00
0766838463	Harnessing AutoCAD 2002 **(B / CDROM)**, *1,215 Pages*	Stellman	200.00
053868822X	HTML & JavaScript: Programming, *166 Pages*	Stellman	300.00
9812542582	Java: A Framework for Program Design and DataStructures, 2/ed **(B / CD-ROM)**, *640 Pages*	Lambert	500.00
9812542590	Java Programming: Advanced Topics, 3/ed **(B / 3CD-ROM)**, *1008 Pages*	Wigglesworth	750.00
0766834700	Mastering the Art of Production with 3D Studio Max 4, **(B / CD-ROM)**, *1,640 Pages*	Bousquet	675.00
981243870X	Oracle 9i Database Administrator: Implementation & Administration, **(B / CD-ROM)**, *562 Pages*	Mccullough-Dieter	575.00

981243867X	Oracle 9i Developers: PL/SQL Programming, *560 Pages*	Casteel	475.00
981243884X	Oracle 9i: SQL with An Introduction to PL/SQL, *560 Pages*	Morris	550.00
0766838501	Using AutoCAD 2002 **(B / CD-ROM)**, *1,110 Pages*	Autodesk Press	200.00
0538689978	Web Page Design (All Pages in 4 Color)	Stubbs	250.00

Shroff / Wrox

(Now an imprint of John Wiley & Sons)

PUBLISHED TITLES

8173661146	ASP .NET Website Programming, *562 Pages*	Bellinaso	400.00
8173666938	ASP .NET Website Programming: Problem-Design-Solution: VB .NET Edition, *558 Pages*	Bellinaso	500.00
8173661499	Beginning Active Server Pages 3.0, *1,224 Pages*	Buser	600.00
8173664781	Beginning ASP .NET 1.0 Databases using VB .NET, *482 Pages*	Kauffman	375.00
8173664773	Beginning ASP .NET 1.0 with C#, *878 Pages*	Goode	400.00
8173662258	Beginning ASP .NET 1.0 with VB .NET, *814 Pages*	Goode	400.00
8173665559	Beginning C#, 2/ed *936 Pages*	Watson	650.00
8173666423	Beginning Dynamic Websites with ASP NET Web Matrix **(B / CD-ROM)**, *574 Pages*	Sussman	575.00
8173665591	Beginning Java 2 SDK 1.4 Edition, *1,188 Pages*	Horton	450.00
8173661618	Beginning JavaScript, *1,052 Pages*	Wilton	300.00
8173661561	Beginning Linux Programming 2/ed, *988 Pages*	Matthew	425.00
8173665052	Beginning VB .NET, 2/ed, *886 Pages*	Blair	350.00
8173660409	Beginning Visual C++ 6, *1,216 Pages*	Horton	600.00
8173666040	Expert One-on-One J2EE Design and Development, *768 Pages*	Johnson	650.00
8173665990	Professional Apache Tomcat, *556 Pages*	Chopra	500.00
8173664331	Professional ASP .NET 1.0 - Special Edition, *1,392 Pages*	Anderson	500.00
8173664528	Professional C#, 2/ed, *1,262 Pages*	Robinson	500.00
8173665966	Professional Crystal Reports for Visual Studio .NET, *348 Pages*	McAmis	275.00
8173666334	Professional IBM WebSphere 5.0 Application Server, **(B / 2 CD-ROMS)**, *784 Pages*	Francis	650.00
8173662053	Professional SQL Server 2000 DTS (Data Transformation Services), *888 Pages*	Chaffin	625.00
8173662061	Professional SQL Server 2000 Programming, *1,428 Pages*	Vieira	850.00
8173666083	Professional UML with Visual Studio .NET, *364 Pages*	Filev	350.00
8173664722	Professional VB .NET, 2/ed, *1,020 Pages*	Barwell	450.00
817366224X	VBScript Programmer's Reference, *844 Pages*	Clark	350.00
8173662223	XSLT Programmer's Reference, 2/ed, *984 Pages*	Kay	275.00

Other Computer Titles

8173660034	AS/400 Architecture & Applications, *332 Pages*	Lawrence	250.00
8173660042	AS/400: A Practical Guide to Programming & Operations, *284 Pages*	Zeilenga	225.00
8173660018	CICS: A How-To for Cobol Programmers, *428 Pages*	Kirk	300.00
0130851353	Enterprise Application Integration with XML & Java **(B / CD-ROM)**, *469 Pages*	Morgenthal	450.00
8173660026	MVS / VSAM for the Application Programmer, *504 Pages*	Brown	325.00
8173660077	TCP/IP Companion: A Guide For The Common User, *284 Pages*	Arick	225.00
8173660425	Vijay Mukhi's ERP Odyssey: Implementing. PeopleSoft Financials 7.0/7.5, *528 Pages*	Mukhi	350.00

V.K. Publishers

8190133152	Database Programming Using VB .NET & SQL Server 2000, *472 Pages*	Krishna	350.00
8190133101	Develop An Accounting Package Using Visual Basic & ADO, *372 Pages*	Krishna	350.00
8190133128	Develop An Inventory Management Application Using Microsoft Visual Basic, *374 Pages*	Krishna	345.00

Dental / Health / Medical

8173669791	**The Balancing Act "A Win Over Obesity", *296 Pages***	**Dr. Gadkari**	**225.00**
8173660980	Notes on Pathology, 11th Millennium Edition, *482 Pages*	Experienced Prof.	350.00

1899066934	Menorrhagia, *376 Pages* (Hardbound)	Sheth & Sutton	1,600.00
8173668973	Splinting Management of MOBILE & Migrating TEETH, *104 Pages*	Dr. Kakar	150.00

Economics

8173665435	Economic Environment of Business: A Casestudy Approach, *176 Pages* (PB)	Singh	125.00
0324072910	Economics: A Contemporary Introduction, 6/ed **(B / CD-ROM)**, *809 Pages*	McEachern	1,700.00
0030343984	Economics: Private & Public Choice, 10/ed **(B / CD-ROM)**, *825 Pages*	Gwartney	1,800.00
0324151829	Macroeconomics Principles & Applications, Updated 2/ed, **(B / CD-ROM)**, *494 Pages*	Hall	1,350.00
0324151837	Microeconomics: Principles & Applications, Updated 2/ed **(B / CD-ROM)**, *578 Pages*	Hall	1,350.00

Electrical Engineering

8173667853	**Basic Electrical Circuits, *408 Pages***	**Dr. Salam**	**250.00**

Electronics & Communication

8173666229	Analog & Digital Communication Systems **(B / CD-ROM)**, 5/ed, *588 Pgs*	Roden	300.00
8175980257	Basic Synchros & Servomechanism Parts 1 & 2, *250 Pages*	Valkenburg	150.00
8173669007	**Electronic Components & Materials, 3/ed *404 Pages***	**Dr.Joshi**	**175.00**
8173664358	Electronic Design **(B / CD-ROM)**, 4/ed, *1,002 Pages*	Roden	450.00

English

8173668728	Vocabulary is Important: Exercise Book, *44 Pages*	Strauss	35.00

Environmental Engineering

8173663777	**Principles of Environmental Science, Engineering and Management, *288 Pages***	**Dr.Thirumurthy**	**175.00**

General Titles

Frog Books

8188811203	**Andrearth, *220 Pages***	**Mayor**	**245.00**
8188811068	**Beyond the Call of Voice, *200 Pages***	**Shanker**	**245.00**
818881122X	**Bright Lights Big Buddha, *64 Pages***	**Rao**	**95.00**
8188811815	**CrossRoad, *76 Pages***	**Chhabra**	**150.00**
8188811092	**Do Not Weep, Lonely Mirror, *56 Pages***	**Agarwal**	**60.00**
8188811130	**Echoes of OM, *168 Pages***	**Singh**	**250.00**
8188811106	**Empressions, *116 Pages***	**Nagarajan**	**145.00**
8188811157	**The Green Dragon, *70 Pages***	**Gupta**	**95.00**
818881119X	**The Highway, *176 Pages***	**Marks**	**150.00**
8188811122	**How to be Exclusive, *116 Pages***	**Khisty Ph.D**	**195.00**
8188811114	**I Have Read That Somewhere, *148 Pages***	**Venkataraman**	**245.00**
8188811173	**In the Pink of Wealth, *168 Pages***	**Ghosal**	**180.00**
8188811084	**Much Travelled Yarns, *60***	**Mukherjee**	**95.00**
8188811033	**A New Friend, *24 Pages***	**Blanco**	**30.00**
8188811807	**People Call Me Charlie, *260 Pages***	**Jha**	**295.00**
8188811823	**The Rape of News - I, *32 Pages***	**Poolani**	**30.00**
8188811017	**The Rape of News - I I, *62 Pages***	**Poolani**	**60.00**
818881105X	**Return to La Paz, *210 Pages***	**Reissmann**	**245.00**
8188811041	**Under A Quick Silver Moon, *68 Pages***	**Warrier**	**100.00**
8188811025	**Urban Voice, *92 Pages***	**Poolani**	**150.00**
8188811181	**Where Doves Fly, *70 Pages***	**Rajesham**	**60.00**
8188811165	**Winter Whispers, *44 Pages***	**Sharma**	**60.00**

Embassy Books

8188452238	**Are You Fired Up?, *180 Pages***	**Whiting**	**175.00**

8188452009	**The Art of Dealing with People, *57 Pages***	**Giblin**	**80.00**
8188452130	**The Art of Living, *171 Pages***	**Goenka**	**125.00**
8188452157	**Attitudes and Altitudes, *204 Pages***	**Mesiti**	**175.00**
1564762545	**Be A People Person, *156 Pages***	**John**	**175.00**
8188452106	**Get A Grip on your Dream, *144 Pages***	**Jeff**	**175.00**
8188452114	**If They Say No Just Say Next, *168 Pages***	**Smith**	**175.00**
8188452076	**It Only Takes Everything You've Got, *128 Pages***	**Melara**	**175.00**
8188455228	**Nine Essential Laws of Becoming Influential, *111 Pages***	**Zeiss**	**150.00**
8188452149	**No Excuse, I Am Doing It, *197 Pages***	**Refinbary**	**175.00**
8188452084	**Opportunity Knocks, *100 Pages***	**Mesiti**	**125.00**
8188452297	**The Power of 2, *164 Pages***	**Hottinger**	**175.00**
8186452300	**Reject Me I Love It, *165 Pages***	**Fuhrman**	**175.00**
8188452165	**Skill With People, *47 Pages***	**Giblin**	**50.00**
8188452262	**Soar To the Top, *143 Pages***	**Anderson**	**175.00**
8188452041	**Ultimate Gift, *128 Pages***	**Stovall**	**150.00**

Geopraphic Information System (GIS) / Global Positioning System (GPS)

3540426434	Commercial Satellite Imagery: A Tactic in Nuclear Weapon Deterrence, *346 Pages*	Jasani	€ 119.00
3540426485	GIS Processing of Geocoded Satellite Data, *352 Pages*	Williams	€ 174.95
3211835342	GPS Theory and Practice, 5th revised edition, *408 Pages*	Hofmann	€ 51.00
3540201300	ISO Standards for Geographic Informations, *338 Pages*	Kresse	€ 89.95
3211008284	Navigation Principles of Positioning & Guidance, *464 Pages*	Hofmann	€ 54.00
354067280X	Satellite Orbits Model, Methods Applications **(B / CD-ROM)**, *386 Pages*	Montenbruck	€ 64.95
3540668403	Satellite Systems for Personal & Broadband Communications , *454 Pages*	Lutz	€ 99.95
3540003282	Weather Radar Principles & Advanced Applications, *362 Pages*	Meischner	€ 79.95

Law

8173664153	Customs Valuation in India 3/ed, *262 Pages*	Satapathy	375.00
8173661421	Law of Sale of Goods & Partnership, *228 Pages*	Chandiramani	150.00

Marine

8173665516	A Textbook on Container & Multimodal Transport Management, *522 Pages*	Dr. Hariharan	500.00
8175980176	Annex VI of MARPOL 73/78 Regulations for the Prevention of Air Pollution from Ships & Nox Technical Code, *156 Pages*	IMO	120.00
8175980273	BC CODE Code of Safe Practise for Solid Bulk Cargoes, 2001 edition, *224 Pages*	IMO	150.00
8175980354	Colreg Consoiidated Edtion 2002, *56 Pages*	IMO	95.00
8173667047	Containerisation & Multimodal Transport in India, 4/ed, *586 Pages*	Hariharan	500.00
8175980346	FSS Code International Code for Fire Safety Systems, *272 Pages*	IMO	150.00
0750600063	General Engineering Knowledge: Marine Engineering Series, 3/ed, *166 Pages*	McGeorge	375.00
8181475143	**Guide to the Collision Avoidance Rules, 6/ed, *260 Pages* (New Edition)**	**Cockcroft**	**400.00**
1843111950	Handbook of Marinetimes Economics & Business, *954 Pages*	Grammenos	5,300.00
8175980281	ISM CODE International Safety Management Code, 2002 Edition, *46 Pgs*	IMO	95.00
8175980370	ISPS Code, 2003 Edition, *156 Pages*	IMO	100.00
8175980362	Life-Saving Appliances, 2003 Edition, *200 Pages*	IMO	150.00
0750625309	Introduction to Marine Engineering, 2/ed Revised Ed, *382 Pages*	Taylor	400.00
0750625295	Introduction to Naval Architecture, 3/ed Revised Ed, *372 Pages*	Tupper	400.00
8181478037	**Lamb's Questions & Answers on the Marine Diesel Engine, 8/ed, *544 Pages***	**Christensen**	**500.00**
8173660379	M.S.(STCW) Rules, 1998 incl. Training & Assessment Programme - 1, *216 Pages*	DG Shipping	225.00
0117724629	Manual of Seamanship (Consolidated Edition), *830 Pages*	Admiralty	1,850.00
8181473329	Marine Auxiliary Machinery, 7/ed, *524 Pages*	McGeorge	500.00
8181479688	**Marine Boilers 3/ed, *212 Pages***	**Flanagan**	**375.00**
8173669279	**Marine Diesel Engines, *428 Pages***	**Aranha**	**250.00**

8173660808	Marine Internal Combustion Engines, *272 Pages*	Kane	150.00
8173663106	**Marine Electrical Technology, 2/ed Pages (New Edition)**	**Fernandes**	**650.00**
0415153107	Maritime Economics, 2/ed, *592 Pages*	Stopford	1,350.00
8173661413	Maritime Education, Training & Assessment Manual (TAP) - Vol II, *474 Pages*,	DG Shipping	400.00
9280151258	MARPOL 73/78: Consolidated Edition 2002, 2/ed, *528 Pages*	IMO	300.00
8175980389	**Merchant Shipping Act, 1958, *268 Pages***	**Sterling**	**150.00**
3211008284	Navigation Principles of Positioning & Guidance, *464 Pages*	Hofmann	€ 54.00
8175980141	Procedures for Port State Control (IMO No. 650E), *104 Pages*	IMO	55.00
8181475151	**Pounder's Marine Diesel Engines & Gas Turbines, 8/ed, *914 Pages* (New Edition)**	**Woodyard**	**995.00**
817366014X	Safety Management Systems - An Underconstruction Activity, *98 Pages*	Capt. Singhal	95.00
8181478096	**Seamanship Techniques Shipboard and Marine Operations, 3/ed, *742 Pages***	**House**	**800.00**
_____	Ship Magnetism & the Magnetic Compass, *246 Pages*	Merrifield	200.00
8181475194	**Ship Construction, 5/ed, 364 Pages (New Edition)**	**Eyres**	**450.00**
8181470044	Ship Construction Sketches & Notes, *144 Pages*	Kemps	375.00
8181478118	**Ship Stability for Masters and Mates, 5/ed, *462 Pages***	**Capt. Derret**	**475.00**
8175980109	Shipping Practice, 11/ed, *186 Pages*	Stevens	150.00
8175980443	**SOLAS Consolidated Edition 2004, *576 Pages* (New Edition)**	**IMO**	**300.00**
8175980397	SOLAS Amendments 2001 and 2002, *56 Pages*	IMO	75.00
0415153727	Theory & Practice of Seamanship, 11/ed, *538 Pages*	Danton	450.00
3540003282	Weather Radar Principles & Advanced Applications, *362 Pages*	Meischner	€ 79.95

Motivation

8173665273	**Heads You Win, Tails You Win, 2/ed, *200 Pages***	**Dr.Mishra**	**200.00**

Physics

9812435158	Classical Dynamics of Particles and Systems, 5/ed *672 Pages*	Thornton	425.00

Project Management

8173668663	Advanced Project Portfolio Management and the PMO: Multiplying ROI at Warp Speed, *452 Pages*	Kendall	425.00
8173667837	**The Art of Project Management, *512 Pages***	**Berkun**	**350.00**
8173668140	PMP - Project Management Professional Workbook, *303 Pages*	Baca	200.00
8173668671	Quantative Methods in Project Management, *292 Pages*	Goodpasture	300.00

Statistics

981243514X	Analyzing Multivariate Data, *580 Pages*	Lattin	400.00

- **Dates & Prices of forthcoming titles are tentative and subject to change without notice.**
- **All Prices are in Indian Rupees except where indicated in € (Euro Dollar) $ (US Dollar) and £ (Pound)**
- **TITLES RELEASED AFTER FEBRUARY 2005 ARE MARKED IN BOLD.**

For Wholesale enquiries contact:-

SHROFF PUBLISHERS & DISTRIBUTORS PVT. LTD. SPD

C-103, TTC Industrial Area, MIDC, Pawane, Navi Mumbai - 400 701.
Tel: (91 22) 2763 4290 • Fax: (91 22) 2768 3337
E-mail: spdorders@shroffpublishers.com

Branches:-

Kolkata
7B Haati Bagan Road, 1st Floor,
Kolkata 700 014
Tel: (91 33) 2284 9329, 2284 7954
Fax: (91 33) 2240 6109
E-mail: spdcal@vsnl.com

Delhi
Basement, 2/11 Ansari Road, Daryaganj
New Delhi - 110 002
Tel: (91 11) 2324 3337 / 8
Fax: (91 11) 2324 3339
E-mail: spddel@shroffpublishers.com

Chennai
23, Damodharan Street, (Old No. 12)
T.Nagar, Chennai - 600 017
Tel: (91 44) 2431 5343
Fax: (91 44) 2431 5342
E-mail: spdchennai@shroffpublishers.com

Bangalore
7, Sharada Colony, Basaveshwamagar,
8th Main, Bangalore 560 079
Tel: (91 80) 5128 7393
Fax: (91 80) 5128 7392
E-mail: spdblr@shroffpublishers.com

Hyderabad Resident Representative
Jayashanker Kuppili
Mobile : 988 50 59050
E-mail: shanker@shroffpublishers.com

For retail enquiries contact:-

COMPUTER BOOKSHOP
www.cb-india.com

Mumbai
Computer Bookshop (India) Pvt. Ltd.
190 Dr. D.N. Road, Fort, Mumbai 400 001
Tel: (91 22) 5631 7922 / 23 / 24
Fax: (91 22) 2262 3551
E-mail: orders@cb-india.com

Kolkata
Franchisee: Millennium
11 Mayfair Road (Opp. Ice Skating Rink),
Kolkata: 700 019 • Tel: (91 33) 2280 0478,
2280 7045 • Fax : (91 33) 2240 6109
E-mail: millennium@vsnl.com

Sterling Book House
181, Dr. D. N. Road, Fort, Mumbai - 400 001.
Tel. : (91-22) 2261 2521, 2265 9599
Fax : 2262 3551 • E-mail : sbh@vsnl.com

BOOKZONE
BOOKSHOP FOR PROFESSIONALS
Shop #1, Cassinath Building, 172/174,
Dr. D. N. Road, Mumbai - 400 001. India.
Tel. : (91-22) 5633 4622
Fax : (91-22) 5633 4624
E-mail : mail@bookzone-india.com