

OSS: (Open Source Software)

Open source software are user friendly and are available as commercial and free licenced software. Therefore, these are easy to use by anyone, anywhere. Open source is easily modifiable as its core code is publicly accessible.

OS (Operating System)

An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system.

Linux operating System

Linux is a free open-source operating system based on Unix. Linux was originally created by Linus Torvalds with the assistance of developers from around the globe. Linux is free to download, edit and distribute. Linux is a very powerful operating system and it is gradually becoming popular throughout the world.

Advantages of Linux:

Low cost: There is no need to spend time and huge amount money to obtain licenses since Linux and much of it's software come with the GNU General Public License. There is no need to worry about any software's that you use in Linux.

Stability: Linux has high stability compared with other operating systems. There is no need to reboot the Linux system to maintain performance levels. Rarely it freeze up or slow down. It has a continuous up-times of hundreds of days or more.

Performance: Linux provides high performance on various networks. It has the ability to handle large numbers of users simultaneously.

Networking: Linux provides a strong support for network functionality; client and server systems can be easily set up on any computer running Linux. It can perform tasks like network backup more faster than other operating systems.

Flexibility: Linux is very flexible. Linux can be used for high performance server applications, desktop applications, and embedded systems. You can install only the needed components for a particular use. You can also restrict the use of specific computers.

Compatibility: It runs all common Unix software packages and can process all common file formats.

Wider Choice: There is a large number of Linux distributions which gives you a wider choice. Each organization develop and support different distribution. You can pick the one you like best; the core function's are the same.

Better use of hard disk: Linux uses its resources well enough even when the hard disk is almost full.

Multitasking: Linux is a multitasking operating system. It can handle many things at the same time.

Security: Linux is one of the most secure operating systems. File ownership and permissions make linux more secure.

Open source: Linux is an Open source operating systems. You can easily get the source code for linux and edit it to develop your personal operating system.

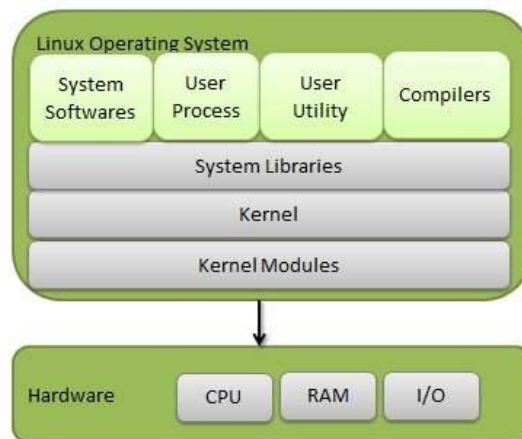
Components of Linux System

Linux Operating System has primarily three components

Kernel – Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.

System Library – System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.

System Utility – System Utility programs are responsible to do specialized, individual level tasks.



Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called kernel mode with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in User Mode which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

Following are some of the important features of Linux Operating System.

Portable – Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.

Open Source – Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

Multi-User – Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.

Multiprogramming – Linux is a multiprogramming system means multiple applications can run at same time.

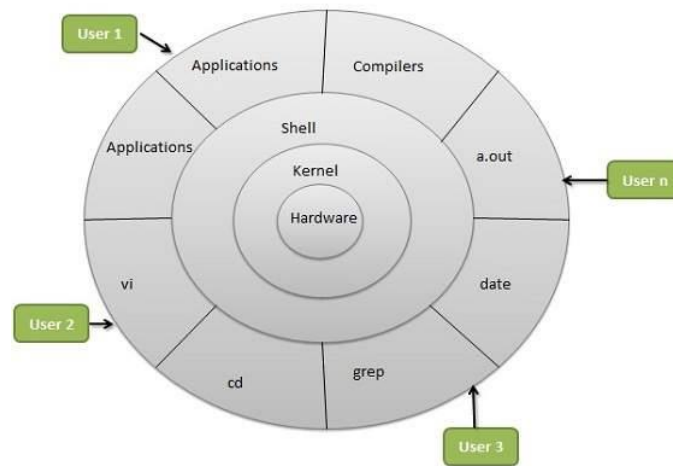
Hierarchical File System – Linux provides a standard file structure in which system files/ user files are arranged.

Shell – Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.

Security – Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Architecture

The following illustration shows the architecture of a Linux system –



The architecture of a Linux System consists of the following layers

Hardware layer – Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).

Kernel – It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.

Shell – An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.

Utilities – Utility programs that provide the user most of the functionalities of an operating systems.

Kernel Mode

In Kernel mode, the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

User Mode

In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

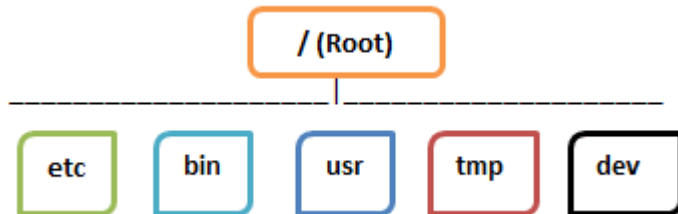
Windows Vs. Linux File System

In Microsoft Windows, files are stored in folders on different data drives like C: D: E:

But, in **Linux, files are ordered in a tree structure starting with the root directory.**

This root directory can be considered as the start of the file system, and it further branches out various other subdirectories. The root is denoted with a forward slash '/'.

A general tree file system on your UNIX may look like this.



Types of Files

In Linux and UNIX, everything is a file. Directories are files, files are files, and devices like Printer, mouse, keyboard etc. are files.

Let's look into the File types in more detail.

General Files

General Files also called as Ordinary files. They can contain image, video, program or simply text. They can be in ASCII or a Binary format. These are the most commonly used files by Linux Users.

Directory Files

These files are a warehouse for other file types. You can have a directory file within a directory (sub-directory). You can take them as 'Folders' found in Windows operating system.

Device Files:

In MS Windows, devices like Printers, CD-ROM, and hard drives are represented as drive letters like G: H:. In Linux, they are represented as files. For example, if the first SATA hard drive had

three primary partitions, they would be named and numbered as `/dev/sda1`, `/dev/sda2` and `/dev/sda3`.

Note: All device files reside in the directory `/dev/`

All the above file types (including devices) have permissions, which allow a user to read, edit or execute (run) them. This is a powerful Linux/Unix feature. Access restrictions can be applied for different kinds of users, by changing permissions.

Windows Vs. Linux: Users

There are 3 types of users in Linux.

1. Regular
2. Administrative(root)
3. Service

Regular User

A regular user account is created for you when you install Ubuntu on your system. All your files and folders are stored in `/home/` which is your home directory. As a regular user, you do not have access to directories of other users.

Root User

Other than your regular account another user account called root is created at the time of installation. The root account is a **superuser** who can access restricted files, install software and has administrative privileges. Whenever you want to install software, make changes to system files or perform any administrative task on Linux; you need to log in as a root user. Otherwise, for general tasks like playing music and browsing the internet, you can use your regular account.

Service user

Linux is widely used as a Server Operating System. Services such as Apache, Squid, email, etc. have their own individual service accounts. Having service accounts increases the security of your computer. Linux can allow or deny access to various resources depending on the service.

In Windows, there are 4 types of user account types.

1. Administrator
2. Standard
3. Child

4. Guest

Windows Vs. Linux: File Name Convention

In Windows, you cannot have 2 files with the same name in the same folder. While in Linux, you can have 2 files with the same name in the same directory, provided they use different cases.

Windows Vs. Linux: HOME Directory

For every user in Linux, a directory is created as **/home/**

Consider, a regular user account "Tom". He can store his personal files and directories in the directory "/home/tom". He can't save files outside his user directory and does not have access to directories of other users. For instance, he cannot access directory "/home/jerry" of another user account "Jerry".

The concept is similar to C:\Documents and Settings in Windows.

When you boot the Linux operating system, your user directory (from the above example /home/tom) is the **default working directory**. Hence the directory "/home/tom is also called the **Home directory** which is a misnomer.

Windows Vs. Linux: Key Differences

Windows	Linux
Windows uses different data drives like C: D: E to stored files and folders.	Unix/Linux uses a tree like a hierarchical file system.
Windows has different drives like C: D: E	There are no drives in Linux
Hard drives, CD-ROMs, printers are considered as devices	Peripherals like hard drives, CD-ROMs, printers are also considered files in Linux/Unix
There are 4 types of user account types 1) Administrator, 2) Standard, 3) Child, 4) Guest	There are 3 types of user account types 1) Regular, 2) Root and 3) Service Account
Administrator user has all administrative privileges of computers.	Root user is the super user and has all administrative privileges.
In Windows, you cannot have 2 files with the same name in the same folder	Linux file naming convention is case sensitive. Thus, sample and SAMPLE are 2 different files in Linux/Unix operating system.
In windows, My Documents is default home directory.	For every user /home/username directory is created which is called his home directory.

Linux Command Line Tutorial: Manipulate Terminal with CD Commands

The most frequent tasks that you perform on your PC is creating, moving or deleting Files. Let's look at various options for File Management.

To manage your files, you can either use

1. Terminal (Command Line Interface - CLI)
2. File manager (Graphical User Interface -GUI)

Why learn Command Line Interface?

Even though the world is moving to GUI based systems, CLI has its specific uses and is widely used in scripting and server administration. Let's look at it some compelling uses -

- Comparatively, Commands offer more options & are flexible. Piping and stdin/stdout are immensely powerful are not available in GUI
- Some configurations in GUI are up to 5 screens deep while in a CLI it's just a single command
- Moving, renaming 1000's of the file in GUI will be time-consuming (Using Control /Shift to select multiple files), while in CLI, using regular expressions so can do the same task with a single command.
- CLI load fast and do not consume RAM compared to GUI. In crunch scenarios this matters.

Both GUI and CLI have their specific uses. For example, **in GUI, performance monitoring graphs** give **instant visual feedback** on system health, while seeing hundreds of lines of logs in CLI is an eyesore.

Files and directory related commands

Command	Meaning
<code>cd <i>directory</i></code>	change to named directory
<code>cd</code>	change to home-directory
<code>cd ~</code>	change to home-directory
<code>cd ..</code>	change to parent directory
<code>pwd</code>	display the path of the current director

Command	Description
<code>ls</code>	Lists all files and directories in the present working directory
<code>ls -R</code>	Lists files in sub-directories as well

ls - a	Lists hidden files as well
ls - al	Lists files and directories with detailed information like permissions, size, owner, etc.
cat > filename	Creates a new file
cat filename	Displays the file content
cat file file2 > file3	Joins two files (file1, file2) and stores the output in a new file (file3)
mv file "new file path"	Moves the files to the new location
mv filename new_file_name	Renames the file to a new filename
sudo	Allows regular users to run programs with the security privileges of the superuser or root
rm	Deletes a file
man	Gives help information on a command
history	Gives a list of all past commands typed in the current terminal session
clear	Clears the terminal
mkdir	Creates a new directory in the present working directory
mkdir	Create a new directory at the specified path
rmdir	Deletes a directory
mv	Renames a directory
pr -x	Divides the file into x columns
pr -h	Assigns a header to the file
pr -n	Denotes the file with Line Numbers
lp -nc lpr c	Prints "c" copies of the File

lp -d lp -P	Specifies name of the printer
apt-get	Command used to install and update packages
mail -s 'subject' -c 'cc-address' -b 'bcc-address' 'to-address'	Command to send email

Command	Meaning
<code>less file</code>	display a file a page at a time
<code>head file</code>	display the first few lines of a file
<code>tail file</code>	display the last few lines of a file
<code>grep 'keyword' file</code>	search a file for keywords
<code>wc file</code>	count number of lines/words/characters in file

Command	Meaning
<code>command > file</code>	redirect standard output to a file
<code>command >> file</code>	append standard output to a file
<code>command < file</code>	redirect standard input from a file
<code>command1 command2</code>	pipe the output of command1 to the input of command2
<code>cat file1 file2 > file0</code>	concatenate file1 and file2 to file0
<code>sort</code>	sort data
<code>who</code>	list users currently logged in

File Permissions in Linux/Unix with Example

Linux is a clone of UNIX, the **multi-user operating system** which can be accessed by many users simultaneously. Linux can also be used in mainframes and servers without any modifications. But this raises security concerns as an unsolicited or **malign user** can **corrupt, change or remove crucial data**. For effective security, Linux divides authorization into 2 levels.

1. Ownership
2. Permission

The concept of **permissions** and **ownership** is crucial in Linux. Here, we will discuss both of them. Let us start with the **Ownership**.

Ownership of Linux files

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below.

User

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

Group

A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

Other

Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your colleague, who works on your Linux computer, to view your images. This is where **Permissions** set in, and they define **user behavior**.

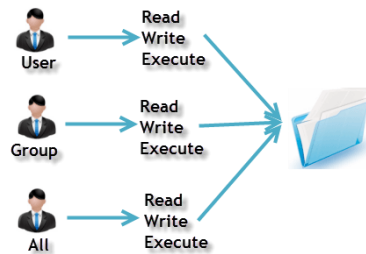
Let us understand the **Permission system** on Linux.

Permissions

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

- **Read:** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.
- **Write:** The right permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.
- **Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

Owners assigned Permission On Every File and Directory



Let's see this in action

ls - l on terminal gives

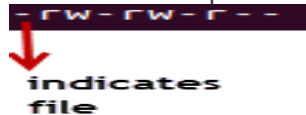
ls - l

File type and Access Permissions.

```
home@VirtualBox: ~  
home@VirtualBox:~$ ls -l  
-rw-rw-r-- 1 home home 0 2012-08-30 19:06 My File
```

Here, we have highlighted '-rw-rw-r--' and this weird looking code is the one that tells us about the permissions given to the owner, user group and the world.

Here, the first '-' implies that we have selected a file.p>



Else, if it were a directory, d would have been shown.

d represents directory

```
drwxr-xr-x 2 ubuntu ubuntu 80 Sep 6 07:27 Desktop
```

The characters are pretty easy to remember.

- r = read permission
- w = write permission
- x = execute permission
- = no permission

Let us look at it this way.

The first part of the code is 'rw-'. This suggests that the owner 'Home' can:



- Read the file
- Write or edit the file
- He cannot execute the file since the execute bit is set to '-'.

By design, many Linux distributions like Fedora, CentOS, Ubuntu, etc. will add users to a group of the same group name as the user name. Thus, a user 'tom' is added to a group named 'tom'.

The second part is 'rw-'. It for the user group 'Home' and group-members can:

- Read the file
- Write or edit the file

The third part is for the world which means any user. It says 'r--'. This means the user can only:

- Read the file



Changing file/directory permissions with 'chmod' command

Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions.

We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world. **Syntax:**

```
chmod permissions filename
```

There are 2 ways to use the command -

1. **Absolute mode**
2. **Symbolic mode**

Absolute(Numeric) Mode

In this mode, file **permissions are not represented as characters but a three-digit octal number**.

The table below gives numbers for all for permissions types.

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--
5	Read + Execute	r-x

6	Read +Write	rw-
7	Read + Write +Execute	rwx

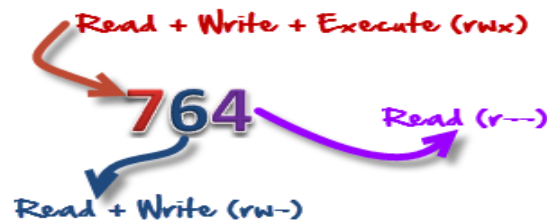
Let's see the chmod command in action.

```

Checking Current File Permissions
ubuntu@ubuntu:~$ ls -l sample
-rw-rw-r-- 1 ubuntu ubuntu 15 Sep  6 08:00 sample

chmod 764 and checking permissions again
ubuntu@ubuntu:~$ chmod 764 sample
ubuntu@ubuntu:~$ ls -l sample
-rwxrw-r-- 1 ubuntu ubuntu 15 Sep  6 08:00 sample
  
```

In the above-given terminal window, we have changed the permissions of the file 'sample' to '764'.



'764' absolute code says the following:

- Owner can read, write and execute
- Usergroup can read and write
- World can only read

This is shown as '-rwxrw-r-

This is how you can change the permissions on file by assigning an absolute number.

Symbolic Mode

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

Operator	Description
+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as -

User Denotations	
u	user/owner
g	group
o	other
a	all

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example

```

Current File Permissions
home@VirtualBox:~$ ls -l sample
-rw-rw-r-- 1 home home 55 2012-09-10 10:59 sample

Setting permissions to the 'other' users
home@VirtualBox:~$ chmod o=rwx sample
home@VirtualBox:~$ ls -l sample
-rw-rw-rwx 1 home home 55 2012-09-10 10:59 sample

Adding 'execute' permission to the user group
home@VirtualBox:~$ chmod g+x sample
home@VirtualBox:~$ ls -l sample
-rw-rwxrwx 1 home home 55 2012-09-10 10:59 sample

Removing 'read' permission for 'user'
home@VirtualBox:~$ chmod u-r sample
home@VirtualBox:~$ ls -l sample
-w-rwxrwx 1 home home 55 2012-09-10 10:59 sample

```

Vi Editor:

Vi is a command line text editor. As you would be quite aware now, the command line is quite a different environment to your GUI. It's a single window with text input and output only. Vi has been designed to work within these limitations and many would argue, is actually quite powerful as a result. Vi is intended as a plain text editor (similar to Notepad on Windows, or Textedit on Mac) as opposed to a word processing suite such as Word or Pages. It does, however have a lot more power compared to Notepad or Textedit.

As a result you have to ditch the mouse. Everything in Vi is done via the keyboard. There are two modes in Vi. **Insert** (or Input) mode and **Edit** mode. In input mode you may input or enter content into the file. In edit mode you can move around the file, perform actions such as deleting, copying, search and replace, saving etc. A common mistake is to start entering commands without first going back into edit mode or to start typing input without first going into insert mode. If you do either of these it is generally easy to recover so don't worry too much.

When we run vi we normally issue it with a single command line argument which is the file you would like to edit

> vi file name

PROCESSES:

A **process** refers to a program in execution; it's a running instance of a program. It is made up of the program instruction, data read from files, other programs or input from a system user.

Types of Processes

init

It is the first process executed by the kernel during the booting of a system. It is a daemon process which runs till the system is shutdown. That is why, it is the parent of all the processes. First of all, **init** reads the script stored in the file **/etc/inittab**. Command **init** reads the initial configuration script which basically take care of everything that a system do at the time of system initialization like setting the clock, initializing the serial port and so on.

Foreground processes (also referred to as interactive processes) – these are initialized and controlled through a terminal session. In other words, there has to be a user connected to the system to start such processes; they haven't started automatically as part of the system functions/services.

Background processes (also referred to as non-interactive/automatic processes) – are processes not connected to a terminal; they don't expect any user input.

Batch process

Processes which are in queue or stack executes one by one in FIFO order are called batch process.

Daemons

These are special types of background processes that start at system startup and keep running forever as a service; they don't die. They are started as system tasks (run as services), spontaneously. However, they can be controlled by a user via the init process.

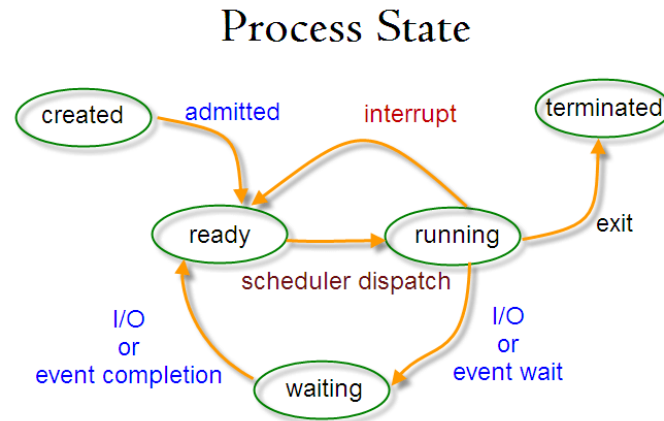
What is a zombie process?

When a process finishes execution, it will have an exit status to report to its parent process. Because of this last little bit of information, the process will remain in the operating system's process table as a *zombie process*, indicating that it is not to be scheduled for further execution, but that it cannot be completely removed (and its process ID cannot be reused) until it has been determined that the exit status is no longer needed.

When a child exits, the parent process will receive a SIGCHLD signal to indicate that one of its children has finished executing; the parent process will typically call the wait() system call at this point. That call will provide the parent with the child's

exit status, and will cause the child to be *reaped*, or removed from the process table.

Process States:



As a process executes it changes *state* according to its circumstances. Linux processes have the following states:

Running

The process is either running (it is the current process in the system) or it is ready to run (it is waiting to be assigned to one of the system's CPUs).

Waiting

The process is waiting for an event or for a resource. Linux differentiates between two types of waiting process; *interruptible* and *uninterruptible*. Interruptible waiting processes can be interrupted by signals whereas uninterruptible waiting processes are waiting directly on hardware conditions and cannot be interrupted under any circumstances.

Stopped

The process has been stopped, usually by receiving a signal. A process that is being debugged can be in a stopped state.

Zombie

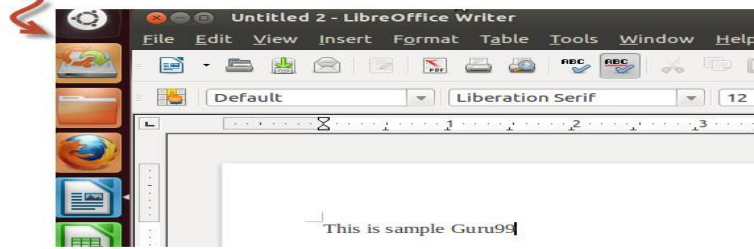
This is a halted process which, for some reason, still has a `task_struct` data structure in the `task` vector. It is what it sounds like, a dead process.

Linux/Unix Process Management: `ps`, `kill`, `top`, `df`, `free`, `nice` Commands

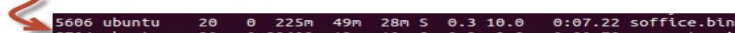
What is a Process?

An instance of a program is called a Process. In simple terms, any command that you give to your Linux machine starts a new process.

When you launch Office to write some article



Corresponding process is created



Running a Foreground Process

To start a foreground process, you can either run it from the dashboard, or you can run it from the terminal.

When using the Terminal, you will have to wait, until the foreground process runs.



OR

```
home@VirtualBox:~$ banshee
```

Running a Background process

If you start a foreground program/process from the terminal, then you cannot work on the terminal, till the program is up and running.

Particular, data-intensive tasks take lots of processing power and may even take hours to complete. You do not want your terminal to be held up for such a long time.

To avoid such a situation, you can run the program and send it to the background so that terminal remains available to you. Let's learn how to do this -

Start the program and press ctrl+z

```
guru99@VirtualBox:~$ banshee
[Info 16:08:36.688] Running Banshee 2.2.1: [Ubuntu 11.
11-12-19 14:51:26 UTC]
^Z
[1]+  Stopped                  banshee
```

Type 'bg' to send the process to the background

```
guru99@VirtualBox:~$ bg
```

Fg

You can use the command "fg" to continue a program which was stopped and bring it to the foreground.

The simple syntax for this utility is:

```
fg jobname
```

Example

1. Launch 'banshee' music player
2. Stop it with the 'ctrl +z' command
3. Continue it with the 'fg' utility.

```
home@VirtualBox:~$ banshee
^Z
[1]+  Stopped                  banshee
home@VirtualBox:~$ fg banshee
banshee
[Info 00:36:19.400] Running Banshee 2.2.0: [Ubuntu oneiric
(linux-gnu, i686) @ 2011-09-23 04:51:00 UTC]
```

Let's look at other important commands to manage processes -

Top

This utility tells the user about all the running processes on the Linux machine.

```
home@VirtualBox:~$ top
top - 23:57:43 up 2:54, 1 user, load average: 0.00, 0.01, 0.05
Tasks: 189 total, 2 running, 187 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.7%us, 3.0%sy, 0.0%ni, 96.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1026080k total, 924508k used, 101572k free, 37000k buffers
Swap: 1046524k total, 21472k used, 1025052k free, 367996k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 1525 home       20   0 1775m 100m 28m  S   1.7  10.0   5:05.34  Photoshop.exe
  961 root        20   0 75972  51m 7952  R   1.0   5.1   2:23.42  Xorg
 1507 home       20   0  7644  4652  696  S   1.0   0.5   2:42.66  wineserver
 1564 home       20   0 75144   29m 9840  S   0.3   3.0   0:25.96  ubuntuone-syncd
 2999 home       20   0  127m   13m 10m  S   0.3   1.4   0:01.36  gnome-terminal
 3077 home       20   0  2820  1188  864  R   0.3   0.1   0:00.76  top
    1 root        20   0   3200  1704 1260  S   0.0   0.2   0:00.98  init
    2 root        20   0     0     0     0  S   0.0   0.0   0:00.00  kthreadd
    3 root        20   0     0     0     0  S   0.0   0.0   0:00.95  ksoftirqd/0
```

Press 'q' on the keyboard to move out of the process display.

The terminology follows:

Field	Description	Example 1	Example 2
PID	The process ID of each task	1525	961
User	The username of task owner	Home	Root

PR	Priority Can be 20(highest) or -20(lowest)	20	20
NI	The nice value of a task	0	0
VIRT	Virtual memory used (kb)	1775	75972
RES	Physical memory used (kb)	100	51
SHR	Shared memory used (kb)	28	7952
S	Status	S	R
%CPU	% of CPU time	1.7	1.0
%MEM	Physical memory used	10	5.1
TIME+	Total CPU time	5:05.34	2:23.42
Command	Command name	Photoshop.exe	Xorg

PS

This command stands for 'Process Status'. It is similar to the "Task Manager" that pop-ups in a Windows Machine when we use Cntrl+Alt+Del. This command is similar to 'top' command but the information displayed is different.

To check all the processes running under a user, use the command -

ps ux

```
home@VirtualBox:~$ ps ux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
home     1114  0.0  0.8 46548  8512 ?        Ssl   Sep03   0:00 gnome-sess
home     1151  0.0  0.0  3856   140 ?        Ss    Sep03   0:00 /usr/bin/s
home     1154  0.0  0.0  3748   484 ?        S     Sep03   0:00 /usr/bin/d
home     1155  0.1  0.2  6656  3036 ?        Ss    Sep03   0:18 //bin/dbus
home     1157  0.0  0.2  9148  2368 ?        S     Sep03   0:00 /usr/lib/g
home     1162  0.0  0.2 31588  2296 ?        Ssl   Sep03   0:00 /usr/lib/g
home     1174  0.0  1.4 132472 14884 ?        Sl    Sep03   0:03 /usr/lib/g
```

You can also check the process status of a single process, use the syntax -

ps PID

```
guru99@VirtualBox:~$ ps 1268
  PID TTY      STAT   TIME COMMAND
 1268 ?        S<l    0:02 /usr/bin/pulseaudio --start --log-target=syslog
```

Kill

This command **terminates running processes** on a Linux machine.

To use these utilities you need to know the PID (process id) of the process you want to kill

Syntax –
kill PID

To find the PID of a process simply type

```
pidof Process name
```

Let us try it with an example.

```
home@VirtualBox:~$ pidof Photoshop.exe  
1525  
home@VirtualBox:~$ kill 1525
```

Creation of Process:

The below given program shows how to create a process:

```
#include<errno.h>  
#include<stdio.h>  
#include<unistd.h>  
Main()  
{  
    pid_t pid  
    pid=fork();//this will create a new process  
    if(pid== -1) //if the process is not created  
        perror("fork"); // printing the error message  
    else if(pid==0)  
        printf("this is a child process with pid:%d\n",pid);  
    else  
        printf("this is a parent process with pid:%d\n",pid);  
}
```

Compiling file process.c

```
$gcc process.c
```

Executing the compiled file

```
$/a.out
```

The output generated by this program is:

```
This is a child process with pid: 0
```

```
This is a parent process with pid: 5335
```

Fork() function:

System call **fork()** is used to create processes. It takes no arguments and returns a process ID. The purpose of **fork()** is to create a **new** process, which becomes the *child* process of the caller. After a new child process is created, **both** processes will execute the next instruction following the **fork()** system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of **fork()**:

- If **fork()** returns a negative value, the creation of a child process was unsuccessful.
- **fork()** returns a zero to the newly created child process.
- **fork()** returns a positive value, the **process ID** of the child process, to the parent. The returned process ID is of type **pid_t** defined in **sys/types.h**. Normally, the process ID is an integer. Moreover, a process can use function **getpid()** to retrieve the process ID assigned to this process.
- the child process will have a parent process ID (PPID) which is same as the PID of the process that created it.
- The child process will have a unique process identifier (PID), like all other processes in the system.
- The child process does not have the possibility to inherit any timers from its parent.
- Resource utilization and CPU timers will be reset to zero in the child process.

After the fork function completes its work, there exist two processes. Each process continues its execution from the position, where fork() returns.

Syntax of fork() function:

```
#include<sys/types.h>
```

```
#include<unistd.h>
```

```
Pid_t fork(void);
```

Vfork() function:

It is used for creating new processes without copying the page tables belonging to the parent process. Vfork() is faster than a fork(). It does not make any copy of the parent process address

space, rather it will borrow the parents memory and thread of the control, until a call to exit or `execve()` is occurred. The `execve()` executes the program pointed by specified filename. Whenever a new process is created by using `vfork()`, the parent process is suspended temporarily and the child process will proceed by borrowing the parent process address space. This will be continued, until the child process calls `execve()` or exits at which the parent process will be continued.

Vfork() function syntax:

```
#include<sys/types.h>
#include<unistd.h>
Pid_t vfork();
```

getpid() and getppid() functions:

getpid(): the child process can acquire its own process ID by using `getpid()` function.

getppid(): the child process can acquire its parent process ID by using `getppid()` function.

Wait() function:

A process is said to be in waiting state, if it needs some resource for its execution or some event to occur.

Syntax of `wait()` function:

```
#include<sys/wait.h>
Int wait(status)
```

Example:

```
#include<sys/wait.h>
#include<stdio.h>
#include<sys/wait.h>
Main()
{
Int statuspr,pid,cpid;
Pid=fork();
If(pid!=0)
{
Printf("process id %d",pid);
Cpid=wait(&statuspr);
Printf("child process is in wait state");
}
Else
{
```

```
Printf("process id %d",getpid());
Exit()
}
Printf("pid %d is going to be terminated:",pid);
}
```

Reasons for failure of creating a new process

The following are the possible reasons for the failure of creation of new processes:

- The resource limit on the number of processes permitted to a particular user ID has been exceeded.
- The system wide limit on the number of processes that can be created has been reached.

Scheduling

The scheduler is the component of the kernel that selects which process to run next. The scheduler (or process scheduler, as it is sometimes called) can be viewed as the code that divides the finite resource of processor time between the runnable processes on a system. The scheduler is the basis of a multitasking operating system such as Linux. By deciding what process can run, the scheduler is responsible for best utilizing the system and giving the impression that multiple processes are simultaneously executing.

The idea behind the scheduler is simple. To best utilize processor time, assuming there are runnable processes, a process should always be running. If there are more processes than processors in a system, some processes will not always be running. These processes are waiting to run. Deciding what process runs next, given a set of runnable processes, is a fundamental decision the scheduler must make.

Objectives of scheduling:

Scheduling policy determines when to switch and what process to choose. The following are some of the scheduling objectives:

- To avoid process starvation
- To achieve good throughput for background jobs
- To improve system performance
- To enforce priorities
- To minimise the wasted resources overhead
- To achieve a balance between a response and utilization
- To support for soft real time processes

Processes running for a long time have their priorities decreased, whereas the processes which are waiting for a long time will have their priorities increased dynamically.

Categories of Processes:

I/O bound processes: the processes which use I/O devices, and wait for a long time for I/O operations to complete, can be treated as I/O bound processes.

CPU bound Processes: the processes which require a lot of CPU time, can be treated as CPU bound processes.

Process Priority

A common type of scheduling algorithm is priority-based scheduling. The idea is to rank processes based on their worth and need for processor time. Processes with a higher priority will run before those with a lower priority, while processes with the same priority are scheduled round-robin (one after the next, repeating).

Static Priority: The users assign this priority to real-time processes ranging from 1 to 99. It is not possible for the scheduler to change this static priority.

Dynamic Priority: it is sum of base priority time and the CPU time left for remaining process.

The static priority is always higher than dynamic priority. When there is no real time process in running state, then only the scheduler will prefer to run conventional processes.

Scheduling Policies

SCHED_FIFO: First in-first out scheduling

SCHED_FIFO can be used only with static priorities higher than 0, which means that when a **SCHED_FIFO** threads becomes runnable, it will always immediately preempt any currently running **SCHED_OTHER**, **SCHED_BATCH**, or **SCHED_IDLE** thread. **SCHED_FIFO** is a simple scheduling algorithm without time slicing

SCHED_RR: round robin scheduling

SCHED_RR is a simple enhancement of **SCHED_FIFO**. Everything described above for **SCHED_FIFO** also applies to **SCHED_RR**, except that each thread is allowed to run only for a maximum time quantum.

SCHED_OTHER: Default Linux time-sharing scheduling

SCHED_OTHER can be used at only static priority 0 (i.e., threads under real-time policies always have priority over **SCHED_OTHER** processes). **SCHED_OTHER** is the standard Linux time-sharing scheduler that is intended for all threads that do not require the special real-time mechanisms.

Scheduling in Multiprocessor Systems

In a multiprocessor system hopefully all of the processors are busily running processes. Each will run the scheduler separately as its current process exhausts its time-slice or has to wait for a system resource. The first thing to notice about an SMP system is that there is not just one idle process in the system. In a single processor system the idle process is the first task in

the task vector, in an SMP system there is one idle process per CPU as you could have more than one idle CPU. Additionally there is one current process per CPU, so SMP systems must keep track of the current and idle processes for each processor.

Personalities

Name

personality - set the process execution domain

Synopsis

```
#include <sys/personality.h>
int personality(unsigned long persona);
```

Description

Linux supports different execution domains, or personalities, for each process. Among other things, execution domains tell Linux how to map signal numbers into signal actions. The execution domain system allows Linux to provide limited support for binaries compiled under other UNIX-like operating systems.

This function will return the current **personality()** when *persona* equals 0xffffffff. Otherwise, it will make the execution domain referenced by *persona* the new execution domain of the calling process.

Return Value

On success, the previous *persona* is returned. On error, -1 is returned, and *errno* is set appropriately.

Cloning

NAME

clone - create a child process

SYNOPSIS

```
#define _GNU_SOURCE
#include <sched.h>
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg, ...
          /* pid_t *ptid, void *newtls, pid_t *ctid */ );
```

/* For the prototype of the raw system call, see NOTES */

DESCRIPTION

`clone()` creates a new process, in a manner similar to [fork\(2\)](#).

This page describes both the glibc `clone()` wrapper function and the underlying system call on which it is based. The main text describes the wrapper function; the differences for the raw system call are described toward the end of this page.

Unlike [fork\(2\)](#), `clone()` allows the child process to share parts of its execution context with the calling process, such as the virtual address space, the table of file descriptors, and the table of signal handlers.

One use of `clone()` is to implement threads: multiple flows of control in a program that run concurrently in a shared address space.

Signals

Signal is a notification, a message sent by either operating system or some application to our program. Signals are a mechanism for one-way asynchronous notifications. A signal may be sent from the kernel to a process, from a process to another process, or from a process to itself. Signal typically alert a process to some event, such as a segmentation fault, or the user pressing Ctrl-C.

Linux kernel implements about 30 signals. Each signal identified by a number, from 1 to 31. Signals don't carry any argument and their names are mostly self explanatory. For instance **SIGKILL** or signal number **9** tells the program that someone tries to kill it, and **SIGHUP** used to signal that a terminal hangup has occurred, and it has a value of **1** on the i386 architecture.

With the exception of **SIGKILL** and **SIGSTOP** which always terminates the process or stops the process, respectively, processes may control what happens when they receive a signal. They can

1. accept the default action, which may be to terminate the process, terminate and coredump the process, stop the process, or do nothing, depending on the signal.
2. Or, processes can elect to explicitly ignore or handle signals.
 1. **Ignored signals** are silently dropped.
 2. **Handled signals** cause the execution of a user-supplied **signal handler** function. The program jumps to this function as soon as the signal is received, and the control of the program resumes at the previously interrupted instructions.

Signal	Name	Description
SIGHUP	1	Hangup (POSIX)
SIGINT	2	Terminal interrupt (ANSI)
SIGQUIT	3	Terminal quit (POSIX)

SIGKILL	9	Kill(can't be caught or ignored) (POSIX)
SIGALRM	14	Alarm clock (POSIX)
SIGCHLD	17	Child process has stopped or exited, changed (POSIX)
SIGTERM	15	Termination (ANSI)

Note that, we cannot ignore SIGSTOP and SIGKILL signals, because these signals provide a way for the kernel or root user to stop or kill process in any circumstances. The default action for these two signals is to terminate the process. These two signals can neither be caught nor be ignored.

Sending and Receiving Signals:

The term **raise** is used to indicate the generation of a signal, and the term **catch** is used to indicate the receipt of a signal.

Signals are raised by error conditions, and they are generated by the shell and terminal handlers to cause interrupts and can also be sent from one process to another to pass information or to modify the behavior.

Signals can be:

1. Raised
2. Caught
3. Acted upon
4. Ignored

If a process receives signals such as **SIGFPE**, **SIGKILL**, etc., the process will be terminated immediately, and a **core dump** file is created. The **core** file is an image of the process, and we can use it to debug.

Here is an example of the common situation when we use a signal: when we type the **interrupt character (Ctrl+C)**, the **SIGINT** signal will be sent to the **foreground process**(the program currently running). This will cause the program to terminate unless it has some arrangement for catching the signal.

The command **kill** can be used to send a signal to a process other than the current foreground process.

Development with LINUX:

Everyone has a favorite development platform. With Windows becoming less popular, is Linux getting the breakthrough it needs? The answer, on the surface, is ["Yes."](#) If you haven't yet experimented with the operating system yourself, now's the time to give it a try.

According to the [Stack Overflow 2016 survey](#), desktop developers now constitute just 6.9 percent of all developers (and that also includes Mac Desktops). Over half of programming jobs are now for Web developers, and that has opened up development on alternative platforms such as Linux and Mac. That's why, after 20 years of focusing on building Windows applications, I decided to learn Linux, specifically Ubuntu.

So Why Switch?

Here's why you might prefer Linux to Mac or Windows.

- It offers greater breadth and depth of open-source software.
- It's less fiddly running open-source software on Linux.
- It runs on anything, especially an old Windows PC.
- Once you've learned the terminal commands, you can be extremely efficient and productive.
- Rebuilding a Windows Development PC after a major crash can take hours or days to reinstall everything. Linux, by contrast, is a lot quicker and also easier to containerize.
- The development environment is similar to production. It's also likely to stay stable longer, as there's less commercial pressure to release new OS versions.
- In the world of servers, it reigns supreme. It's also a major player in mobile, thanks to Android.

Let's look at a couple of these points:

Terminal Commands

Linux at heart is a command-line operating system with many commands for controlling and configuring the system, running applications, and so on. You can control everything from the command line.

This might be a bit of a controversial statement, but I've not seen any Linux GUIs that are quite as good as Windows. To be fair to those GUIs, they are far more user-friendly than a terminal, and let you run multiple terminals at once; you can also browse around the file system far more easily than issuing commands in a terminal. (To be a Linux developer, you should feel at home with terminal commands, particularly for installing and updating software.)

Irrespective of the GUI used, if you are familiar with the terminal commands and one of the shells, you can find your way around any Linux deployment. There are many popular shells, though everyone knows Bash (a.k.a. Bourne-again Shell); they differ in features such as history, saving commands as scripts, command line completion and the like.

Shell scripts are not unlike Windows batch files for controlling the system, but a lot more powerful. They constitute a "mini-programming language," albeit with many more commands, and include constants, string and integer variables; if, case, for, while and until loops; as well as job control, shell functions and aliases (and lots more).

Rebuilding After a Major Crash

If you've been developing on Windows for a couple of years, you've likely experienced a catastrophic failure at some point, or had to move all your dev tools to a new version of Windows.

Windows 10 is perhaps better at handling crashes than previous versions, but I've had to reload my entire development package on Windows 95, 2000, XP and 7. None of these were total disk crashes, and I had backups, but each wasted a day or two before I could get back to my pre-crash state.

There are probably less than 20 pieces of development software that I need on a PC, but that still means a lot of time spent reacquiring and reinstalling.

You can have the same issues with Linux, except most Linux upgrades don't require application reinstalls (and I've never experienced a catastrophic crash while working with Linux). In the event of a total implosion, much of the software can be reinstalled with batch files made up of multiple *sudo apt install* statements. It's just far quicker.

Gcc:

The GNU compiler collection(gcc) is a command line compiler on linux systems. It is also c and c++ compiler developed by the GNU project. For example, if one had written a program whose filename is 'add', the compilation procedure is as follows:

- The primary way of compiling that file 'add.c' into an executable file called 'add' is gcc -o add add.c.
- If the program compiles without any errors, it can be executed by typing './add'

Gdb:

The gdb will step through the source code line by line or instruction by instruction. It is also possible to know the value of any variable at run time.

Gnat:

It is an acronym for GNU NYU ada translator (GNAT). It is a free software compiler for the programming language 'ada' which is part of the GNU compiler collection. The gcc compiler is having the capability of compiling programs written in several languages including ada95, and c. if the file extension is either '.ads' or '.adb', it assumes that the given program has been written in ada and, then it will call the GNAT compiler to compile the particular specified file.

What is MySQL

MySQL is a fast, easy to use relational database. It is currently the most popular open-source database. It is very commonly used in conjunction with PHP scripts to create powerful and dynamic server-side applications.

MySQL is used for many small and big businesses. It is developed, marketed and supported by MySQL AB, a Swedish company. It is written in C and C++.

Reasons of popularity

MySQL is becoming so popular because of these following reasons:

- MySQL is an open-source database so you don't have to pay a single penny to use it.
- MySQL is a very powerful program so it can handle a large set of functionality of the most expensive and powerful database packages.
- MySQL is customizable because it is an open source database and the open-source GPL license facilitates programmers to modify the SQL software according to their own specific environment.
- MySQL is quicker than other databases so it can work well even with the large data set.
- MySQL supports many operating systems with many languages like PHP, PERL, C, C++, JAVA, etc.
- MySQL uses a standard form of the well-known SQL data language.
- MySQL is very friendly with PHP, the most popular language for web development.
- MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB).

History of MySQL

MySQL is an open source database product that was created by MySQL AB, a company founded in 1995 in Sweden. In 2008, MySQL AB announced that it had agreed to be acquired by Sun Microsystems for approximately \$1 billion.

Initial Efforts

The project of MySQL was started in 1979, when MySQL's inventor, Michael Widenius developed an in-house database tool called UNIREG for managing databases. After that UNIREG has been rewritten in several different languages and extended to handle big databases. After some time Michael Widenius contacted David Hughes, the author of mSQL, to see if Hughes would be interested in connecting mSQL to UNIREG's B+ ISAM handler to provide indexing to mSQL. That's the way MySQL came in existence.

MySQL is named after the daughter of Michael Widenius whose name is "My".

MySQL Features

- **Relational Database Management System (RDBMS):** MySQL is a relational database management system.
- **Easy to use:** MySQL is easy to use. You have to get only the basic knowledge of SQL. You can build and interact with MySQL with only a few simple SQL statements.
- **It is secure:** MySQL consist of a solid data security layer that protects sensitive data from intruders. Passwords are encrypted in MySQL.
- **Client/ Server Architecture:** MySQL follows a client /server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they query data, save changes, etc.
- **Free to download:** MySQL is free to use and you can download it from MySQL official website.
- **It is scalable:** MySQL can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB. However, you can increase this number to a theoretical limit of 8 TB of data.
- **Compatibale on many operating systems:** MySQL is compatible to run on many operating systems, like Novell NetWare, Windows* Linux*, many varieties of UNIX* (such as Sun* Solaris*, AIX, and DEC* UNIX), OS/2, FreeBSD*, and others. MySQL also provides a facility that the clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).
- **Allows roll-back:** MySQL allows transactions to be rolled back, commit and crash recovery.
- **High Performance:** MySQL is faster, more reliable and cheaper because of its unique storage engine architecture.
- **High Flexibility:** MySQL supports a large number of embedded applications which makes MySQL very flexible.
- **High Productivity:** MySQL uses Triggers, Stored procedures and views which allows the developer to give a higher productivity.

Disadvantages / Drawback of MySQL:

Following are the few disadvantages of MySQL:

- MySQL version less than 5.0 doesn't support ROLE, COMMIT and stored procedure.
- MySQL does not support a very large database size as efficiently.
- MySQL doesn't handle transactions very efficiently and it is prone to data corruption.
- MySQL is accused that it doesn't have a good developing and debugging tool compared to paid databases.
- MySQL doesn't support SQL check constraints.

MySQL Data Types

A Data Type specifies a particular type of data, like integer, floating points, Boolean etc. It also identifies the possible values for that type, the operations that can be performed on that type and the way the values of that type are stored.

MySQL supports a lot number of SQL standard data types in various categories. It uses many different data types broken into mainly three categories: numeric, date and time, and string types.

Numeric Data Type

Data Type Syntax	Description
INT	A normal-sized integer that can be signed or unsigned. If signed, the allowable range is from -2147483648 to 2147483647. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
TINYINT	A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
SMALLINT	A small integer that can be signed or unsigned. If signed, the allowable range is from -32768 to 32767. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
MEDIUMINT	A medium-sized integer that can be signed or unsigned. If signed, the allowable range is from -8388608 to 8388607. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
BIGINT	A large integer that can be signed or unsigned. If signed, the allowable range is from -9223372036854775808 to 9223372036854775807. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.
FLOAT(m,d)	A floating-point number that cannot be unsigned. You can define the display length (m) and the number of decimals (d). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals). Decimal precision can go to 24 places for a float.
DOUBLE(m,d)	A double precision floating-point number that cannot be unsigned. You can define the display length (m) and the number of decimals (d). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a double. Real is a synonym for double.
DECIMAL(m,d)	An unpacked floating-point number that cannot be unsigned. In unpacked decimals, each decimal corresponds to one byte. Defining the display length (m) and the number of decimals (d) is required. Numeric is a synonym for decimal.

Date and Time Data Type:

Data Type Syntax	Maximum Size	Explanation
DATE	Values range from '1000-01-01' to '9999-12-31'.	Displayed as 'yyyy-mm-dd'.
DATETIME	Values range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.	Displayed as 'yyyy-mm-dd hh:mm:ss'.
TIMESTAMP(m)	Values range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' TC.	Displayed as 'YYYY-MM-DD HH:MM:SS'.
TIME	Values range from '-838:59:59' to '838:59:59'.	Displayed as 'HH:MM:SS'.
YEAR[(2 4)]	Year value as 2 digits or 4 digits.	Default is 4 digits.

String Data Types:

Data Type Syntax	Maximum Size	Explanation
CHAR(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Fixed-length strings. Space padded on right to equal size characters.
VARCHAR(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Variable-length string.
TINYTEXT(size)	Maximum size of 255 characters.	Where size is the number of characters to store.
TEXT(size)	Maximum size of 65,535 characters.	Where size is the number of characters to store.
MEDIUMTEXT(size)	Maximum size of 16,777,215 characters.	Where size is the number of characters to store.
LONGTEXT(size)	Maximum size of 4GB or 4,294,967,295 characters.	Where size is the number of characters to store.
BINARY(size)	Maximum size of 255 characters.	Where size is the number of binary characters to store. Fixed-length strings. Space padded on right to equal size characters. (introduced in MySQL 4.1.2)
VARBINARY(size)	Maximum size of 255 characters.	Where size is the number of characters to store. Variable-length string. (introduced in MySQL 4.1.2)

Large Object Data Types (LOB) Data Types:

Data Type Syntax	Maximum Size
TINYBLOB	Maximum size of 255 bytes.
BLOB(size)	Maximum size of 65,535 bytes.
MEDIUMBLOB	Maximum size of 16,777,215 bytes.
LONGTEXT	Maximum size of 4gb or 4,294,967,295 characters.

How to install MySQL

Download MySQL

Follow these steps:

- Go to MySQL official website <http://www.mysql.com/downloads/>
- Choose the version number for MySQL community server which you want.

Installing MySQL on Windows

Your downloaded MySQL is neatly packaged with an installer. Download the installer package, unzip it anywhere and run setup.exe.

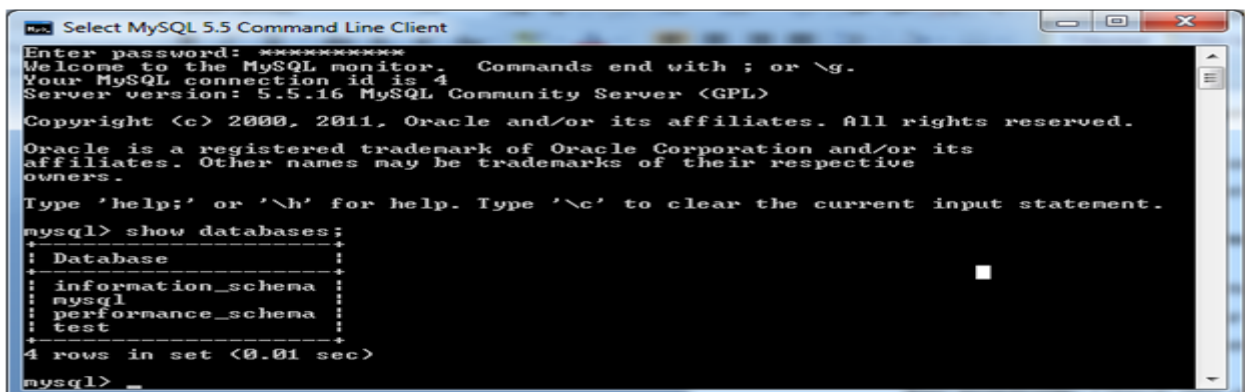
By default, this process will install everything under C:\mysql.

Verify MySQL installation

Once MySQL has been successfully installed, the base tables have been initialized, and the server has been started, you can verify its working via some simple tests.

Open your MySQL Command Line Client, it should be appeared with a mysql> prompt. If you have set any password, write your password here. Now, you are connected to the MySQL server and you can execute all the SQL command at mysql> prompt as follows:

For example: Check the already created databases with show databases command:



```
Select MySQL 5.5 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.16 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| test |
+-----+
4 rows in set (0.01 sec)

mysql> _
```

MySQL User Creation: Setting Up a New MySQL User Account

How do I create a user account on MySQL database server?

When you try to access MySQL database server from client such as mysql or even programming language such as php or perl you need a user account. MySQL has sophisticated user management system that controls who can access server and from which client system. It uses special tables in mysql database. In order to create a new user account you need a MySQL root account password. You need to use the **GRANT** SQL command to set up the MySQL user account. Finally, use the account's name and password to make connections to the MySQL server.

Please note that **MySQL root user account is different from UNIX/Linux root login account**. For example, the MySQL root user and the Linux/Unix root user are separate and have nothing to do with each other, even though the username is the same in each case.

Setup a root user password

To setup root password for first time, use mysqladmin command at shell prompt as follows:

```
$ mysqladmin -u root password NEWPASSWO
```

If you want to change or update a root user password, then you need to use the following command:

```
$ mysqladmin -u root -p'oldpassword' password newpass
```

Procedure for setting up a MySQL user account

Login in as mysql root user. At shell prompt type the following command:

```
$ mysql -u root -p
```

OR

```
$ mysql -u root -h your-mysql-server-host-name -p
```

Create a new mysql database called demo. Type the following command at mysql> prompt:

```
mysql> CREATE DATABASE demo;
```

Create a new user called user1 for database called demo:

```
mysql> GRANT ALL ON demo.* TO user1@localhost IDENTIFIED BY 'mypassword';
```

How do I connect to MySQL database demo using user1 account?

User *user1* can connect to *demo* database using the following shell command:

```
$ mysql -u user1 -p demo
```

OR

```
$ mysql -u user1 -h your-mysql-server-host-name-here -p demo
```

Where,

- **-u user1** : MySQL Username
- **-h** : MySQL server name (default is localhost)
- **-p** : Prompt for password
- **demo**: demo is name of mysql database (optional)

Starting and Terminating mysql

Problem

You want to start and stop the **mysql** program.

Solution

Invoke **mysql** from your command prompt to start it, specifying any connection parameters that may be necessary. To leave **mysql**, use a QUIT statement.

Discussion

To start the **mysql** program, try just typing its name at your command-line prompt. If **mysql** starts up correctly, you'll see a short message, followed by a `mysql>` prompt that indicates the program is ready to accept queries. To illustrate, here's what the welcome message looks like

```
% mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18427 to server version: 3.23.51-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

If **mysql** tries to start but exits immediately with an "access denied" message, you'll need to specify connection parameters. The most commonly needed parameters are the host to connect to (the host where the MySQL server runs), your MySQL username, and a password. For example:

```
% mysql -h localhost -p -u cbuser
Enter password: cbpass
```

In general, I'll show **mysql** commands in examples with no connection parameter options. I assume that you'll supply any parameters that you need, either on the command line, or in an option file (Recipe 1.5) so that you don't have to type them each time you invoke **mysql**.

If you don't have a MySQL username and password, you need to obtain permission to use the MySQL server,

The syntax and default values for the connection parameter options are shown in the following table. These options have both a single-dash short form and a double-dash long form.

Parameter type	Option syntax forms	Default value
Hostname	<code>-h <i>hostname</i></code> <code>--host=<i>hostname</i></code>	localhost
Username	<code>-u <i>username</i></code> <code>--user=<i>username</i></code>	Your login name
Password	<code>-p</code> <code>--password</code>	None

As the

table indicates, there is no default password. To supply one, use `--password` or `-p`, then enter your password when **mysql** prompts you for it:

```
% mysql -pEnter password: ← enter your password here
```

If you like, you can specify the password directly on the command line by using either `-ppassword` (note that there is no space after the `-p`) or `--password=password`. I don't recommend doing this on a multiple-user machine, because the password may be visible momentarily to other users who are running tools such as `ps` that report process information.

If you get an error message that `mysql` cannot be found or is an invalid command when you try to invoke it, that means your command interpreter doesn't know where `mysql` is installed.

To terminate a `mysql` session, issue a QUIT statement:

```
mysql> QUIT
```

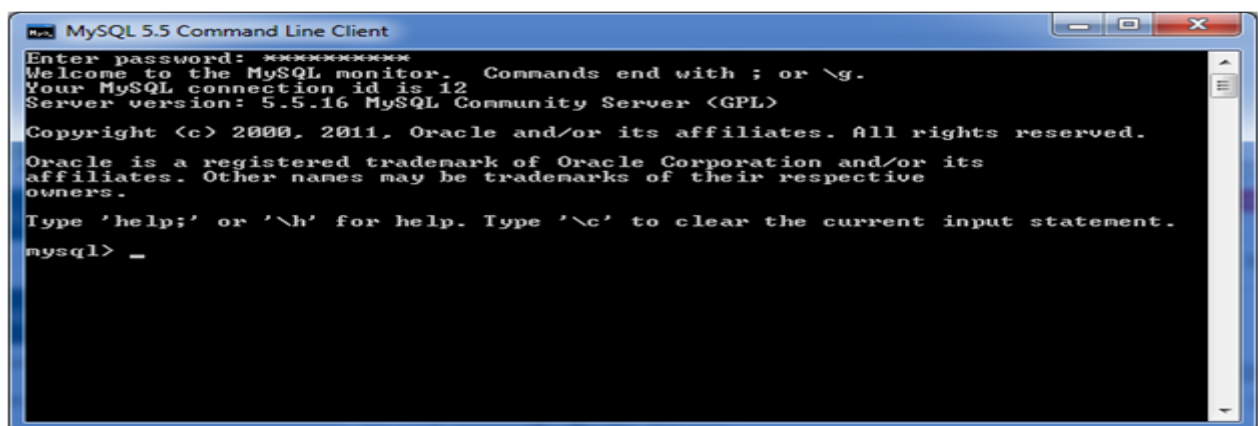
You can also terminate the session by issuing an EXIT statement or (under Unix) by typing Ctrl-D. The way you specify connection parameters for `mysql` also applies to other MySQL programs such as `mysqldump` and `mysqladmin`. For example, some of the actions that `mysqladmin` can perform are available only to the MySQL root account, so you need to specify name and password options for that user:

```
% mysqladmin -p -u root shutdown
Enter password:
```

MySQL Create Database

You can create a MySQL database by using MySQL Command Line Client.

Open the MySQL console and write down password, if you set one while installation. You will get the following:



Now you are ready to create database.

Syntax:

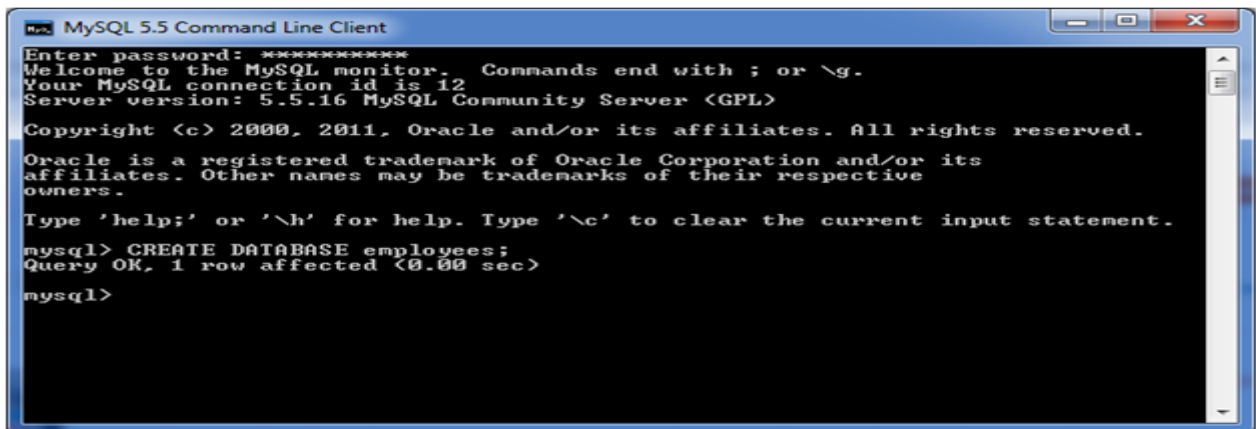
```
CREATE DATABASE database_name;
```

Example:

Let's take an example to create a database name "employees"

```
CREATE DATABASE employees;
```

It will look like this:



```
MySQL 5.5 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 5.5.16 MySQL Community Server (GPL)

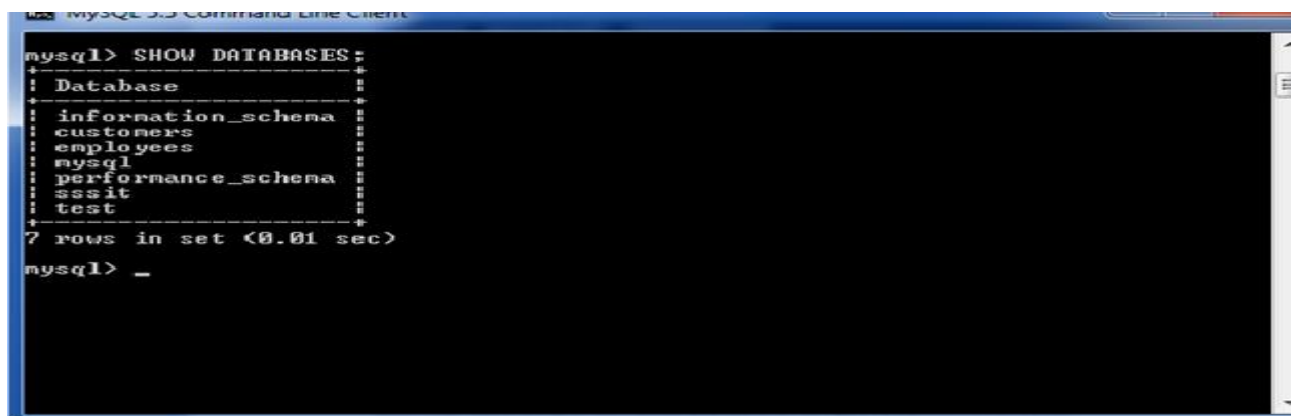
Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> CREATE DATABASE employees;
Query OK, 1 row affected (0.00 sec)
mysql>
```

You can check the created database by the following query:

```
SHOW DATABASES;
```

Output



```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| customers          |
| employees          |
| mysql              |
| performance_schema |
| sssit              |
| test               |
+-----+
7 rows in set (0.01 sec)
mysql> _
```

Here, you can see the all created databases.

MySQL SELECT Database

SELECT Database is used in MySQL to select a particular database to work with. This query is used when multiple databases are available with MySQL Server.

You can use SQL command **USE** to select a particular database.

Syntax:

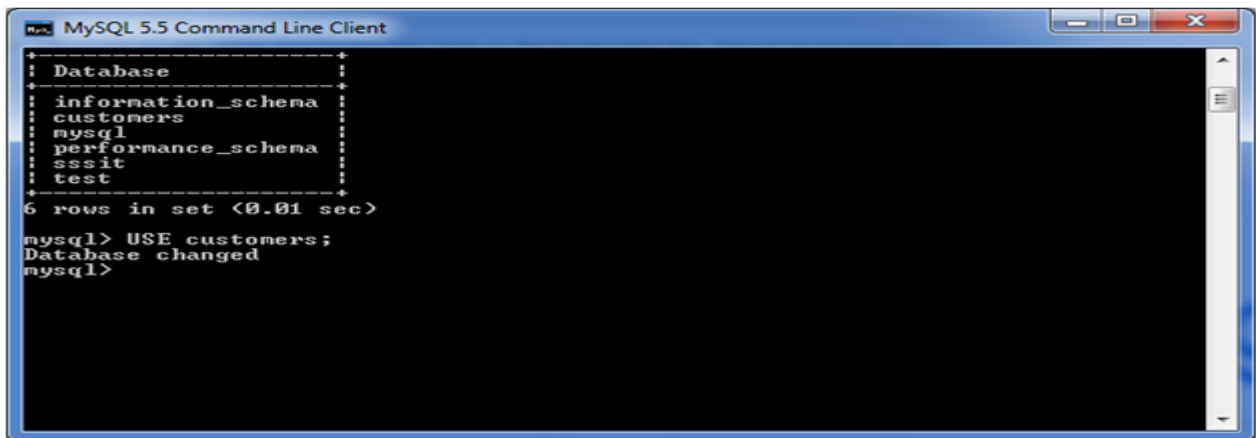
```
USE database_name;
```

Example:

Let's take an example to use a database name "customers".

```
USE customers;
```

It will look like this:



```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| customers |
| mysql |
| performance_schema |
| sssit |
| test |
+-----+
6 rows in set (0.01 sec)

mysql> USE customers;
Database changed
mysql>
```

Note: All the database names, table names and table fields name are case sensitive. You must have to use proper names while giving any SQL command.

MySQL Drop Database

You can drop/delete/remove a MySQL database easily with the MySQL command. You should be careful while deleting any database because you will lose your all the data available in your database.

Syntax:

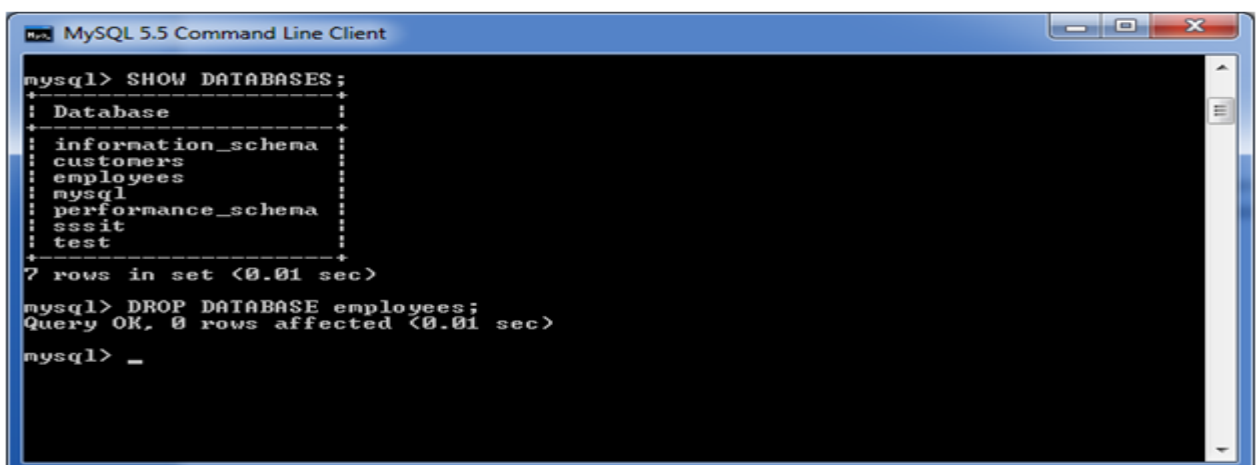
DROP DATABASE database_name;

Example:

Let's take an example to drop a database name "employees"

DROP DATABASE employees;

It will look like this:



```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| customers |
| employees |
| mysql |
| performance_schema |
| sssit |
| test |
+-----+
7 rows in set (0.01 sec)

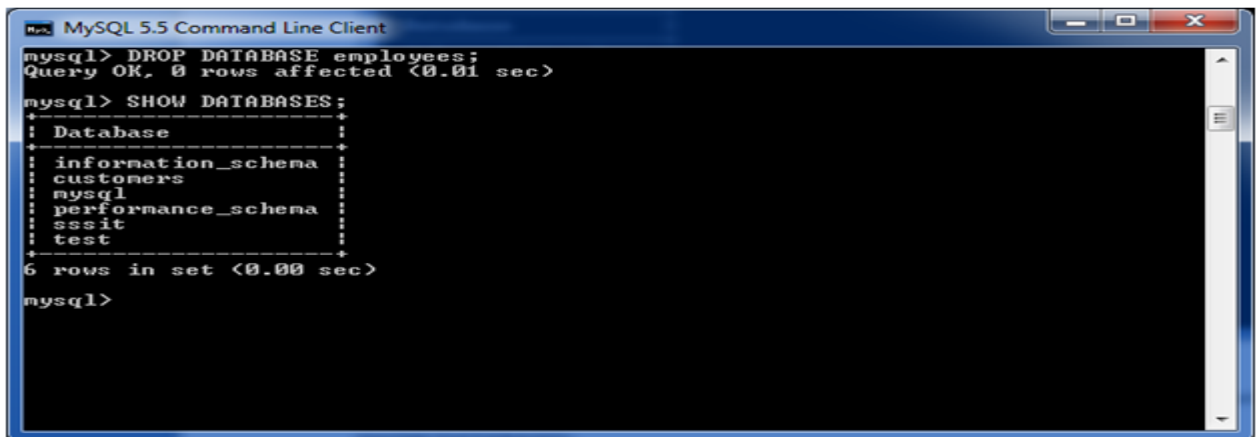
mysql> DROP DATABASE employees;
Query OK, 0 rows affected (0.01 sec)

mysql> _
```

Now you can check that either your database is removed by executing the following query:

SHOW DATABASES;

Output:

A screenshot of the MySQL 5.5 Command Line Client window. The window title is "MySQL 5.5 Command Line Client". The command prompt shows the following sequence of commands and output:

```
mysql> DROP DATABASE employees;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| customers |
| mysql |
| performance_schema |
| sssit |
| test |
+-----+
6 rows in set (0.00 sec)

mysql>
```

Here, you can see that the database "employees" is removed.

MySQL CREATE TABLE

The MySQL CREATE TABLE command is used to create a new table into the database. A table creation command requires three things:

- Name of the table
- Names of fields
- Definitions for each field

Syntax:

Following is a generic syntax for creating a MySQL table in the database.

CREATE TABLE table_name (column_name column_type...);

Example:

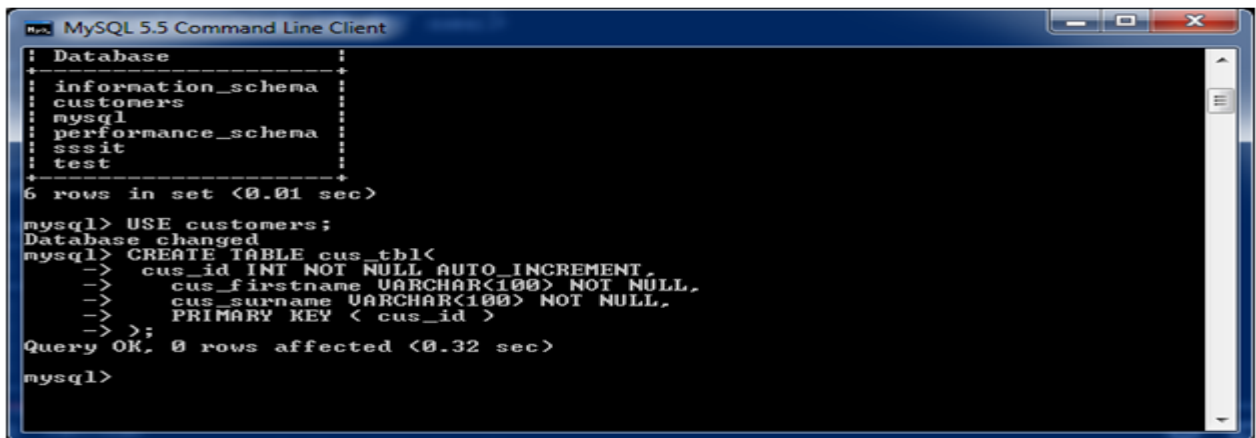
Here, we will create a table named "cus_tbl" in the database "customers".

```
CREATE TABLE cus_tbl(
  cus_id INT NOT NULL AUTO_INCREMENT,
  cus_firstname VARCHAR(100) NOT NULL,
  cus_surname VARCHAR(100) NOT NULL,
  PRIMARY KEY ( cus_id )
);
```

Note:

1. Here, NOT NULL is a field attribute and it is used because we don't want this field to be NULL. If you will try to create a record with NULL value, then MySQL will raise an error.
2. The field attribute AUTO_INCREMENT specifies MySQL to go ahead and add the next available number to the id field. PRIMARY KEY is used to define a column as primary key. You can use multiple columns separated by comma to define a primary key.

Visual representation of creating a MySQL table:

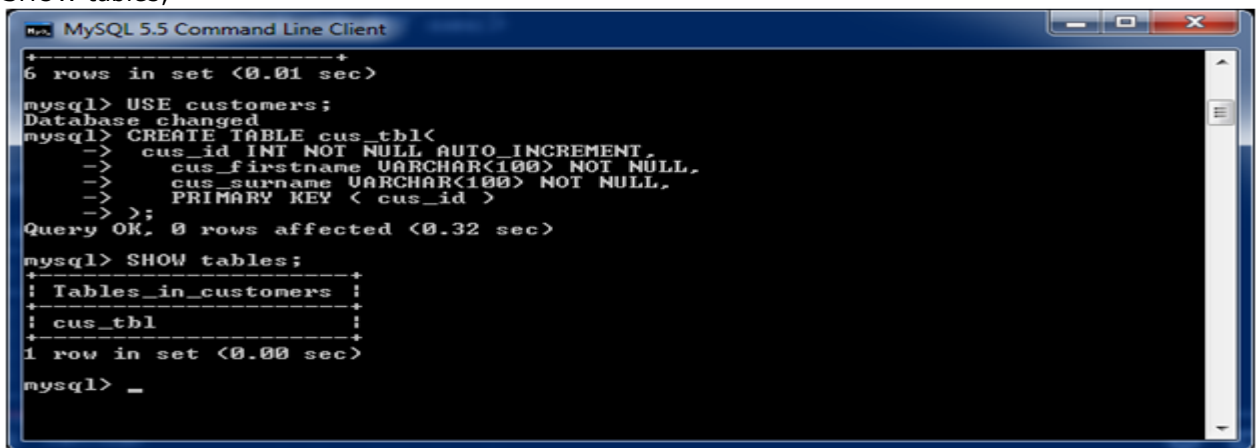


```
mysql> USE customers;
Database changed
mysql> CREATE TABLE cus_tbl(
  ->   cus_id INT NOT NULL AUTO_INCREMENT,
  ->   cus_firstname VARCHAR(100) NOT NULL,
  ->   cus_surname VARCHAR(100) NOT NULL,
  ->   PRIMARY KEY ( cus_id )
  -> );
Query OK, 0 rows affected (0.32 sec)
mysql>
```

See the created table:

Use the following command to see the table already created:

1. SHOW tables;

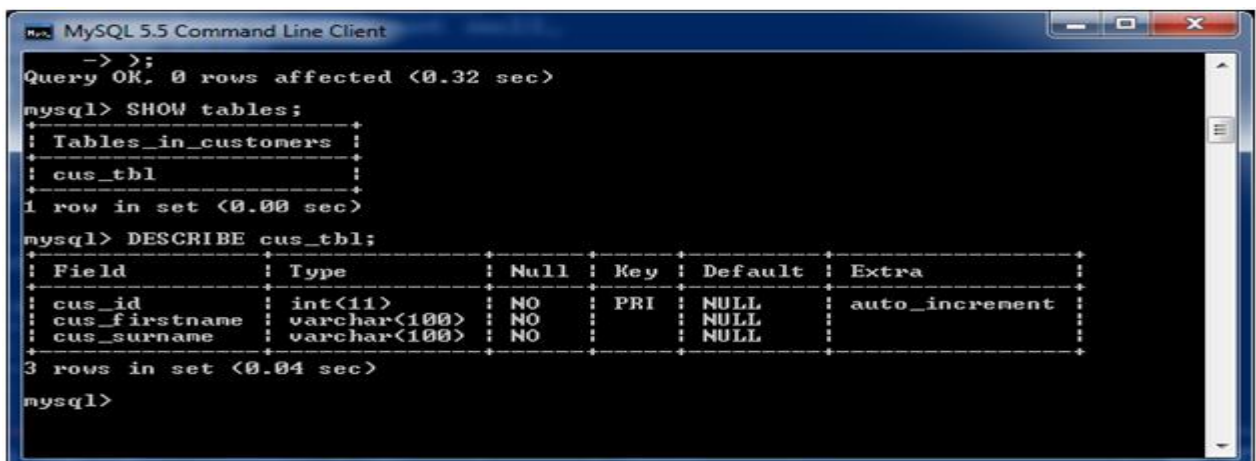


```
mysql> SHOW tables;
+-----+
| Tables_in_customers |
+-----+
| cus_tbl              |
+-----+
1 row in set (0.00 sec)
mysql> _
```

See the table structure:

Use the following command to see the table already created:

1. DESCRIBE cus_tbl;



```
mysql> DESCRIBE cus_tbl;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| cus_id         | int(11)      | NO   | PRI | NULL    | auto_increment |
| cus_firstname  | varchar(100) | NO   |     | NULL    |                |
| cus_surname    | varchar(100) | NO   |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.04 sec)
mysql>
```

MySQL ALTER Table

MySQL ALTER statement is used when you want to change the name of your table or any table field. It is also used to add or delete an existing column in a table.

The ALTER statement is always used with "ADD", "DROP" and "MODIFY" commands according to the situation.

1) ADD a column in the table

Syntax:

```
ALTER TABLE table_name
ADD new_column_name column_definition
[ FIRST | AFTER column_name ];
```

Parameters

table_name: It specifies the name of the table that you want to modify.

new_column_name: It specifies the name of the new column that you want to add to the table.

column_definition: It specifies the data type and definition of the column (NULL or NOT NULL, etc).

FIRST | AFTER column_name: It is optional. It tells MySQL where in the table to create the column. If this parameter is not specified, the new column will be added to the end of the table.

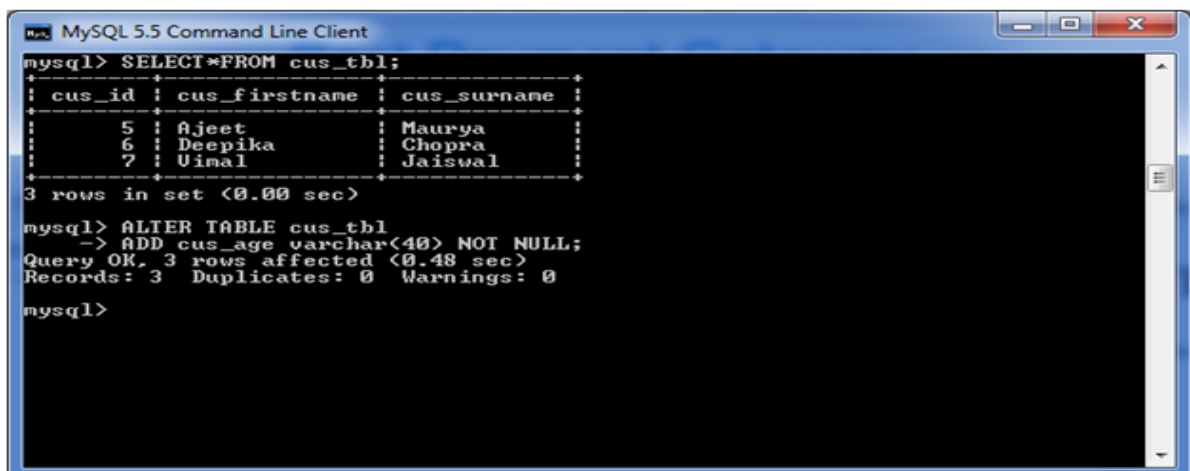
Example:

In this example, we add a new column "cus_age" in the existing table "cus_tbl".

Use the following query to do this:

```
ALTER TABLE cus_tbl
ADD cus_age varchar(40) NOT NULL;
```

Output:



```
MySQL 5.5 Command Line Client
mysql> SELECT*FROM cus_tbl;
+----+-----+-----+
| cus_id | cus_firstname | cus_surname |
+----+-----+-----+
| 5 | Ajeet | Maurya |
| 6 | Deepika | Chopra |
| 7 | Uimal | Jaiswal |
+----+-----+-----+
3 rows in set (0.00 sec)

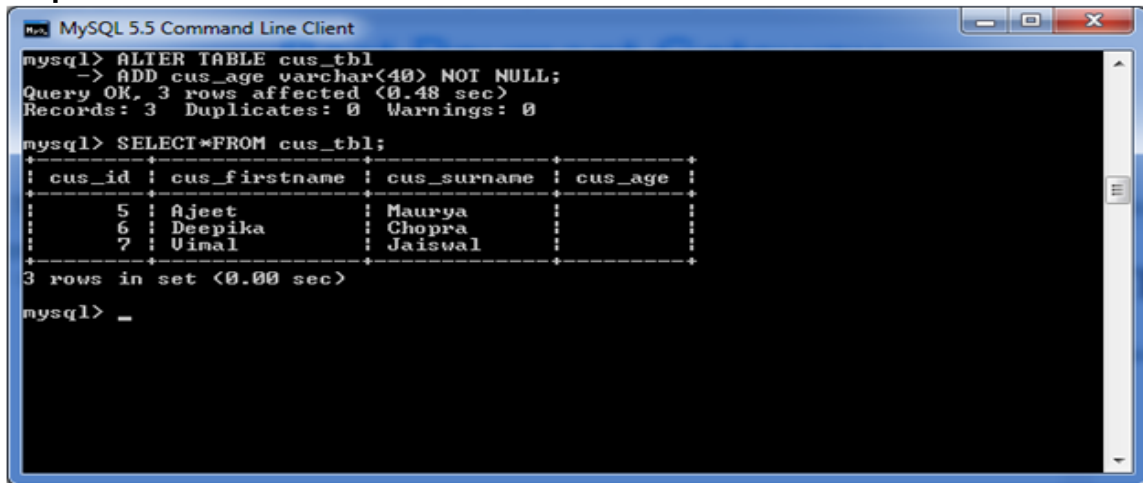
mysql> ALTER TABLE cus_tbl
-> ADD cus_age varchar(40) NOT NULL;
Query OK, 3 rows affected (0.48 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql>
```

See the recently added column:

1. **SELECT* FROM** cus_tbl;

Output:



```
mysql> ALTER TABLE cus_tbl
-> ADD cus_age varchar(40) NOT NULL;
Query OK, 3 rows affected (0.48 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT*FROM cus_tbl;
+----+-----+-----+-----+
| cus_id | cus_firstname | cus_surname | cus_age |
+----+-----+-----+-----+
| 5 | Ajeet | Maurya | |
| 6 | Deepika | Chopra | |
| 7 | Uimal | Jaiswal | |
+----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

- 2) Add multiple columns in the table

Syntax:

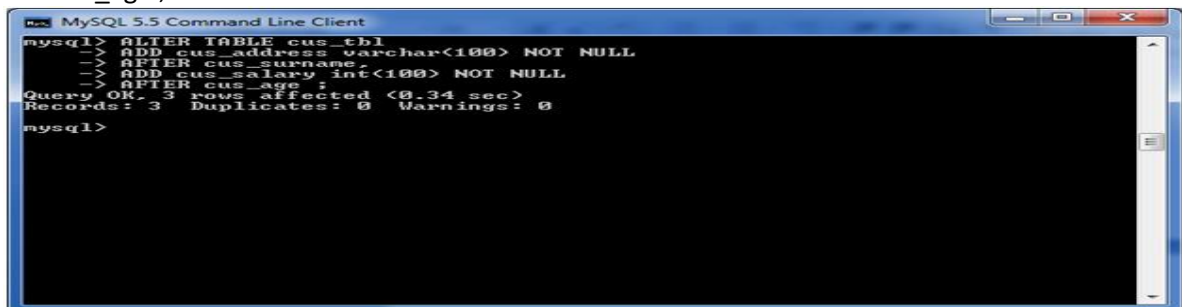
```
ALTER TABLE table_name
ADD new_column_name column_definition
[ FIRST | AFTER column_name ],
ADD new_column_name column_definition
[ FIRST | AFTER column_name ],
... ;
```

Example:

In this example, we add two new columns "cus_address", and cus_salary in the existing table "cus_tbl". cus_address is added after cus_surname column and cus_salary is added after cus_age column.

Use the following query to do this:

```
ALTER TABLE cus_tbl
ADD cus_address varchar(100) NOT NULL
AFTER cus_surname,
ADD cus_salary int(100) NOT NULL
AFTER cus_age ;
```

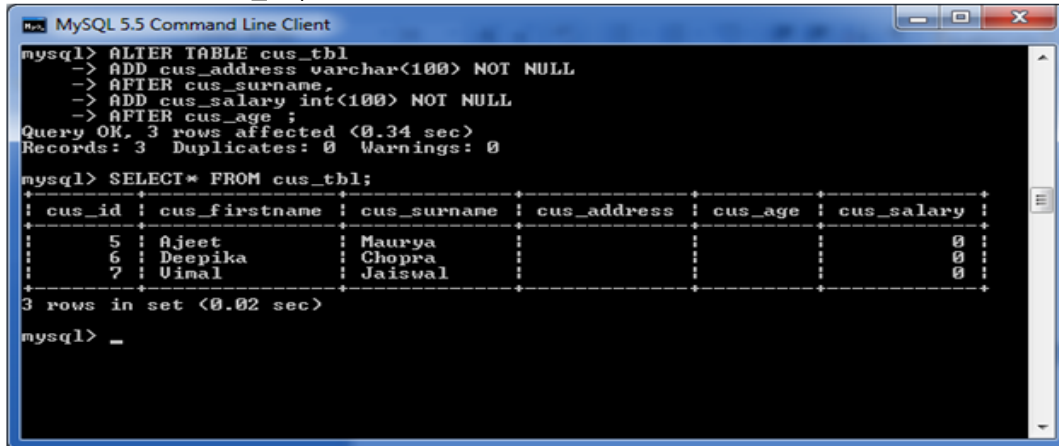


```
mysql> ALTER TABLE cus_tbl
-> ADD cus_address varchar(100) NOT NULL
-> AFTER cus_surname,
-> ADD cus_salary int(100) NOT NULL
-> AFTER cus_age ;
Query OK, 3 rows affected (0.34 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql>
```

See the recently added columns:

SELECT* FROM cus_tbl;



```
mysql> ALTER TABLE cus_tbl
-> ADD cus_address varchar(100) NOT NULL
-> AFTER cus_surname,
-> ADD cus_salary int(100) NOT NULL
-> AFTER cus_age ;
Query OK, 3 rows affected (0.34 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT* FROM cus_tbl;
+----+-----+-----+-----+-----+-----+
| cus_id | cus_firstname | cus_surname | cus_address | cus_age | cus_salary |
+----+-----+-----+-----+-----+-----+
| 5 | Ajeet | Maurya | | | 0 |
| 6 | Deepika | Chopra | | | 0 |
| 7 | Uimal | Jaiswal | | | 0 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.02 sec)

mysql> _
```

3) MODIFY column in the table

The MODIFY command is used to change the column definition of the table.

Syntax:

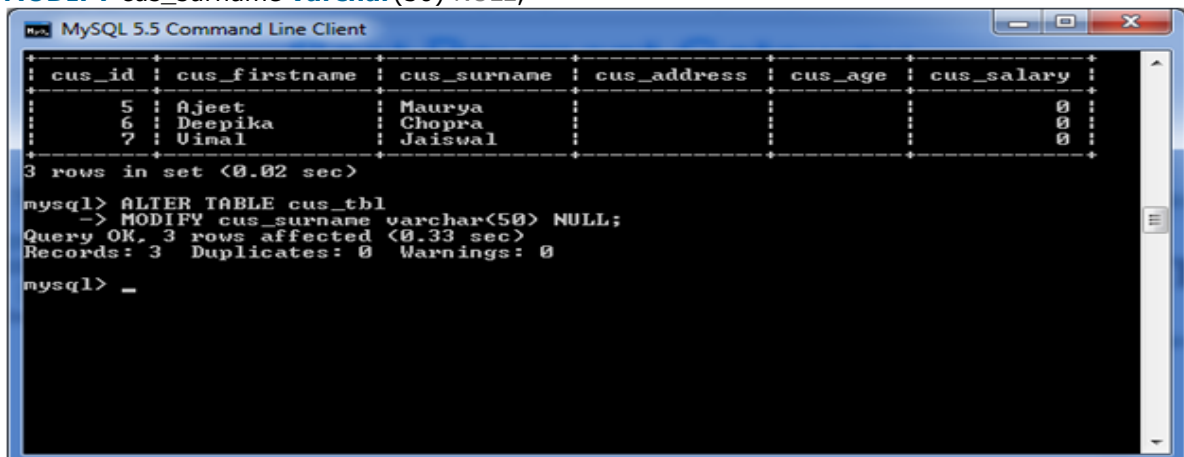
ALTER TABLE table_name
MODIFY column_name column_definition
[**FIRST** | **AFTER** column_name];

Example:

In this example, we modify the column cus_surname to be a data type of varchar(50) and force the column to allow NULL values.

Use the following query to do this:

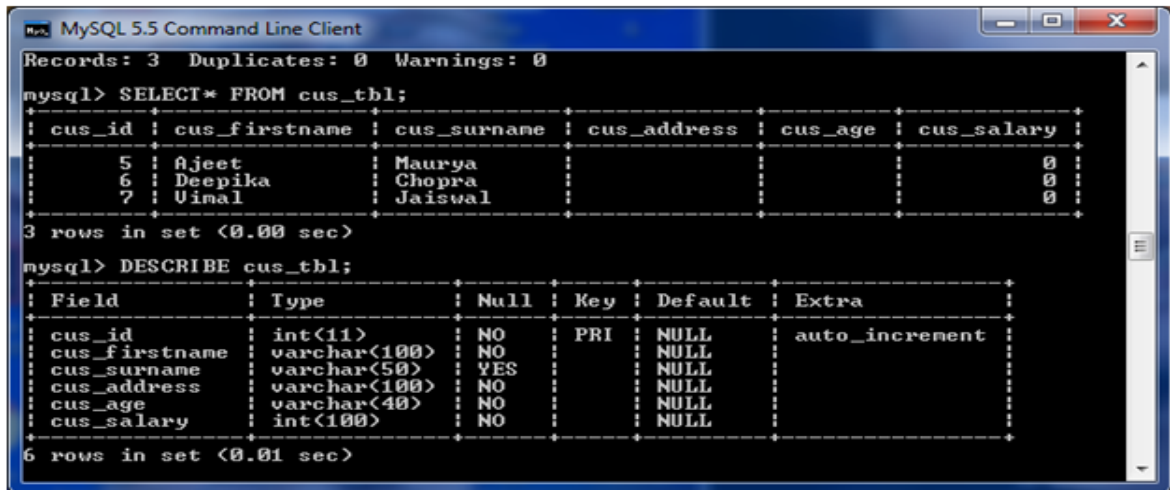
ALTER TABLE cus_tbl
MODIFY cus_surname **varchar**(50) **NULL**;



```
mysql> ALTER TABLE cus_tbl
-> MODIFY cus_surname varchar(50) NULL;
Query OK, 3 rows affected (0.33 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> _
```

See the table structure:



```
MySQL 5.5 Command Line Client
Records: 3 Duplicates: 0 Warnings: 0
mysql> SELECT* FROM cus_tbl;
+----+-----+-----+-----+-----+-----+
| cus_id | cus_firstname | cus_surname | cus_address | cus_age | cus_salary |
+----+-----+-----+-----+-----+-----+
| 5 | Ajeet | Maurya | | | 0 |
| 6 | Deepika | Chopra | | | 0 |
| 7 | Vimal | Jaiswal | | | 0 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> DESCRIBE cus_tbl;
+----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+----+-----+-----+-----+-----+-----+
| cus_id | int(11) | NO | PRI | NULL | auto_increment |
| cus_firstname | varchar(100) | NO | | NULL | |
| cus_surname | varchar(50) | YES | | NULL | |
| cus_address | varchar(100) | NO | | NULL | |
| cus_age | varchar(40) | NO | | NULL | |
| cus_salary | int(100) | NO | | NULL | |
+----+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)
```

4) DROP column in table

Syntax:

ALTER TABLE table_name

DROP COLUMN column_name;

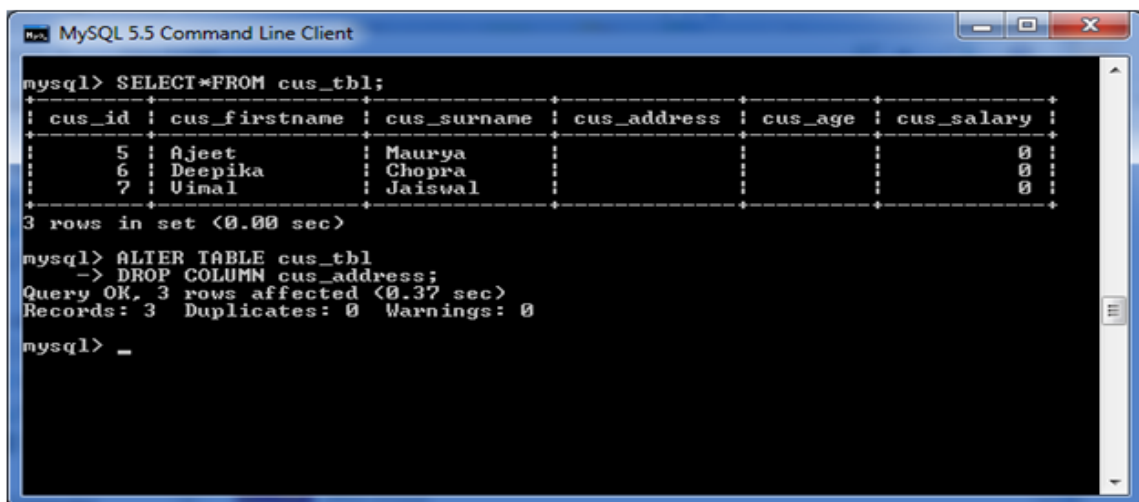
Let's take an example to drop the column name "cus_address" from the table "cus_tbl".

Use the following query to do this:

ALTER TABLE cus_tbl

DROP COLUMN cus_address;

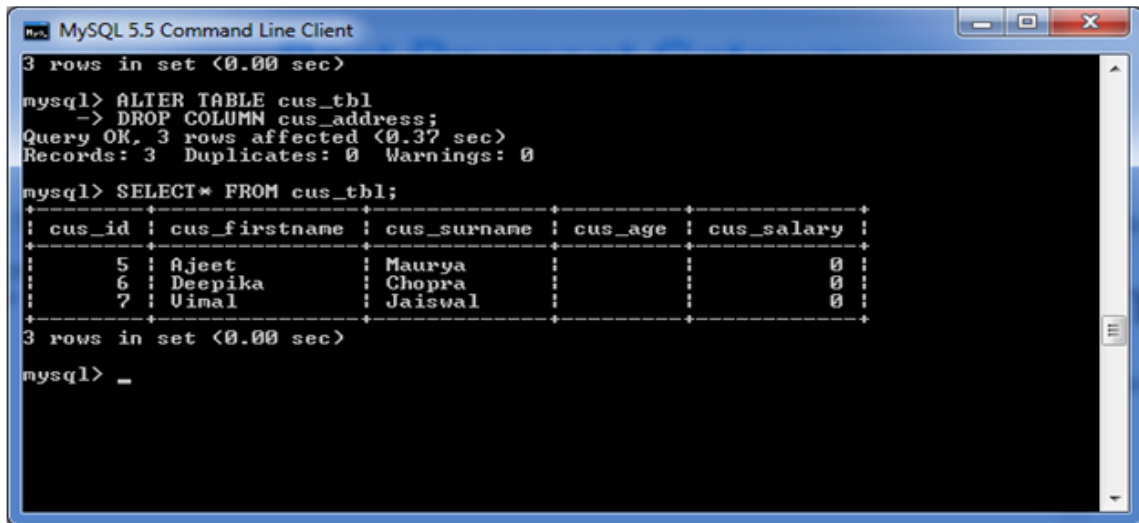
Output:



```
MySQL 5.5 Command Line Client
mysql> SELECT*FROM cus_tbl;
+----+-----+-----+-----+-----+-----+
| cus_id | cus_firstname | cus_surname | cus_address | cus_age | cus_salary |
+----+-----+-----+-----+-----+-----+
| 5 | Ajeet | Maurya | | | 0 |
| 6 | Deepika | Chopra | | | 0 |
| 7 | Vimal | Jaiswal | | | 0 |
+----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> ALTER TABLE cus_tbl
-> DROP COLUMN cus_address;
Query OK, 3 rows affected (0.37 sec)
Records: 3 Duplicates: 0 Warnings: 0
mysql> _
```

See the table structure:



```
mysql> ALTER TABLE cus_tbl
-> DROP COLUMN cus_address;
Query OK, 3 rows affected (0.37 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> SELECT* FROM cus_tbl;
+----+-----+-----+-----+-----+
| cus_id | cus_firstname | cus_surname | cus_age | cus_salary |
+----+-----+-----+-----+-----+
| 5 | Ajeet | Maurya |  | 0 |
| 6 | Deepika | Chopra |  | 0 |
| 7 | Uimal | Jaiswal |  | 0 |
+----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

5) RENAME column in table

Syntax:

```
ALTER TABLE table_name
CHANGE COLUMN old_name new_name
column_definition
[ FIRST | AFTER column_name ]
```

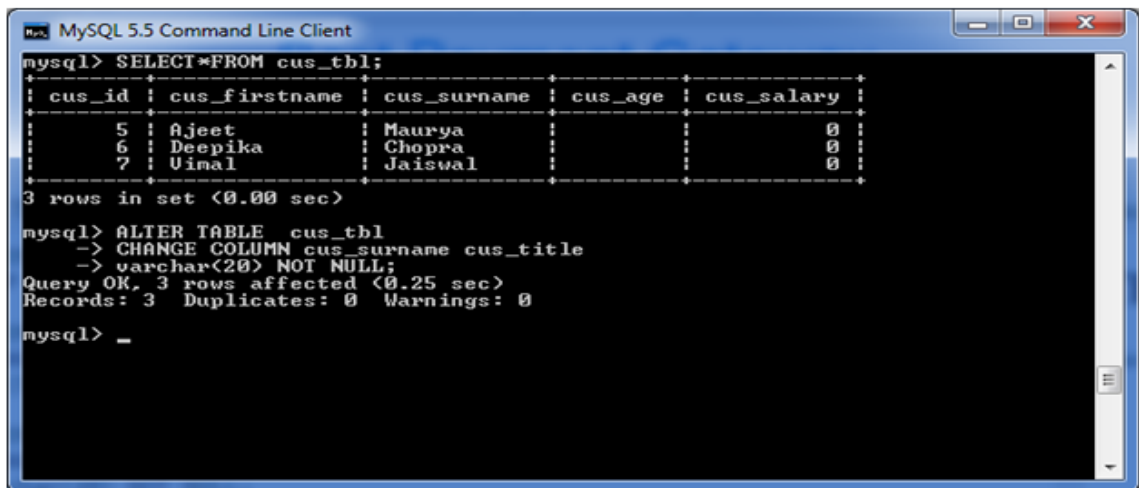
Example:

In this example, we will change the column name "cus_surname" to "cus_title".

Use the following query to do this:

```
ALTER TABLE cus_tbl
CHANGE COLUMN cus_surname cus_title
varchar(20) NOT NULL;
```

Output:



```
mysql> SELECT*FROM cus_tbl;
+----+-----+-----+-----+-----+
| cus_id | cus_firstname | cus_surname | cus_age | cus_salary |
+----+-----+-----+-----+-----+
| 5 | Ajeet | Maurya |  | 0 |
| 6 | Deepika | Chopra |  | 0 |
| 7 | Uimal | Jaiswal |  | 0 |
+----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> ALTER TABLE cus_tbl
-> CHANGE COLUMN cus_surname cus_title
-> varchar(20) NOT NULL;
Query OK, 3 rows affected (0.25 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> _
```

6) RENAME table

Syntax:

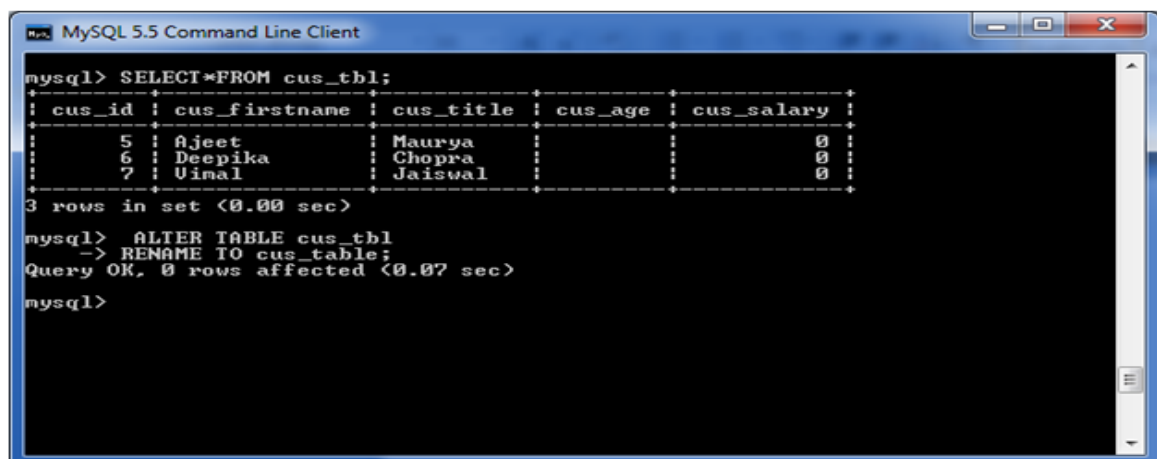
1. **ALTER TABLE** table_name
2. RENAME **TO** new_table_name;

Example:

In this example, the table name cus_tbl is renamed as cus_table.

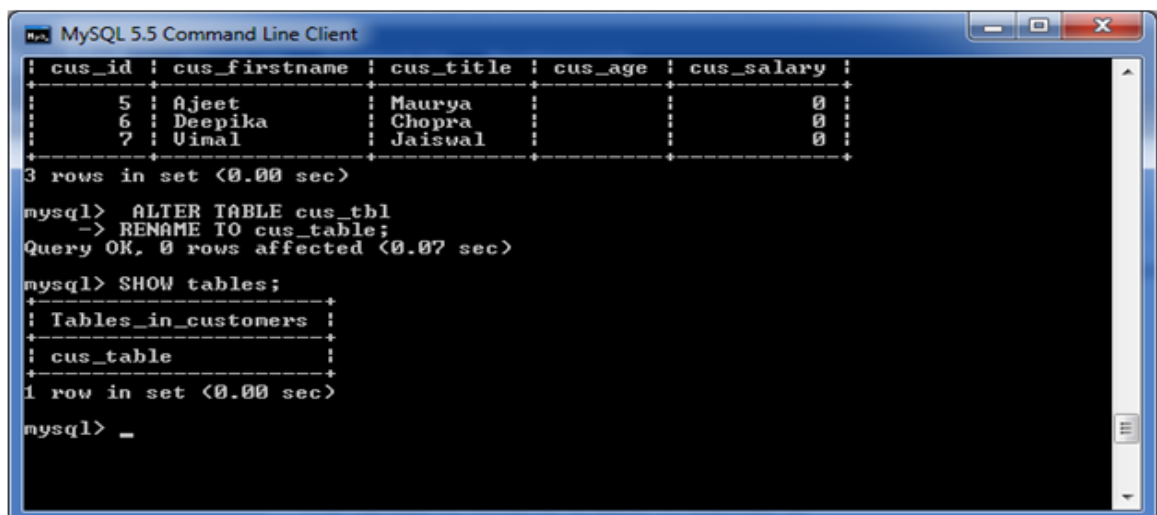
```
ALTER TABLE cus_tbl  
RENAME TO cus_table;
```

Output:



```
mysql> SELECT * FROM cus_tbl;  
+----+-----+-----+-----+-----+  
| cus_id | cus_firstname | cus_title | cus_age | cus_salary |  
+----+-----+-----+-----+-----+  
| 5 | Ajeet | Maurya | 0 | 0 |  
| 6 | Deepika | Chopra | 0 | 0 |  
| 7 | Uimal | Jaiswal | 0 | 0 |  
+----+-----+-----+-----+-----+  
3 rows in set (0.00 sec)  
  
mysql> ALTER TABLE cus_tbl  
-> RENAME TO cus_table;  
Query OK, 0 rows affected (0.07 sec)  
  
mysql>
```

See the renamed table:



```
mysql> SHOW tables;  
+-----+  
| Tables_in_customers |  
+-----+  
| cus_table |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> _
```


MySQL TRUNCATE Table

MYSQL TRUNCATE statement removes the complete data without removing its structure.

The TRUNCATE TABLE statement is used when you want to delete the complete data from a table without removing the table structure.

Syntax:

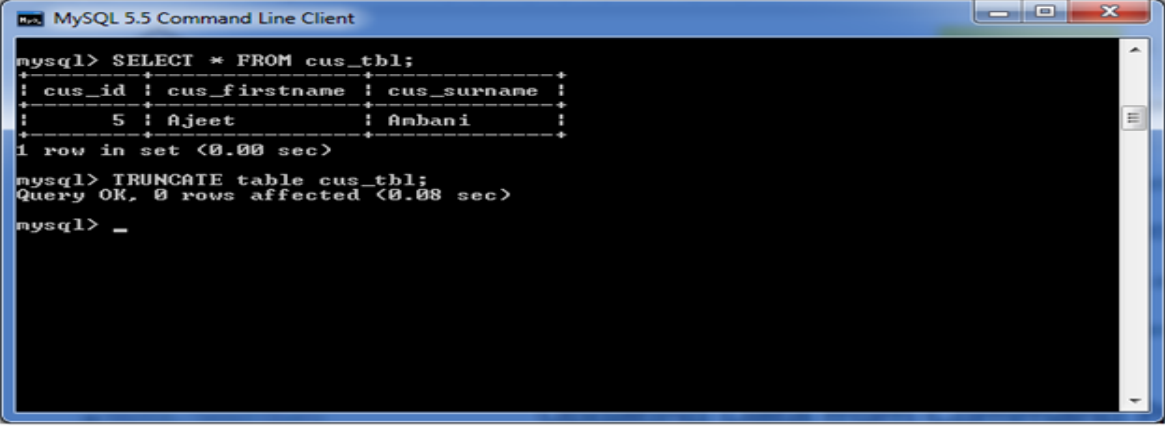
```
TRUNCATE TABLE table_name;
```

Example:

This example specifies how to truncate a table. In this example, we truncate the table "cus_tbl".

```
TRUNCATE TABLE cus_tbl;
```

Output:



```
mysql> SELECT * FROM cus_tbl;
+----+-----+-----+
| cus_id | cus_firstname | cus_surname |
+----+-----+-----+
|      5 | Ajeet        | Ambani      |
+----+-----+-----+
1 row in set (0.00 sec)

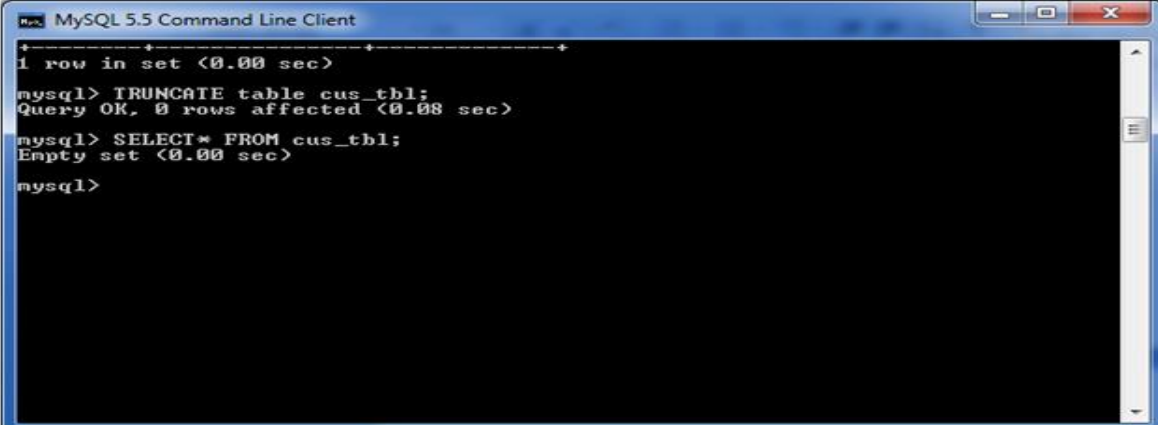
mysql> TRUNCATE table cus_tbl;
Query OK, 0 rows affected (0.08 sec)

mysql> _
```

See the table:

1. **SELECT* FROM** cus_tbl;

Output:



```
1 row in set (0.00 sec)

mysql> TRUNCATE table cus_tbl;
Query OK, 0 rows affected (0.08 sec)

mysql> SELECT* FROM cus_tbl;
Empty set (0.00 sec)

mysql>
```

MySQL DROP Table

MySQL DROP table statement removes the complete data with structure.

Syntax:

```
DROP TABLE table_name;
```

Example:

This example specifies how to drop a table. In this example, we are dropping the table "cus_tbl".

```
DROP TABLE cus_tbl;
```

MySQL TRUNCATE Table vs DROP Table

You can also use DROP TABLE command to delete complete table but it will remove complete table data and structure both. You need to re-create the table again if you have to store some data. But in the case of TRUNCATE TABLE, it removes only table data not structure. You don't need to re-create the table again because the table structure already exists.

MySQL View

In MySQL, View is a virtual table created by a query by joining one or more tables.

MySQL Create VIEW

A VIEW is created by SELECT statements. SELECT statements are used to take data from the source table to make a VIEW.

Syntax:

```
CREATE [OR REPLACE] VIEW view_name AS  
SELECT columns  
FROM tables  
[WHERE conditions];
```

Parameters:

OR REPLACE: It is optional. It is used when a VIEW already exist. If you do not specify this clause and the VIEW already exists, the CREATE VIEW statement will return an error.

view_name: It specifies the name of the VIEW that you want to create in MySQL.

WHERE conditions: It is also optional. It specifies the conditions that must be met for the records to be included in the VIEW.

The following example will create a VIEW name "trainer". This is a virtual table made by taking data from the table "courses".

```
CREATE VIEW trainer AS  
SELECT course_name, course_trainer  
FROM courses;
```

```
MySQL 5.5 Command Line Client
mysql> SHOW tables;
+-----+
| Tables_in_sssit |
+-----+
| courses          |
| students        |
+-----+
2 rows in set (0.00 sec)

mysql> CREATE VIEW trainer AS
-> SELECT course_name, course_trainer
-> FROM courses;
Query OK, 0 rows affected (0.10 sec)

mysql>
```

To see the created VIEW:

Syntax:

SELECT * FROM view_name;

Let's see how it looks the created VIEW:

SELECT * FROM trainer;

```
MySQL 5.5 Command Line Client
+-----+
| Tables_in_sssit |
+-----+
| courses          |
| students        |
+-----+
2 rows in set (0.00 sec)

mysql> CREATE VIEW trainer AS
-> SELECT course_name, course_trainer
-> FROM courses;
Query OK, 0 rows affected (0.10 sec)

mysql> SELECT*FROM trainer;
+-----+-----+
| course_name | course_trainer |
+-----+-----+
| Java        | Sonoo Jaiswal  |
| Hadoop      | R.K. Agrawal   |
| MongoDB     | Ajeet Maurya  |
+-----+-----+
3 rows in set (0.07 sec)

mysql>
```

MySQL Update VIEW

In MYSQL, the ALTER VIEW statement is used to modify or update the already created VIEW without dropping it.

Syntax:

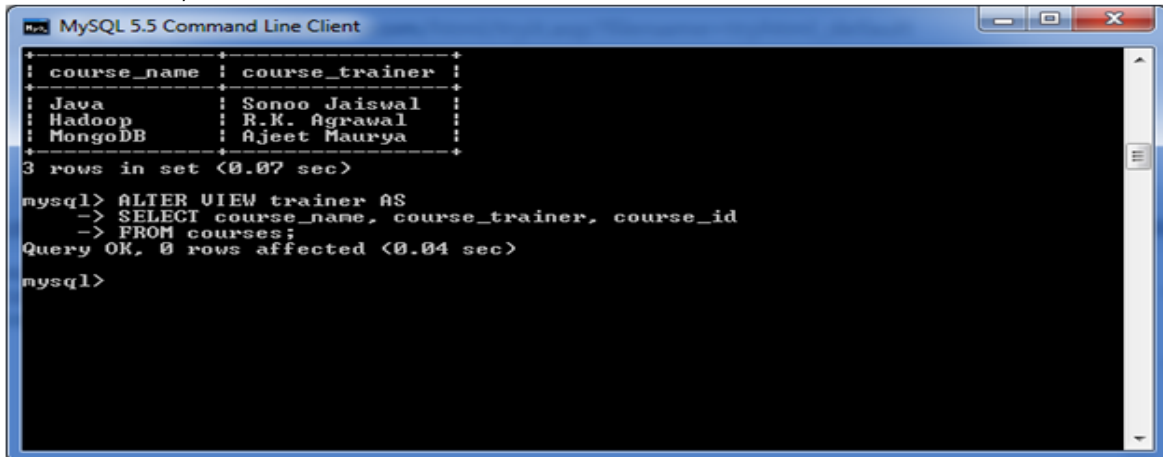
ALTER VIEW view_name **AS**
SELECT columns
FROM table
WHERE conditions;

Example:

The following example will alter the already created VIEW name "trainer" by adding a new column.

ALTER VIEW trainer AS

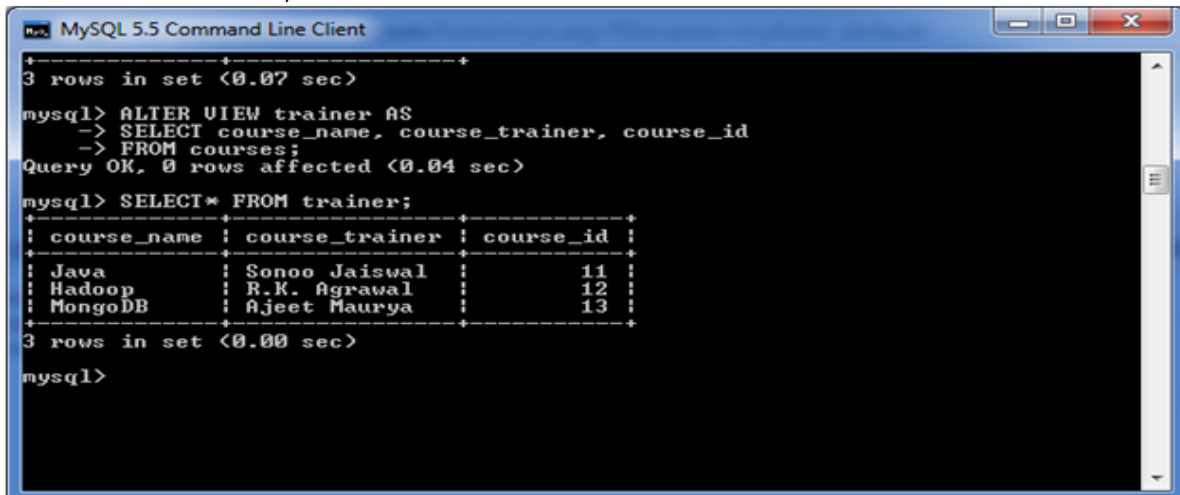
```
SELECT course_name, course_trainer, course_id  
FROM courses;
```



```
MySQL 5.5 Command Line Client  
+-----+  
| course_name | course_trainer |  
+-----+  
| Java        | Sonoo Jaiswal  |  
| Hadoop      | R.K. Agrawal   |  
| MongoDB     | Ajeet Maurya  |  
+-----+  
3 rows in set <0.07 sec>  
  
mysql> ALTER VIEW trainer AS  
-> SELECT course_name, course_trainer, course_id  
-> FROM courses;  
Query OK, 0 rows affected <0.04 sec>  
  
mysql>
```

To see the altered VIEW:

```
SELECT*FROM trainer;
```



```
MySQL 5.5 Command Line Client  
+-----+  
3 rows in set <0.07 sec>  
  
mysql> ALTER VIEW trainer AS  
-> SELECT course_name, course_trainer, course_id  
-> FROM courses;  
Query OK, 0 rows affected <0.04 sec>  
  
mysql> SELECT* FROM trainer;  
+-----+-----+-----+  
| course_name | course_trainer | course_id |  
+-----+-----+-----+  
| Java        | Sonoo Jaiswal  | 11        |  
| Hadoop      | R.K. Agrawal   | 12        |  
| MongoDB     | Ajeet Maurya  | 13        |  
+-----+-----+-----+  
3 rows in set <0.00 sec>  
  
mysql>
```

MySQL Drop VIEW

You can drop the VIEW by using the DROP VIEW statement.

Syntax:

```
DROP VIEW [IF EXISTS] view_name;
```

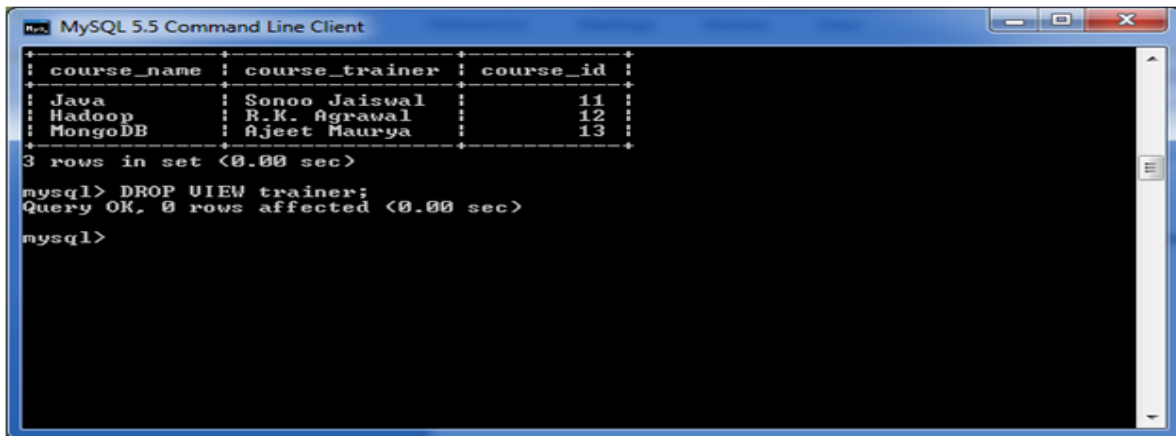
Parameters:

view_name: It specifies the name of the VIEW that you want to drop.

IF EXISTS: It is optional. If you do not specify this clause and the VIEW doesn't exist, the DROP VIEW statement will return an error.

Example:

```
DROP VIEW trainer;
```



```
MySQL 5.5 Command Line Client
+-----+-----+-----+
| course_name | course_trainer | course_id |
+-----+-----+-----+
| Java        | Sonoo Jaiswal  | 11        |
| Hadoop      | R.K. Agrawal   | 12        |
| MongoDB     | Ajeet Maurya  | 13        |
+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> DROP VIEW trainer;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

MySQL Queries

A list of commonly used MySQL queries to create database, use database, create table, insert record, update record, delete record, select record, truncate table and drop table are given below.

1) MySQL Create Database

MySQL create database is used to create database. For example

```
create database db1;
```

2) MySQL Select/Use Database

MySQL use database is used to select database. For example

```
use db1;
```

3) MySQL Create Query

MySQL create query is used to create a table, view, procedure and function. For example:

```
CREATE TABLE customers
(id int(10),
name varchar(50),
city varchar(50),
PRIMARY KEY (id ) );
```

4) MySQL Alter Query

MySQL alter query is used to add, modify, delete or drop columns of a table. Let's see a query to add column in customers table:

```
ALTER TABLE customers
ADD age varchar(50);
```

5) MySQL Insert Query

MySQL insert query is used to insert records into table. For example:

```
insert into customers values(101,'rahul','delhi');
```

6) MySQL Update Query

MySQL update query is used to update records of a table. For example:

```
update customers set name='bob', city='london' where id=101;
```

7) MySQL Delete Query

MySQL update query is used to delete records of a table from database. For example:

```
delete from customers where id=101;
```

8) MySQL Select Query

Oracle select query is used to fetch records from database. For example:

```
SELECT * from customers;
```

9) MySQL Truncate Table Query

MySQL update query is used to truncate or remove records of a table. It doesn't remove structure. For example:

```
truncate table customers;
```

10) MySQL Drop Query

MySQL drop query is used to drop a table, view or database. It removes structure and data of a table if you drop table. For example:

```
drop table customers;
```

MySQL INSERT Statement

MySQL INSERT statement is used to insert data in MySQL table within the database. We can insert single or multiple records using a single query in MySQL.

Syntax:

The SQL INSERT INTO command is used to insert data in MySQL table. Following is a generic syntax:

```
INSERT INTO table_name ( field1, field2,...fieldN )  
VALUES  
( value1, value2,...valueN );
```

Field name is optional. If you want to specify partial values, field name is mandatory.

Syntax for all fields:

```
INSERT INTO table_name VALUES ( value1, value2,...valueN );
```

MySQL INSERT Example 1: for all fields

If you have to store all the field values, either specify all field name or don't specify any field.

Example:

```
INSERT INTO emp VALUES (7, 'Sonoo', 40000);
```

Or,

```
INSERT INTO emp(id,name,salary) VALUES (7, 'Sonoo', 40000);
```

MySQL INSERT Example 2: for partial fields

In such case, it is mandatory to specify field names.

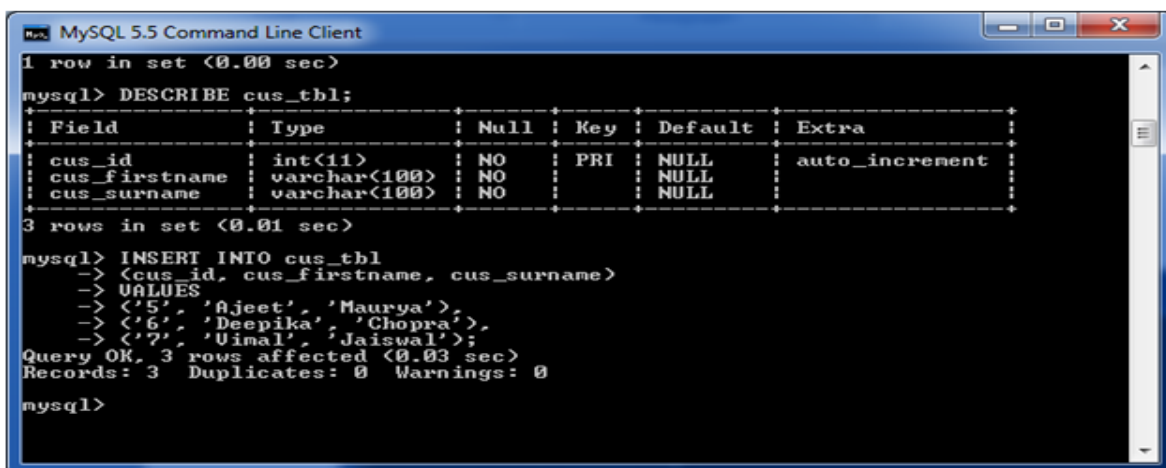
```
INSERT INTO emp(id,name) VALUES (7, 'Sonoo');
```

MySQL INSERT Example 3: inserting multiple records

Here, we are going to insert record in the "cus_tbl" table of "customers" database.

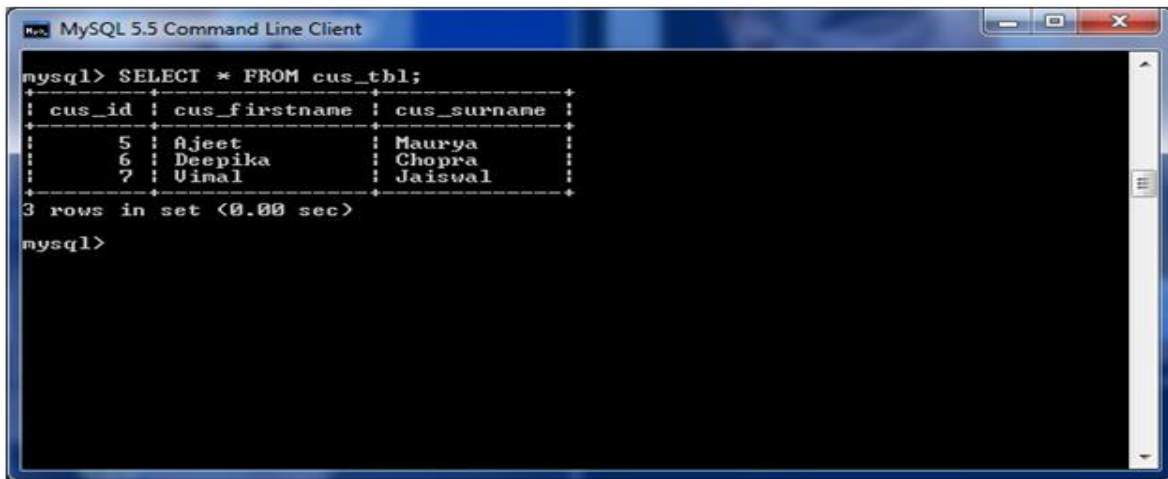
```
INSERT INTO cus_tbl  
(cus_id, cus_firstname, cus_surname)  
VALUES  
(5, 'Ajeet', 'Maurya'),  
(6, 'Deepika', 'Chopra'),  
(7, 'Vimal', 'Jaiswal');
```

Visual Representation:



```
MySQL 5.5 Command Line Client  
1 row in set (0.00 sec)  
mysql> DESCRIBE cus_tbl;  
+-----+-----+-----+-----+-----+-----+  
| Field          | Type          | Null | Key | Default | Extra          |  
+-----+-----+-----+-----+-----+-----+  
| cus_id         | int(11)       | NO   | PRI | NULL    | auto_increment |  
| cus_firstname  | varchar(100)  | NO   |     | NULL    |                |  
| cus_surname    | varchar(100)  | NO   |     | NULL    |                |  
+-----+-----+-----+-----+-----+-----+  
3 rows in set (0.01 sec)  
mysql> INSERT INTO cus_tbl  
-> (cus_id, cus_firstname, cus_surname)  
-> VALUES  
-> (5, 'Ajeet', 'Maurya'),  
-> (6, 'Deepika', 'Chopra'),  
-> (7, 'Vimal', 'Jaiswal');  
Query OK, 3 rows affected (0.03 sec)  
Records: 3 Duplicates: 0 Warnings: 0  
mysql>
```

See the data within the table by using the SELECT command:



```
mysql> SELECT * FROM cus_tbl;
+----+-----+-----+
| cus_id | cus_firstname | cus_surname |
+----+-----+-----+
| 5 | Ajeet | Maurya |
| 6 | Deepika | Chopra |
| 7 | Uinal | Jaiswal |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

MySQL UPDATE Query

MySQL UPDATE statement is used to update data of the MySQL table within the database. It is used when you need to modify the table.

Syntax:

Following is a generic syntax of UPDATE command to modify data into the MySQL table:

```
UPDATE table_name SET field1=new-value1, field2=new-value2  
[WHERE Clause]
```

Note:

- One or more field can be updated altogether.
- Any condition can be specified by using WHERE clause.
- You can update values in a single table at a time.
- WHERE clause is used to update selected rows in a table.

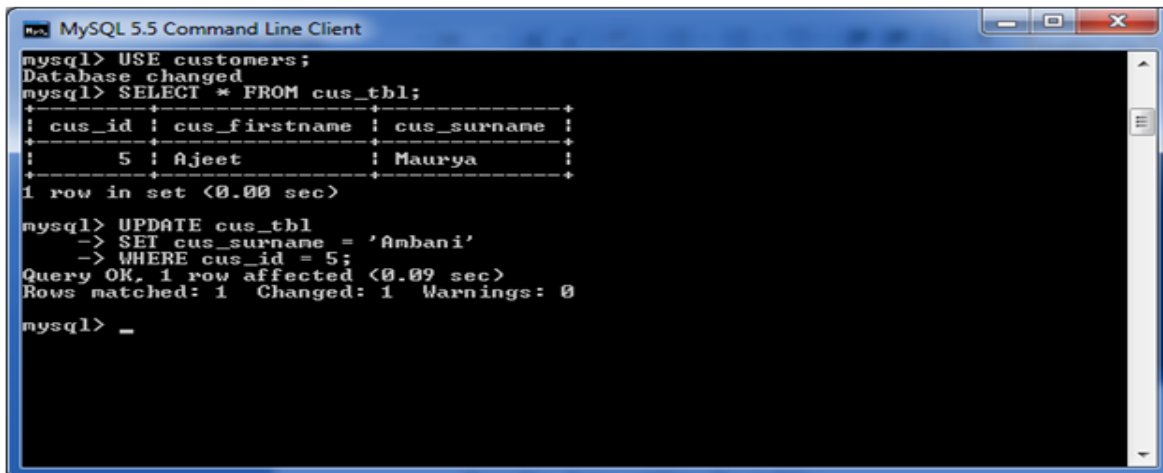
Example:

Here, we have a table "cus_tbl" within the database "customers". We are going to update the data within the table "cus_tbl".

This query will update cus_surname field for a record having cus_id as 5.

```
UPDATE cus_tbl  
SET cus_surname = 'Ambani'  
WHERE cus_id = 5;
```


Visual Representation:



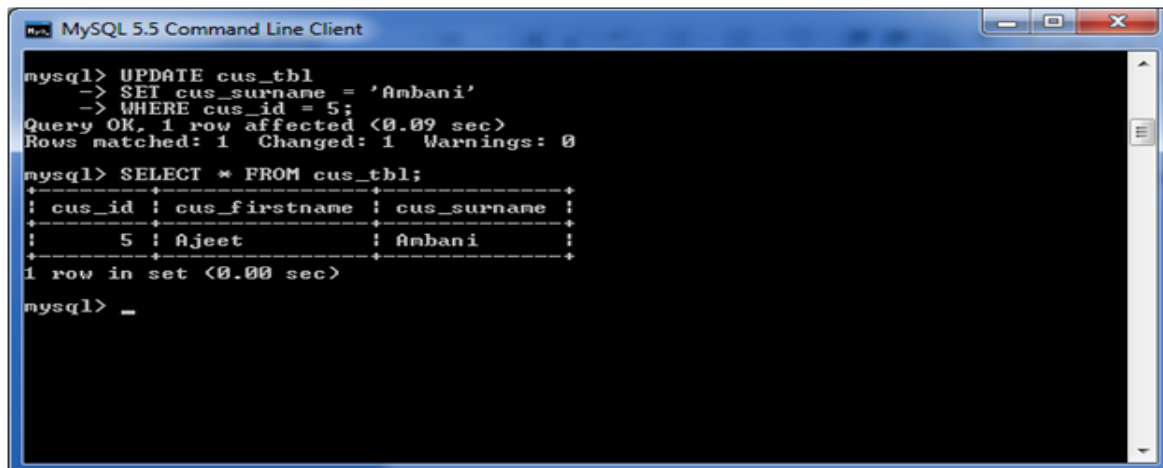
```
mysql> USE customers;
Database changed
mysql> SELECT * FROM cus_tbl;
+-----+-----+-----+
| cus_id | cus_firstname | cus_surname |
+-----+-----+-----+
|      5 | Ajeet        | Maurya      |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> UPDATE cus_tbl
-> SET cus_surname = 'Ambani'
-> WHERE cus_id = 5;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> _
mysql> _
```

Output by SELECT query:

1. **SELECT * FROM cus_tbl;**



```
mysql> UPDATE cus_tbl
-> SET cus_surname = 'Ambani'
-> WHERE cus_id = 5;
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM cus_tbl;
+-----+-----+-----+
| cus_id | cus_firstname | cus_surname |
+-----+-----+-----+
|      5 | Ajeet        | Ambani      |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql> _
mysql> _
```

Here, you can see that the table is updated as per your conditions.

MySQL DELETE Statement

MySQL DELETE statement is used to delete data from the MySQL table within the database. By using delete statement, we can delete records on the basis of conditions.


Syntax:

```
DELETE FROM table_name
WHERE
(Condition specified);
```

Example:

```
DELETE FROM cus_tbl
WHERE cus_id = 6;
```

Output:



```
mysql> SELECT * FROM cus_tbl;
+----+-----+-----+
| cus_id | cus_firstname | cus_surname |
+----+-----+-----+
| 5      | Ajeet        | Maurya      |
| 7      | Uimal        | Jaiswal     |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

MySQL SELECT Statement

The MySQL SELECT statement is used to fetch data from the one or more tables in MySQL. We can retrieve records of all fields or specified fields.

Syntax for specified fields:

```
SELECT expressions
FROM tables
[WHERE conditions];
```

Syntax for all fields:

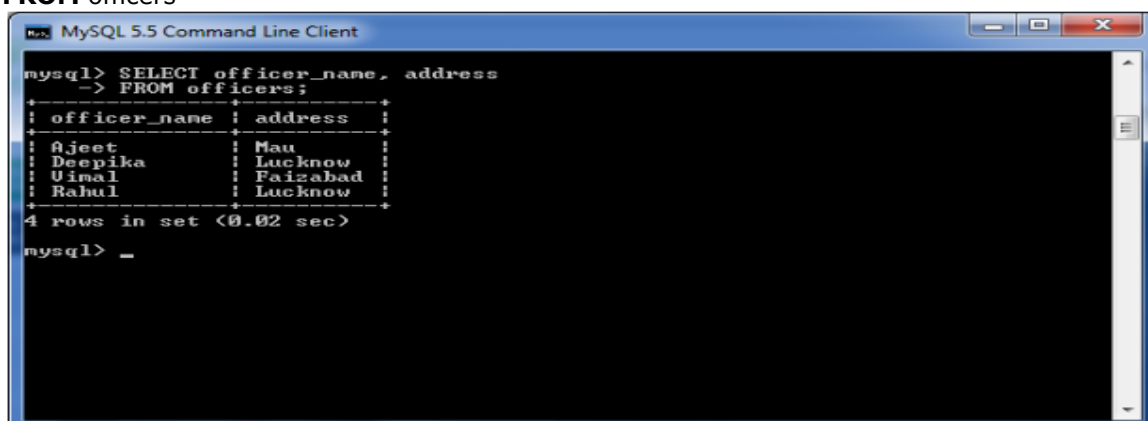
```
SELECT * FROM tables [WHERE conditions];
```

MySQL SELECT Example 1: for specified fields

In such case, it is mandatory to specify field names.

Example:

```
SELECT officer_name, address
FROM officers
```



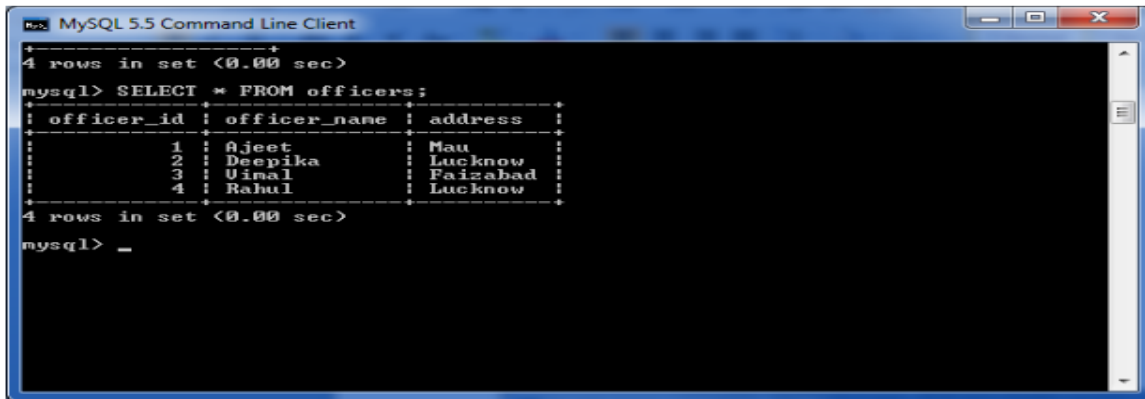
```
mysql> SELECT officer_name, address
-> FROM officers;
+-----+-----+
| officer_name | address |
+-----+-----+
| Ajeet        | Mau     |
| Deepika      | Lucknow|
| Uimal        | Faizabad|
| Rahul        | Lucknow|
+-----+-----+
4 rows in set (0.02 sec)

mysql> _
```

MySQL SELECT Example 2: for all fields

In such case, we can specify either all fields or * (asterisk) symbol.

SELECT * FROM officers



```
mysql> SELECT * FROM officers;
```

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Uimal	Faizabad
4	Rahul	Lucknow

```
mysql> _
```

MySQL SELECT Example 3: from multiple tables

MySQL WHERE Clause

MySQL WHERE Clause is used with SELECT, INSERT, UPDATE and DELETE clause to filter the results. It specifies a specific position where you have to do the operation.

Syntax:

WHERE conditions;

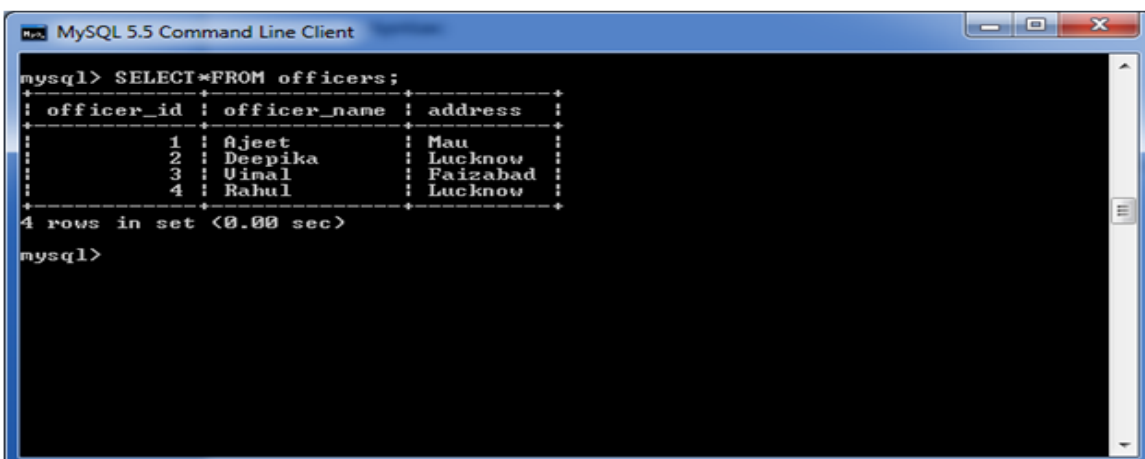
Parameter:

conditions: It specifies the conditions that must be fulfilled for records to be selected.

MySQL WHERE Clause with single condition

Let's take an example to retrieve data from a table "officers".

Table structure:



```
mysql> SELECT * FROM officers;
```

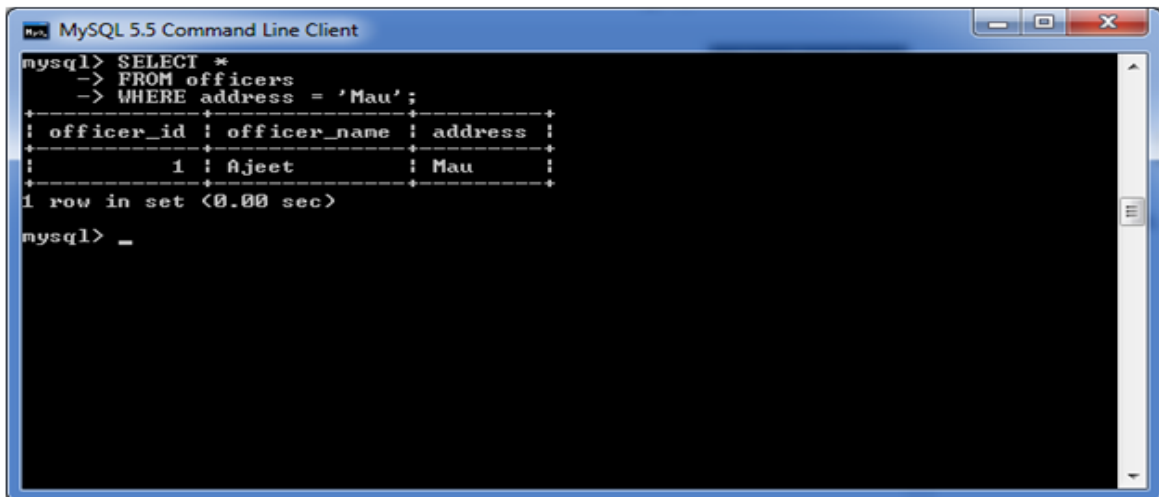
officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Uimal	Faizabad
4	Rahul	Lucknow

```
mysql>
```

Execute this query:

```
SELECT *  
FROM officers  
WHERE address = 'Mau';
```

Output:



```
mysql> SELECT *  
-> FROM officers  
-> WHERE address = 'Mau';  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
| 1 | Ajeet | Mau |  
+-----+-----+-----+  
1 row in set (0.00 sec)  
mysql> _
```

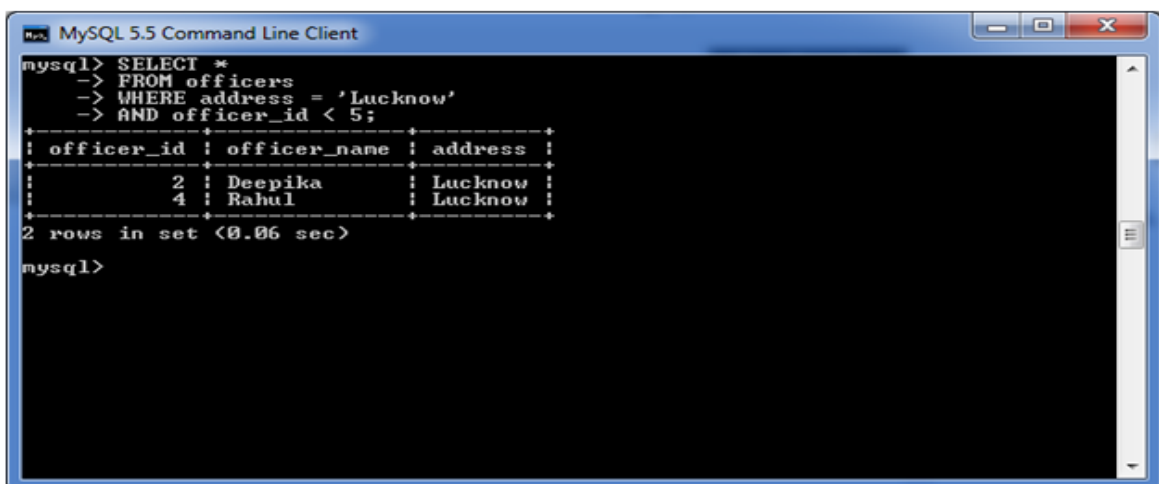
MySQL WHERE Clause with AND condition

In this example, we are retrieving data from the table "officers" with AND condition.

Execute the following query:

```
SELECT *  
FROM officers  
WHERE address = 'Lucknow'  
AND officer_id < 5;
```

Output:



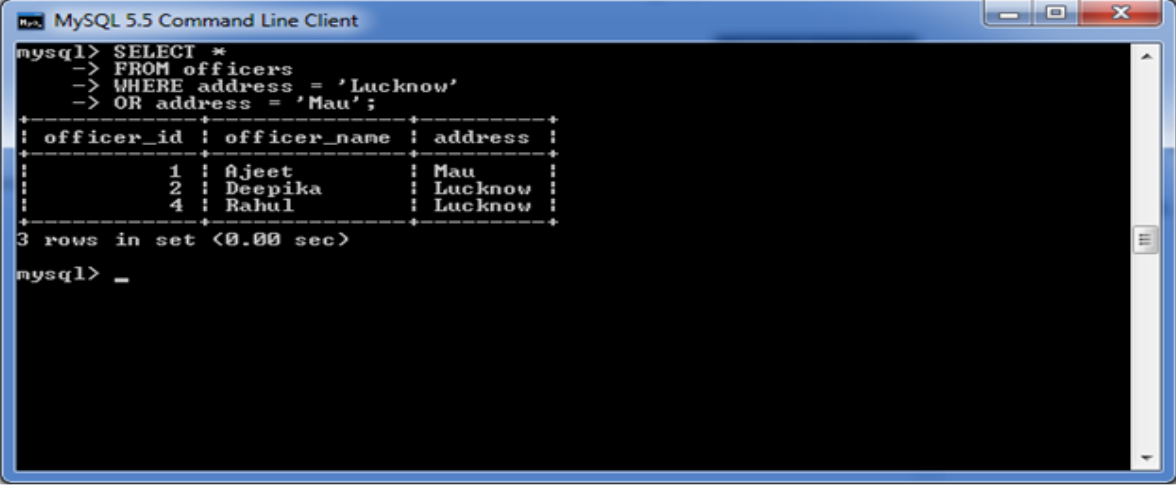
```
mysql> SELECT *  
-> FROM officers  
-> WHERE address = 'Lucknow'  
-> AND officer_id < 5;  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
| 2 | Deepika | Lucknow |  
| 4 | Rahul | Lucknow |  
+-----+-----+-----+  
2 rows in set (0.06 sec)  
mysql>
```

WHERE Clause with OR condition

Execute the following query:

```
SELECT *
FROM officers
WHERE address = 'Lucknow'
OR address = 'Mau';
```

Output:



```
mysql> SELECT *
-> FROM officers
-> WHERE address = 'Lucknow'
-> OR address = 'Mau';
+----+-----+-----+
| officer_id | officer_name | address |
+----+-----+-----+
| 1          | Ajeet       | Mau     |
| 2          | Deepika    | Lucknow |
| 4          | Rahul       | Lucknow |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

MySQL WHERE Clause with combination of AND & OR conditions

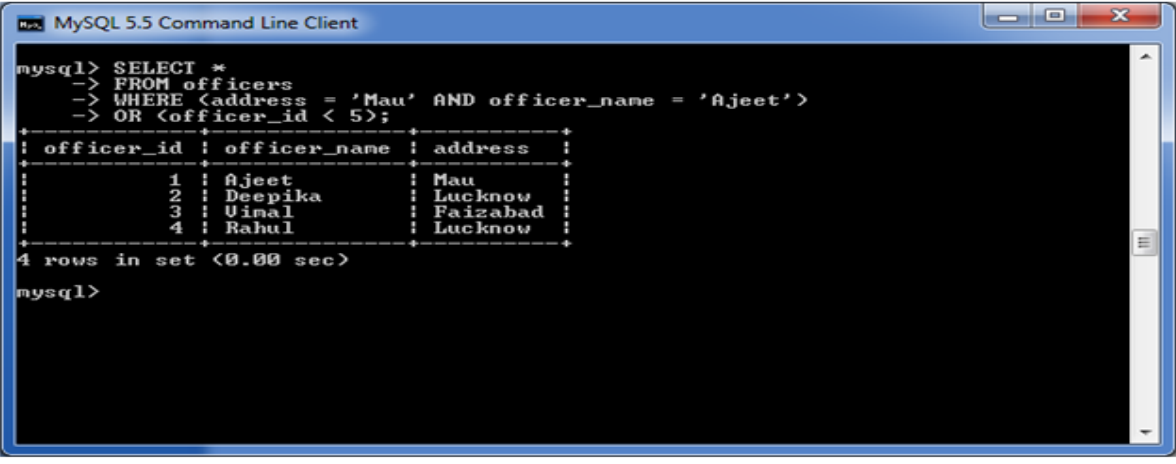
You can also use the AND & OR conditions altogether with the WHERE clause.

See this example:

Execute the following query:

```
SELECT *
FROM officers
WHERE (address = 'Mau' AND officer_name = 'Ajeet')
OR (officer_id < 5);
```

Output:



```
mysql> SELECT *
-> FROM officers
-> WHERE (address = 'Mau' AND officer_name = 'Ajeet')
-> OR (officer_id < 5);
+----+-----+-----+
| officer_id | officer_name | address |
+----+-----+-----+
| 1          | Ajeet       | Mau     |
| 2          | Deepika    | Lucknow |
| 3          | Uinal       | Faizabad |
| 4          | Rahul       | Lucknow |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

MySQL Distinct Clause

MySQL DISTINCT clause is used to remove duplicate records from the table and fetch only the unique records. The DISTINCT clause is only used with the SELECT statement.

Syntax:

```
SELECT DISTINCT expressions  
FROM tables  
[WHERE conditions];
```

Parameters

expressions: specify the columns or calculations that you want to retrieve.

tables: specify the name of the tables from where you retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be met for the records to be selected.

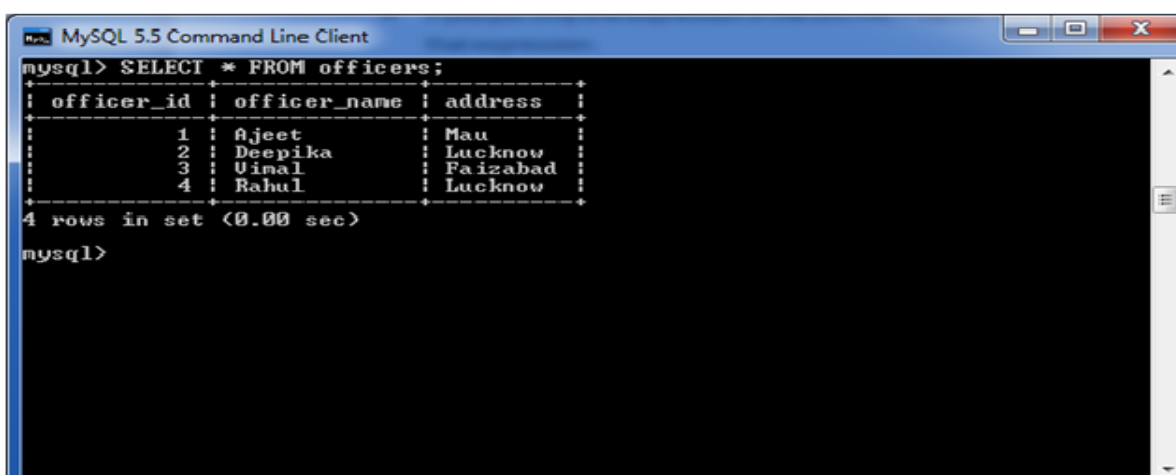
Note:

- If you put only one expression in the DISTINCT clause, the query will return the unique values for that expression.
- If you put more than one expression in the DISTINCT clause, the query will retrieve unique combinations for the expressions listed.
- In MySQL, the DISTINCT clause doesn't ignore NULL values. So if you are using the DISTINCT clause in your SQL statement, your result set will include NULL as a distinct value.

MySQL DISTINCT Clause with single expression

If you use a single expression then the MySQL DISTINCT clause will return a single field with unique records (no duplicate record).

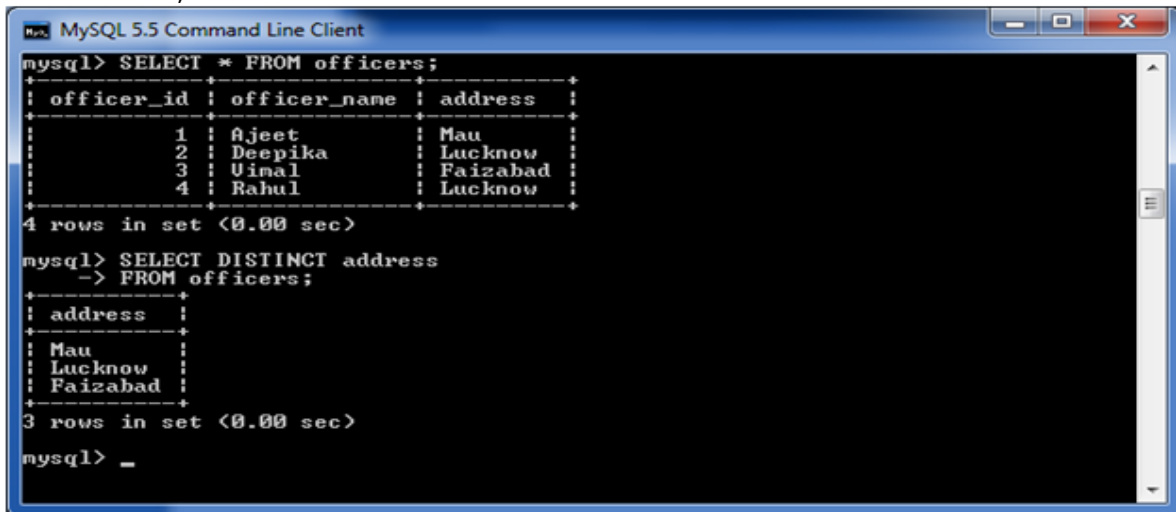
See the table:



```
mysql> SELECT * FROM officers;  
+----+-----+-----+  
| officer_id | officer_name | address |  
+----+-----+-----+  
| 1 | Ajeet | Mau |  
| 2 | Deepika | Lucknow |  
| 3 | Uinal | Faizabad |  
| 4 | Rahul | Lucknow |  
+----+-----+-----+  
4 rows in set (0.00 sec)  
mysql>
```

Use the following query:

SELECT DISTINCT address
FROM officers;



```
mysql> SELECT * FROM officers;
+----+-----+-----+
| officer_id | officer_name | address |
+----+-----+-----+
| 1          | Ajeet        | Mau     |
| 2          | Deepika      | Lucknow |
| 3          | Uimal        | Faizabad |
| 4          | Rahul        | Lucknow |
+----+-----+-----+
4 rows in set (0.00 sec)

mysql> SELECT DISTINCT address
-> FROM officers;
+-----+
| address |
+-----+
| Mau     |
| Lucknow |
| Faizabad |
+-----+
3 rows in set (0.00 sec)

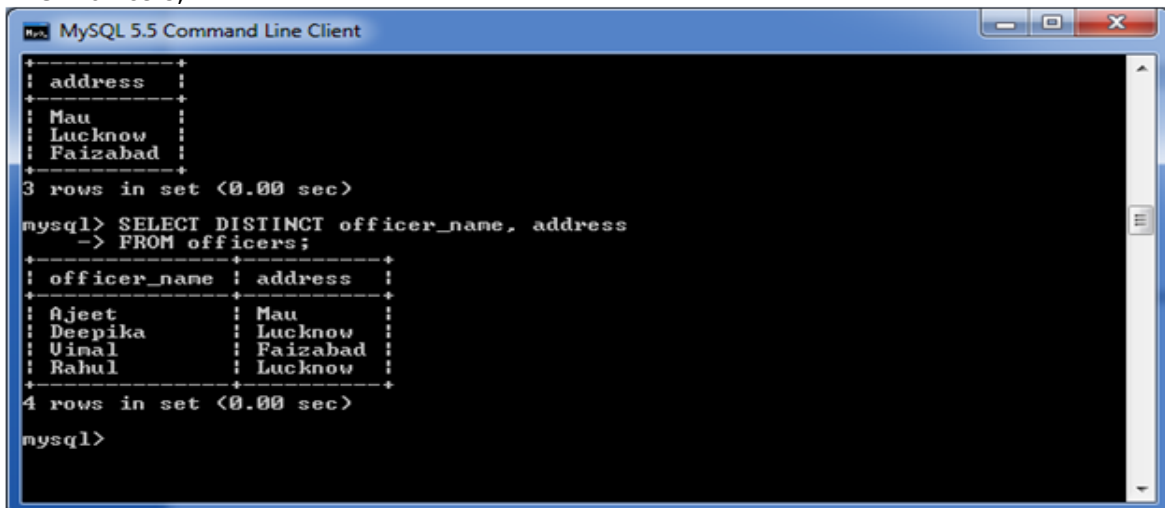
mysql> _
```

MySQL DISTINCT Clause with multiple expressions

If you use multiple expressions with DISTINCT Clause then MySQL DISTINCT clause will remove duplicates from more than one field in your SELECT statement.

Use the following query:

SELECT DISTINCT officer_name, address
FROM officers;



```
mysql> SELECT DISTINCT officer_name, address
-> FROM officers;
+-----+-----+
| officer_name | address |
+-----+-----+
| Ajeet        | Mau     |
| Deepika      | Lucknow |
| Uimal        | Faizabad |
| Rahul        | Lucknow |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

MySQL FROM Clause

The MySQL FROM Clause is used to select some records from a table. It can also be used to retrieve records from multiple tables using JOIN condition.

Syntax:

FROM table1
[{ **INNER JOIN** | **LEFT** [OUTER] **JOIN**| **RIGHT** [OUTER] **JOIN** } table2
ON table1.column1 = table2.column1]

Parameters

table1 and table2: specify tables used in the MySQL statement. The two tables are joined based on table1.column1 = table2.column1.

Note:

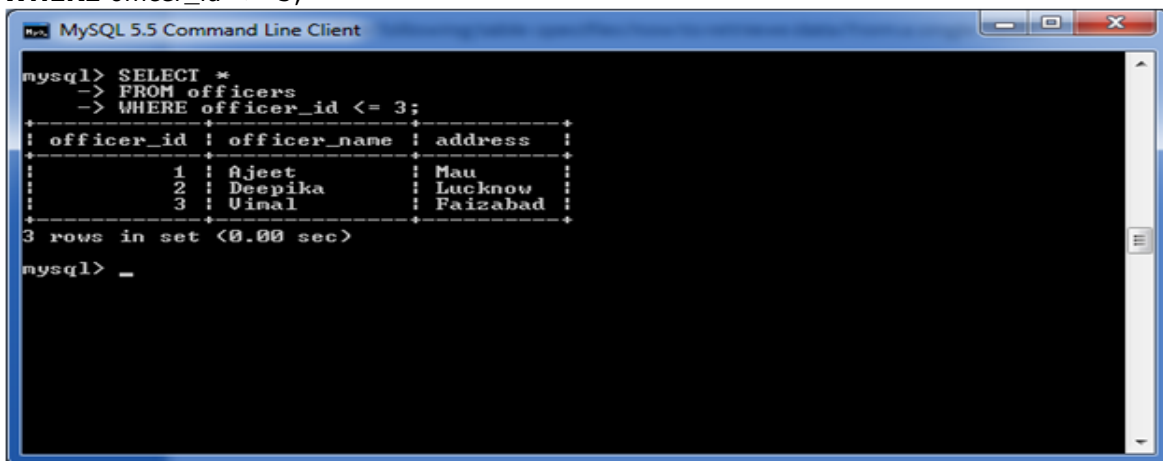
- If you are using the FROM clause in a MySQL statement then at least one table must have been selected.
- If you are using two or more tables in the MySQL FROM clause, these tables are generally joined using INNER or OUTER joins.

MySQL FROM Clause: Retrieve data from one table

The following query specifies how to retrieve data from a single table.

Use the following Query:

```
SELECT *  
FROM officers  
WHERE officer_id <= 3;
```



The screenshot shows a MySQL 5.5 Command Line Client window. The command prompt shows the following query being executed:

```
mysql> SELECT *  
-> FROM officers  
-> WHERE officer_id <= 3;
```

The output is a table with three columns: officer_id, officer_name, and address. The data is as follows:

officer_id	officer_name	address
1	Ajeet	Mau
2	Deepika	Lucknow
3	Uimal	Faizabad

Below the table, it says "3 rows in set <0.00 sec>". The prompt then shows "mysql> _".

MySQL FROM Clause: Retrieve data from two tables with inner join

MySQL FROM Clause: Retrieve data from two tables using outer join

MySQL ORDER BY Clause

The MYSQL ORDER BY Clause is used to sort the records in ascending or descending order.

Syntax:

```
SELECT expressions  
FROM tables  
[WHERE conditions]  
ORDER BY expression [ ASC | DESC ];
```

Parameters

expressions: It specifies the columns that you want to retrieve.

tables: It specifies the tables, from where you want to retrieve records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies conditions that must be fulfilled for the records to be selected.

ASC: It is optional. It sorts the result set in ascending order by expression (default, if no modifier is provided).

DESC: It is also optional. It sorts the result set in descending order by expression.

Note: You can use MySQL ORDER BY clause in a SELECT statement, SELECT LIMIT statement, and DELETE LIMIT statement.

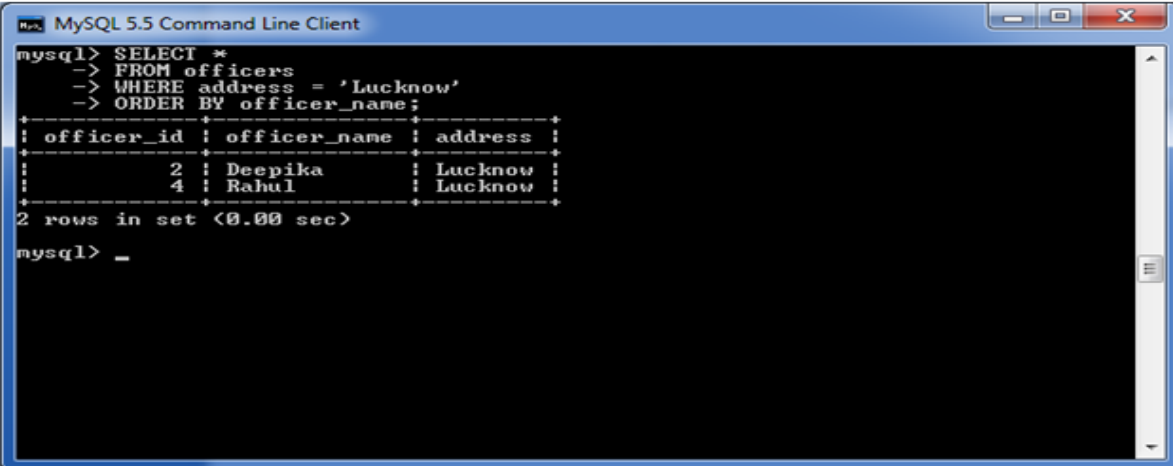
MySQL ORDER BY: without using ASC/DESC attribute

If you use MySQL ORDER BY clause without specifying the ASC and DESC modifier then by default you will get the result in ascending order.

Execute the following query:

```
SELECT *
FROM officers
WHERE address = 'Lucknow'
ORDER BY officer_name;
```

Output:



```
mysql> SELECT *
-> FROM officers
-> WHERE address = 'Lucknow'
-> ORDER BY officer_name;
+-----+-----+-----+
| officer_id | officer_name | address |
+-----+-----+-----+
| 2 | Deepika | Lucknow |
| 4 | Rahul | Lucknow |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql> _
```

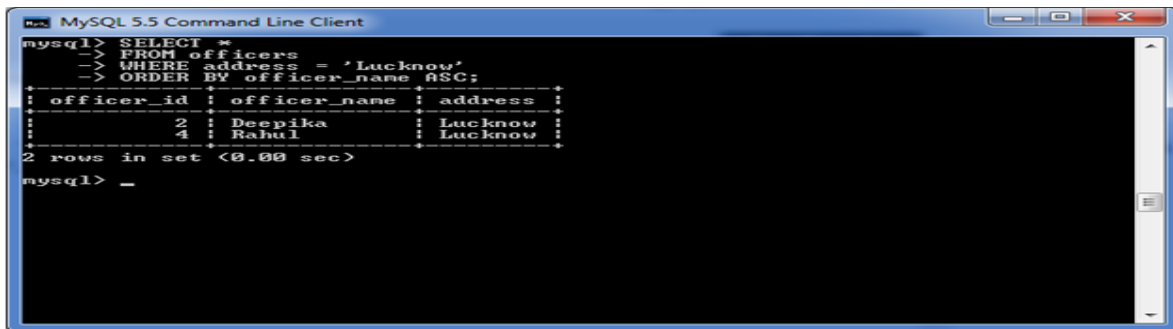
MySQL ORDER BY: with ASC attribute

Let's take an example to retrieve the data in ascending order.

Execute the following query:

```
SELECT *
FROM officers
WHERE address = 'Lucknow'
ORDER BY officer_name ASC;
```

Output:

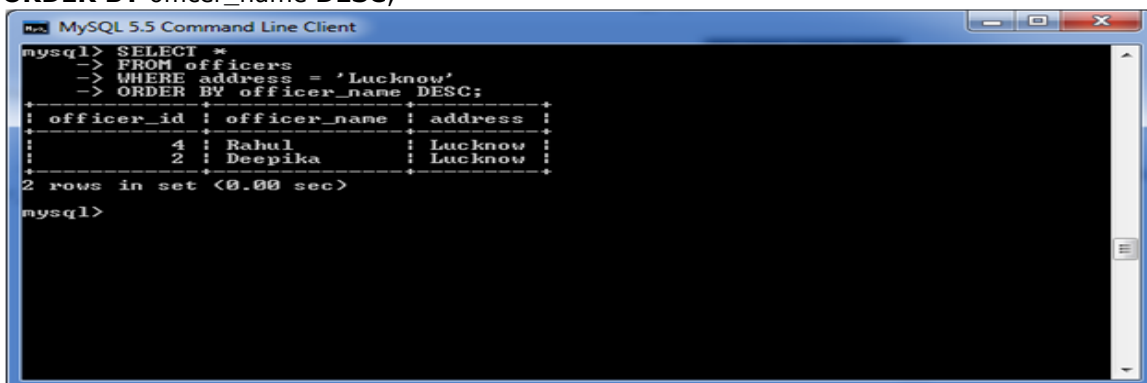


```
mysql> SELECT *
-> FROM officers
-> WHERE address = 'Lucknow'
-> ORDER BY officer_name ASC;
+-----+-----+-----+
| officer_id | officer_name | address |
+-----+-----+-----+
| 2          | Deepika      | Lucknow |
| 4          | Rahul        | Lucknow |
+-----+-----+-----+
2 rows in set <0.00 sec>

mysql> _
```

MySQL ORDER BY: with DESC attribute

```
SELECT *
FROM officers
WHERE address = 'Lucknow'
ORDER BY officer_name DESC;
```



```
mysql> SELECT *
-> FROM officers
-> WHERE address = 'Lucknow'
-> ORDER BY officer_name DESC;
+-----+-----+-----+
| officer_id | officer_name | address |
+-----+-----+-----+
| 4          | Rahul        | Lucknow |
| 2          | Deepika      | Lucknow |
+-----+-----+-----+
2 rows in set <0.00 sec>

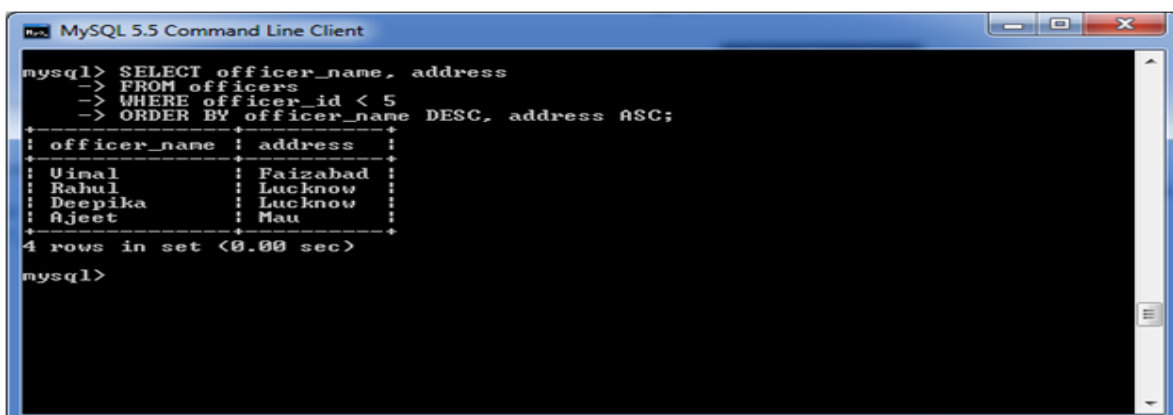
mysql>
```

MySQL ORDER BY: using both ASC and DESC attributes

Execute the following query:

```
SELECT officer_name, address
FROM officers
WHERE officer_id < 5
ORDER BY officer_name DESC, address ASC;
```

Output:



```
mysql> SELECT officer_name, address
-> FROM officers
-> WHERE officer_id < 5
-> ORDER BY officer_name DESC, address ASC;
+-----+-----+
| officer_name | address |
+-----+-----+
| Uinal        | Faizabad |
| Rahul        | Lucknow  |
| Deepika      | Lucknow  |
| Ajeet        | Mau      |
+-----+-----+
4 rows in set <0.00 sec>

mysql>
```

MySQL GROUP BY Clause

The MySQL GROUP BY Clause is used to collect data from multiple records and group the result by one or more column. It is generally used in a SELECT statement.

You can also use some aggregate functions like COUNT, SUM, MIN, MAX, AVG etc. on the grouped column.

Syntax:

```
SELECT expression1, expression2, ... expression_n,  
aggregate_function (expression)  
FROM tables  
[WHERE conditions]  
GROUP BY expression1, expression2, ... expression_n;
```

Parameters

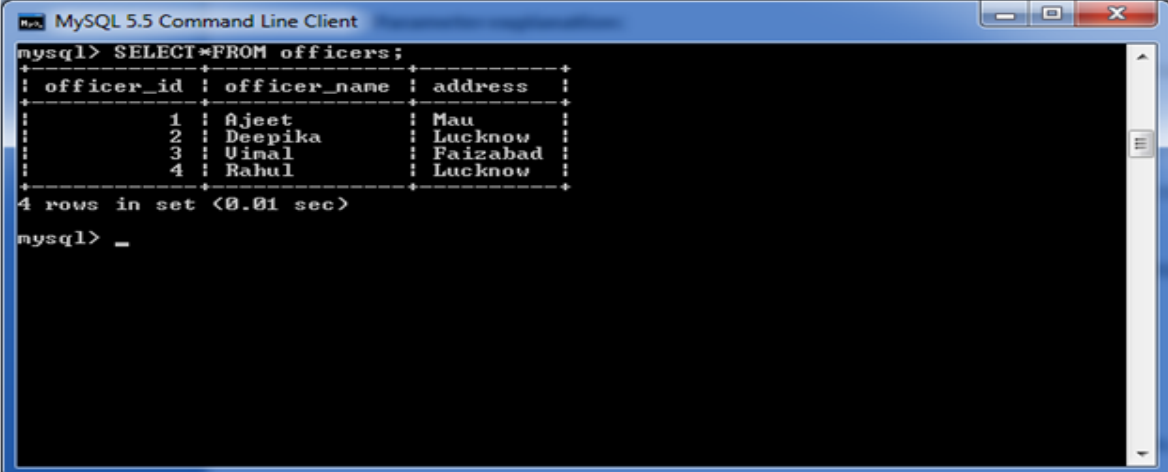
expression1, expression2, ... expression_n: It specifies the expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY clause.

aggregate_function: It specifies a function such as SUM, COUNT, MIN, MAX, or AVG etc. tables: It specifies the tables, from where you want to retrieve the records. There must be at least one table listed in the FROM clause.

WHERE conditions: It is optional. It specifies the conditions that must be fulfilled for the records to be selected.

(i) MySQL GROUP BY Clause with COUNT function

Consider a table named "officers" table, having the following records.



```
mysql> SELECT * FROM officers;  
+-----+-----+-----+  
| officer_id | officer_name | address |  
+-----+-----+-----+  
| 1 | Ajeet | Mau |  
| 2 | Deepika | Lucknow |  
| 3 | Uimal | Faizabad |  
| 4 | Rahul | Lucknow |  
+-----+-----+-----+  
4 rows in set (0.01 sec)  
mysql> _
```

Now, let's count repetitive number of cities in the column address.

Execute the following query:

```
SELECT address, COUNT(*)  
FROM officers  
GROUP BY address;
```

Output:

```
MySQL 5.5 Command Line Client
+----+-----+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Uinal | Faizabad |
| 4 | Rahul | Lucknow |
+----+-----+-----+
4 rows in set <0.01 sec>

mysql> SELECT address, COUNT(*)
-> FROM officers
-> GROUP BY address;
+----+-----+
| address | COUNT(*) |
+----+-----+
| Faizabad | 1 |
| Lucknow | 2 |
| Mau | 1 |
+----+-----+
3 rows in set <0.04 sec>

mysql>
```

(ii) MySQL GROUP BY Clause with SUM function

Let's take a table "employees" table, having the following data.

```
MySQL 5.5 Command Line Client
-> <3, 'Milan', '2015-01-25', 9>
-> <1, 'Ajeet', '2015-01-26', 12>
-> <3, 'Milan', '2015-01-26', 9>
Query OK, 10 rows affected <0.06 sec>
Records: 10 Duplicates: 0 Warnings: 0

mysql>
mysql> SELECT * FROM employees;
+----+-----+-----+-----+
| emp_id | emp_name | working_date | working_hours |
+----+-----+-----+-----+
| 1 | Ajeet | 2015-01-24 | 12 |
| 2 | Ayan | 2015-01-24 | 10 |
| 3 | Milan | 2015-01-24 | 9 |
| 4 | Ruchi | 2015-01-24 | 6 |
| 1 | Ajeet | 2015-01-25 | 12 |
| 2 | Ayan | 2015-01-25 | 10 |
| 4 | Ruchi | 2015-01-25 | 6 |
| 3 | Milan | 2015-01-25 | 9 |
| 1 | Ajeet | 2015-01-26 | 12 |
| 3 | Milan | 2015-01-26 | 9 |
+----+-----+-----+-----+
10 rows in set <0.00 sec>

mysql>
```

Now, the following query will GROUP BY the example using the SUM function and return the emp_name and total working hours of each employee.

Execute the following query:

```
SELECT emp_name, SUM(working_hours) AS "Total working hours"
FROM employees
GROUP BY emp_name;
```

Output:

```
MySQL 5.5 Command Line Client
+----+-----+-----+-----+
| 1 | Ajeet | 2015-01-26 | 12 |
| 3 | Milan | 2015-01-26 | 9 |
+----+-----+-----+-----+
10 rows in set <0.00 sec>

mysql>
mysql> SELECT emp_name, SUM(working_hours) AS "Total working hours"
-> FROM employees
-> GROUP BY name;
ERROR 1054 (42S22): Unknown column 'name' in 'group statement'

mysql>
mysql> SELECT emp_name, SUM(working_hours) AS "Total working hours"
-> FROM employees
-> GROUP BY emp_name;
+----+-----+-----+
| emp_name | Total working hours |
+----+-----+-----+
| Ajeet | 36 |
| Ayan | 20 |
| Milan | 27 |
| Ruchi | 12 |
+----+-----+-----+
4 rows in set <0.00 sec>

mysql> _
```

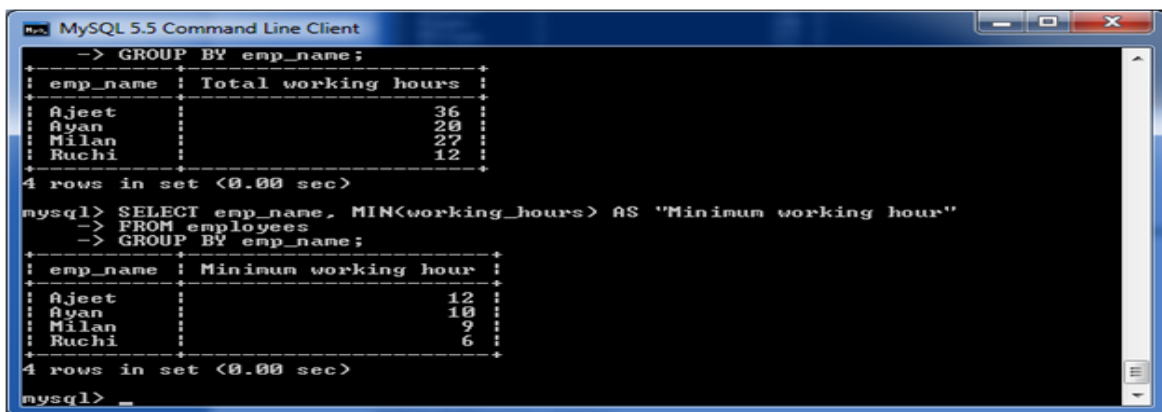
(iii) MySQL GROUP BY Clause with MIN function

The following example specifies the minimum working hours of the employees from the table "employees".

Execute the following query:

```
SELECT emp_name, MIN(working_hours) AS "Minimum working hour"  
FROM employees  
GROUP BY emp_name;
```

Output:



```
mysql> SELECT emp_name, MIN(working_hours) AS "Minimum working hour"  
-> FROM employees  
-> GROUP BY emp_name;  
+-----+-----+  
| emp_name | Minimum working hour |  
+-----+-----+  
| Ajeet   | 12                    |  
| Ayan    | 10                    |  
| Milan   | 9                     |  
| Ruchi   | 6                     |  
+-----+-----+  
4 rows in set <0.00 sec>  
mysql>
```

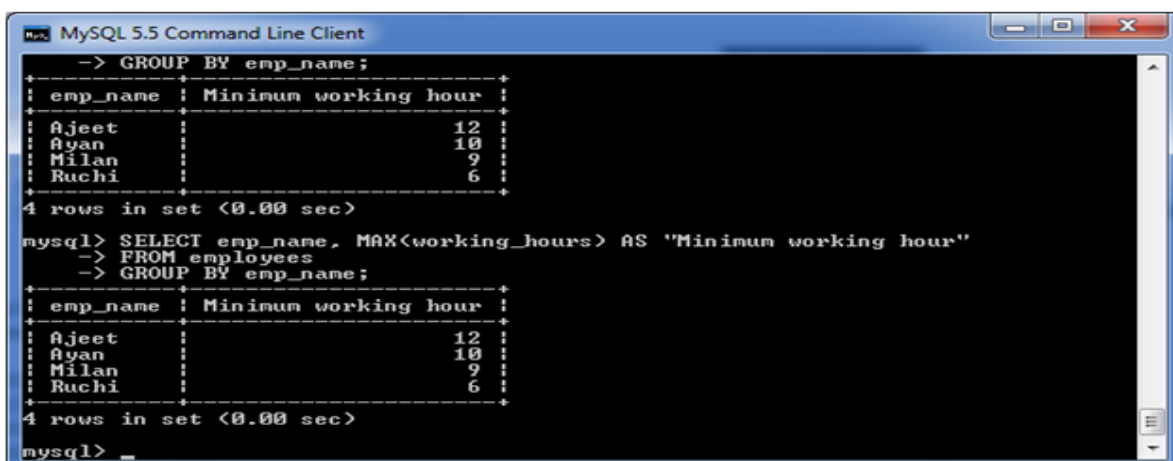
(iv) MySQL GROUP BY Clause with MAX function

The following example specifies the maximum working hours of the employees from the table "employees".

Execute the following query:

```
SELECT emp_name, MAX(working_hours) AS "Minimum working hour"  
FROM employees  
GROUP BY emp_name;
```

Output:



```
mysql> SELECT emp_name, MAX(working_hours) AS "Minimum working hour"  
-> FROM employees  
-> GROUP BY emp_name;  
+-----+-----+  
| emp_name | Minimum working hour |  
+-----+-----+  
| Ajeet   | 12                    |  
| Ayan    | 10                    |  
| Milan   | 9                     |  
| Ruchi   | 6                     |  
+-----+-----+  
4 rows in set <0.00 sec>  
mysql>
```

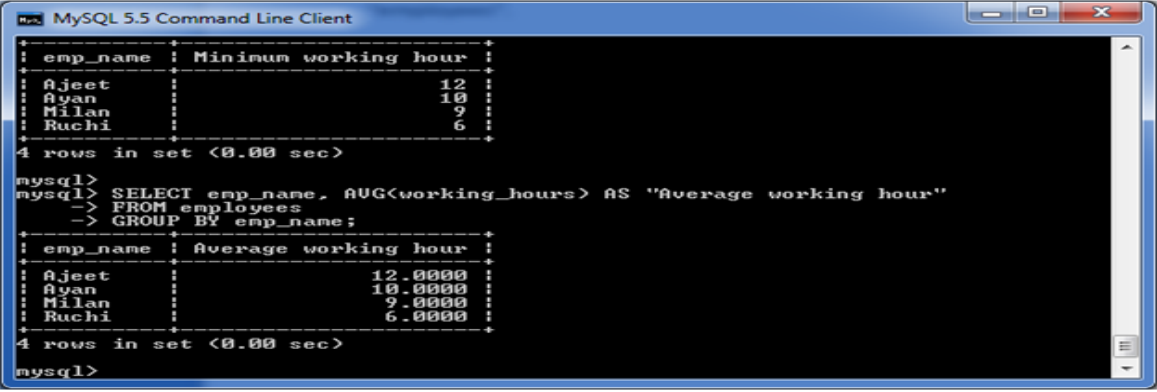
(v) MySQL GROUP BY Clause with AVG function

The following example specifies the average working hours of the employees from the table "employees".

Execute the following query:

```
SELECT emp_name, AVG(working_hours) AS "Average working hour"
FROM employees
GROUP BY emp_name;
```

Output:



```
mysql>
+-----+-----+
| emp_name | Minimum working hour |
+-----+-----+
| Ajeet    | 12                    |
| Ayan     | 10                    |
| Milan    | 9                    |
| Ruchi    | 6                    |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
mysql> SELECT emp_name, AVG(working_hours) AS "Average working hour"
-> FROM employees
-> GROUP BY emp_name;
+-----+-----+
| emp_name | Average working hour |
+-----+-----+
| Ajeet    | 12.0000              |
| Ayan     | 10.0000              |
| Milan    | 9.0000               |
| Ruchi    | 6.0000               |
+-----+-----+
4 rows in set (0.00 sec)

mysql>
```

MySQL HAVING Clause

MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE.

Syntax:

```
SELECT expression1, expression2, ... expression_n,
aggregate_function (expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n
HAVING condition;
```

Parameters

aggregate_function: It specifies any one of the aggregate function such as SUM, COUNT, MIN, MAX, or AVG.

expression1, expression2, ... expression_n: It specifies the expressions that are not encapsulated within an aggregate function and must be included in the GROUP BY clause.

WHERE conditions: It is optional. It specifies the conditions for the records to be selected.

HAVING condition: It is used to restrict the groups of returned rows. It shows only those groups in result set whose conditions are TRUE.

HAVING Clause with SUM function

Consider a table "employees" table having the following data.

```

MySQL 5.5 Command Line Client
-> <3, 'Milan', '2015-01-25', 9>
-> <1, 'Ajeet', '2015-01-26', 12>
-> <3, 'Milan', '2015-01-26', 9>
Query OK, 10 rows affected (0.06 sec)
Records: 10 Duplicates: 0 Warnings: 0

mysql>
mysql> SELECT * FROM employees;
+----+-----+-----+-----+
| emp_id | emp_name | working_date | working_hours |
+----+-----+-----+-----+
| 1 | Ajeet | 2015-01-24 | 12 |
| 2 | Ayan | 2015-01-24 | 10 |
| 3 | Milan | 2015-01-24 | 9 |
| 4 | Ruchi | 2015-01-24 | 6 |
| 1 | Ajeet | 2015-01-25 | 12 |
| 2 | Ayan | 2015-01-25 | 10 |
| 4 | Ruchi | 2015-01-25 | 6 |
| 3 | Milan | 2015-01-25 | 9 |
| 1 | Ajeet | 2015-01-26 | 12 |
| 3 | Milan | 2015-01-26 | 9 |
+----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql>

```

Here, we use the SUM function with the HAVING Clause to return the emp_name and sum of their working hours.

Execute the following query:

```

SELECT emp_name, SUM(working_hours) AS "Total working hours"
FROM employees
GROUP BY emp_name
HAVING SUM(working_hours) > 5;

```

```

MySQL 5.5 Command Line Client
+----+-----+-----+-----+
| emp_id | emp_name | working_date | working_hours |
+----+-----+-----+-----+
| 1 | Ajeet | 2015-01-24 | 12 |
| 2 | Ayan | 2015-01-24 | 10 |
| 3 | Milan | 2015-01-24 | 9 |
| 4 | Ruchi | 2015-01-24 | 6 |
| 1 | Ajeet | 2015-01-25 | 12 |
| 2 | Ayan | 2015-01-25 | 10 |
| 4 | Ruchi | 2015-01-25 | 6 |
| 3 | Milan | 2015-01-25 | 9 |
| 1 | Ajeet | 2015-01-26 | 12 |
| 3 | Milan | 2015-01-26 | 9 |
+----+-----+-----+-----+
10 rows in set (0.00 sec)

mysql> SELECT emp_name, SUM(working_hours) AS "Total working hours"
-> FROM employees
-> GROUP BY emp_name
-> HAVING SUM(working_hours) > 5;
+-----+-----+
| emp_name | Total working hours |
+-----+-----+
| Ajeet | 36 |
| Ayan | 20 |
| Milan | 27 |
| Ruchi | 12 |
+-----+-----+
4 rows in set (0.00 sec)

mysql> _

```

Simply, it can also be used with COUNT, MIN, MAX and AVG functions.

MySQL Conditions

MySQL AND MySQL OR MySQL AND OR MySQL LIKE MySQL IN MySQL NOT MySQL IS NULL MySQL IS NOT NULL MySQL BETWEEN

MySQL Join

MySQL JOINS

MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.

There are three types of MySQL joins:

- MySQL INNER JOIN (or sometimes called simple join)
- MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
- MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

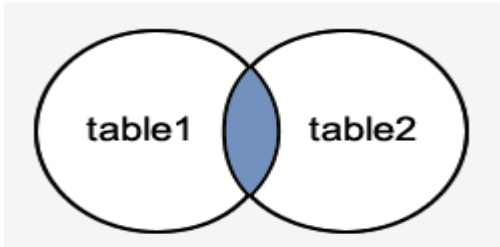
MySQL Inner JOIN (Simple Join)

The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.

Syntax:

SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data.

```
MySQL 5.5 Command Line Client
4 rows in set <0.00 sec>
mysql> SELECT * FROM officers;
+----+-----+-----+
| officer_id | officer_name | address |
+----+-----+-----+
| 1          | Ajeet        | Mau     |
| 2          | Deepika      | Lucknow|
| 3          | Uimal        | Faizabad|
| 4          | Rahul        | Lucknow|
+----+-----+-----+
4 rows in set <0.00 sec>
mysql> SELECT * FROM students;
+----+-----+-----+
| student_id | student_name | course_name |
+----+-----+-----+
| 1          | Aryan        | Java        |
| 2          | Rohini       | Hadoop      |
| 3          | Lallu        | MongoDB     |
+----+-----+-----+
3 rows in set <0.00 sec>
mysql> _
```

Execute the following query:

SELECT officers.officer_name, officers.address, students.course_name
FROM officers
INNER JOIN students
ON officers.officer_id = students.student_id;

Output:

```
MySQL 5.5 Command Line Client
mysql> SELECT officers.officer_name, officers.address, students.course_name
-> FROM officers
-> INNER JOIN students
-> ON officers.officer_id = students.student_id;
+----+-----+-----+
| officer_name | address | course_name |
+----+-----+-----+
| Ajeet        | Mau     | Java        |
| Deepika      | Lucknow| Hadoop      |
| Uimal        | Faizabad| MongoDB     |
+----+-----+-----+
3 rows in set <0.00 sec>
mysql> _
```

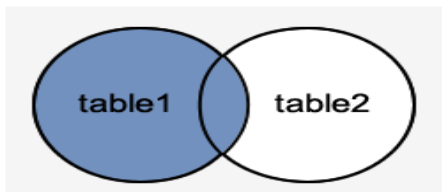

MySQL Left Outer Join

The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

Syntax:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

Consider two tables "officers" and "students", having the following data.

```
MySQL 5.5 Command Line Client
4 rows in set (0.00 sec)
mysql> SELECT * FROM officers;
+----+-----+-----+
| officer_id | officer_name | address |
+----+-----+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Uimal | Faizabad |
| 4 | Rahul | Lucknow |
+----+-----+-----+
4 rows in set (0.00 sec)
mysql> SELECT * FROM students;
+----+-----+-----+
| student_id | student_name | course_name |
+----+-----+-----+
| 1 | Aryan | Java |
| 2 | Rohini | Hadoop |
| 3 | Lallu | MongoDB |
+----+-----+-----+
3 rows in set (0.00 sec)
mysql>
```

Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name
FROM officers
LEFT JOIN students
ON officers.officer_id = students.student_id;
```

Output:

```
MySQL 5.5 Command Line Client
mysql> SELECT officers.officer_name, officers.address, students.course_name
-> FROM officers
-> LEFT JOIN students
-> ON officers.officer_id = students.student_id;
+----+-----+-----+
| officer_name | address | course_name |
+----+-----+-----+
| Ajeet | Mau | Java |
| Deepika | Lucknow | Hadoop |
| Uimal | Faizabad | MongoDB |
| Rahul | Lucknow | NULL |
+----+-----+-----+
4 rows in set (0.01 sec)
mysql>
```

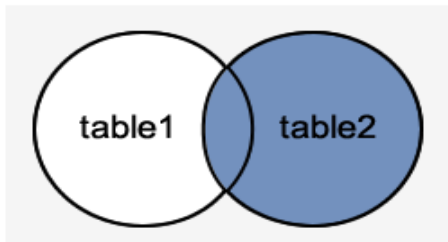
MySQL Right Outer Join

The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.

Syntax:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

Image representation:



Let's take an example:

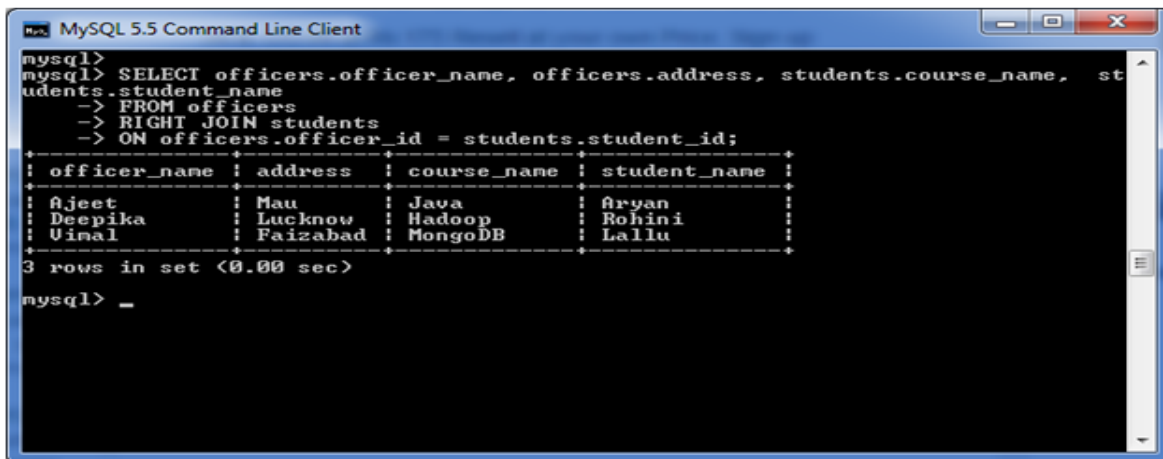
Consider two tables "officers" and "students", having the following data.

```
MySQL 5.5 Command Line Client
4 rows in set (0.00 sec)
mysql> SELECT * FROM officers;
+----+-----+-----+
| officer_id | officer_name | address |
+----+-----+-----+
| 1 | Ajeet | Mau |
| 2 | Deepika | Lucknow |
| 3 | Uimal | Faizabad |
| 4 | Rahul | Lucknow |
+----+-----+-----+
4 rows in set (0.00 sec)
mysql> SELECT * FROM students;
+----+-----+-----+
| student_id | student_name | course_name |
+----+-----+-----+
| 1 | Aryan | Java |
| 2 | Rohini | Hadoop |
| 3 | Lallu | MongoDB |
+----+-----+-----+
3 rows in set (0.00 sec)
mysql> _
```

Execute the following query:

```
SELECT officers.officer_name, officers.address, students.course_name, students.student_name
FROM officers
RIGHT JOIN students
ON officers.officer_id = students.student_id;
```

Output:



```
mysql> SELECT officers.officer_name, officers.address, students.course_name, st
udents.student_name
-> FROM officers
-> RIGHT JOIN students
-> ON officers.officer_id = students.student_id;
+-----+-----+-----+-----+
| officer_name | address | course_name | student_name |
+-----+-----+-----+-----+
| Ajeet       | Mau    | Java        | Aryan       |
| Deepika     | Lucknow | Hadoop      | Rohini      |
| Vimal       | Faizabad | MongoDB     | Lallu       |
+-----+-----+-----+-----+
3 rows in set <0.00 sec>

mysql> _
```

Aggregate Functions

[MySQL count\(\)](#) [MySQL sum\(\)](#) [MySQL avg\(\)](#) [MySQL min\(\)](#) [MySQL max\(\)](#) [MySQL first\(\)](#) [MySQL last\(\)](#)

String Functions:

S.No.	Function	Example
1	<p>CHAR_LENGTH(str)</p> <p>Returns the length of the string str, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, LENGTH() returns 10, whereas CHAR_LENGTH() returns 5.</p>	<pre>mysql> SELECT CHAR_LENGTH("text"); +-----+ CHAR_LENGTH("text") +-----+ 4 +-----+ 1 row in set (0.00 sec)</pre>
2	<p>CONCAT(str1,str2,...)</p> <p>Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example:</p>	<pre>mysql> SELECT CONCAT('My', 'S', 'QL'); +-----+ CONCAT('My', 'S', 'QL') +-----+ MySQL +-----+ 1 row in set (0.00 sec)</pre>
3	<p>BIT_LENGTH(str)</p> <p>Returns the length of the string str in bits.</p>	<pre>mysql> SELECT BIT_LENGTH('text'); +-----+ BIT_LENGTH('text') +-----+ 32 +-----+ 1 row in set (0.00 sec)</pre>

4	<p>CONCAT_WS(separator,str1,str2,...) CONCAT_WS() stands for Concatenate With Separator and is a special form of CONCAT(). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is NULL, the result is NULL.</p>	<pre>mysql> SELECT CONCAT_WS(',', 'First name', 'Last Name'); +-----+ CONCAT_WS(',', 'First name', 'Last Name') +-----+ First name, Last Name +-----+ 1 row in set (0.00 sec)</pre>
5	<p>FORMAT(X,D) Formats the number X to a format like '#,###,###.##', rounded to D decimal places, and returns the result as a string. If D is 0, the result has no decimal point or fractional part.</p>	<pre>mysql> SELECT FORMAT(12332.123456, 4); +-----+ FORMAT(12332.123456, 4) +-----+ 12,332.1235 +-----+ 1 row in set (0.00 sec)</pre>
6	<p>INSERT(str,pos,len,newstr) Returns the string str, with the substring beginning at position pos and len characters long replaced by the string newstr. Returns the original string if pos is not within the length of the string. Replaces the rest of the string from position pos if len is not within the length of the rest of the string. Returns NULL if any argument is NULL.</p>	<pre>mysql> SELECT INSERT('Quadratic', 3, 4, 'What'); +-----+ INSERT('Quadratic', 3, 4, 'What') +-----+ QuWhattic +-----+ 1 row in set (0.00 sec)</pre>
7	<p>INSTR(str,substr) Returns the position of the first occurrence of substring substr in string str. This is the same as the two-argument form of LOCATE(), except that the order of the arguments is reversed.</p>	<pre>mysql> SELECT INSTR('foobarbar', 'bar'); +-----+ INSTR('foobarbar', 'bar') +-----+ 4 +-----+ 1 row in set (0.00 sec)</pre>
8	<p>LOWER(str) Returns the string str with all characters changed to lowercase according to the current character set mapping.</p>	<pre>mysql> SELECT LOWER('QUADRATICALLY'); +-----+ LOWER('QUADRATICALLY') +-----+ quadratically +-----+ 1 row in set (0.00 sec)</pre>
9	<p>LTRIM(str) Returns the string str with leading space characters removed.</p>	<pre>mysql> SELECT LTRIM(' barbar'); +-----+ LTRIM(' barbar') +-----+ barbar +-----+ 1 row in set (0.00 sec)</pre>

Date and Time Functions

S.No.	Function	Example
1	<p>CURDATE()</p> <p>Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.</p>	<pre>mysql> SELECT CURDATE();</pre> <pre>+-----+ CURDATE() +-----+ 1997-12-15 +-----+ 1 row in set (0.00 sec)</pre> <pre>mysql> SELECT CURDATE() + 0;</pre> <pre>+-----+ CURDATE() + 0 +-----+ 19971215 +-----+ 1 row in set (0.00 sec)</pre>
2	<p>CURTIME()</p> <p>Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context. The value is expressed in the current time zone.</p>	<pre>mysql> SELECT CURTIME();</pre> <pre>+-----+ CURTIME() +-----+ 23:50:26 +-----+ 1 row in set (0.00 sec)</pre> <pre>mysql> SELECT CURTIME() + 0;</pre> <pre>+-----+ CURTIME() + 0 +-----+ 235026 +-----+ 1 row in set (0.00 sec)</pre>
3	<p>DATE(expr)</p> <p>Extracts the date part of the date or datetime expression expr.</p>	<pre>mysql> SELECT DATE('2003-12-31 01:02:03');</pre> <pre>+-----+ DATE('2003-12-31 01:02:03') +-----+ 2003-12-31 +-----+ 1 row in set (0.00 sec)</pre>
4	<p>DATEDIFF(expr1,expr2)</p> <p>DATEDIFF() returns expr1 . expr2 expressed as a value in days from one date to the other. expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.</p>	<pre>mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');</pre> <pre>+-----+ DATEDIFF('1997-12-31 23:59:59','1997- 12-30') +-----+ 1 +-----+ 1 row in set (0.00 sec)</pre>

METADATA

Obtaining Database Metadata

MySQL provides several ways to obtain database metadata—that is, information about databases and the objects in them:

- `SHOW` statements such as `SHOW DATABASES` or `SHOW TABLES`
- Tables in the `INFORMATION_SCHEMA` database
- Command-line programs such as `mysqlshow` or `mysqldump`

The following sections describe how to use each of these information sources to access metadata.

2.7.1. Obtaining Metadata with SHOW

MySQL provides a `SHOW` statement that displays many types of database metadata. `SHOW` is helpful for keeping track of the contents of your databases and reminding yourself about the structure of your tables. The following examples demonstrate a few uses for `SHOW` statements.

List the databases you can access:

```
SHOW DATABASES;
```

Display the `CREATE DATABASE` statement for a database:

```
SHOW CREATE DATABASE db_name;
```

List the tables in the default database or a given database:

```
SHOW TABLES;
```

```
SHOW TABLES FROM db_name;
```

`SHOW TABLES` doesn't show `TEMPORARY` tables.

Display the `CREATE TABLE` statement for a table:

```
SHOW CREATE TABLE tbl_name;
```

Display information about columns or indexes in a table:

```
SHOW COLUMNS FROM tbl_name;
```

```
SHOW INDEX FROM tbl_name;
```

The `DESCRIBE tbl_name` and `EXPLAIN tbl_name` statements are synonymous with `SHOW COLUMNS FROM tbl_name`.

Display descriptive information about tables in the default database or in a given database:

```
SHOW TABLE STATUS;
```

```
SHOW TABLE STATUS FROM db_name;
```

Several forms of the `SHOW` statement take a `LIKE 'pattern'` clause permitting a pattern to be given that limits the scope of the output. MySQL interprets `'pattern'` as an SQL pattern that may include the `'%'` and `'_'` wildcard characters. For example, this statement displays the names of columns in the `student` table that begin with `'s'`:

```
mysql> SHOW COLUMNS FROM student LIKE 's%';
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| sex   | enum('F','M') | NO  |     | NULL    |           |
| student_id | int(10) unsigned | NO  | PRI | NULL    | auto_increment |
+-----+-----+-----+-----+-----+-----+
```

To match a literal instance of a wildcard character in a `LIKE` pattern, precede it with a back-slash. This is commonly done to match a literal `'_'`, which occurs frequently in database, table, and column names.

Any `SHOW` statement that supports a `LIKE` clause can also be written to use a `WHERE` clause. The statement displays the same columns, but `WHERE` provides more flexibility about specifying which rows to return. The `WHERE` clause should refer to the `SHOW` statement column names. If the column name is a reserved word such as `KEY`, specify it as a quoted identifier. This statement determines which column in the `student` table is the primary key:

```
mysql> SHOW COLUMNS FROM student WHERE `Key` = `PRI`;
```

```
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| student_id | int(10) unsigned | NO  | PRI | NULL    | auto_increment |
+-----+-----+-----+-----+-----+-----+
```

MySQL: Sequences (AUTO_INCREMENT)

This MySQL tutorial explains how to **create sequences** using the `AUTO_INCREMENT` attribute in MySQL with syntax and examples.

Description

In MySQL, you can create a column that contains a sequence of numbers (1, 2, 3, and so on) by using the `AUTO_INCREMENT` attribute. The `AUTO_INCREMENT` attribute is used when you need to create a unique number to act as a primary key in a table.

Syntax

The syntax to create a sequence (or use the `AUTO_INCREMENT` attribute) in MySQL is:

```
CREATE TABLE table_name
(
  column1 datatype NOT NULL AUTO_INCREMENT,
  column2 datatype [ NULL | NOT NULL ],
  ...
);
```

AUTO_INCREMENT

The attribute to use when you want MySQL to assign a sequence of numbers automatically to a field (in essence, creating an autonumber field).

NULL or NOT NULL

Each column should be defined as `NULL` or `NOT NULL`. If this parameter is omitted, the database assumes `NULL` as the default.

Note

You can use the [LAST_INSERT_ID](#) function to find last value assigned by the AUTO_INCREMENT field.

Example

Let's look at an example of how to use a sequence or the AUTO_INCREMENT attribute in MySQL. For example:

```
CREATE TABLE contacts
( contact_id INT(11) NOT NULL AUTO_INCREMENT,
  last_name VARCHAR(30) NOT NULL,
  first_name VARCHAR(25),
  birthday DATE,
  CONSTRAINT contacts_pk PRIMARY KEY (contact_id)
);
```

This MySQL AUTO_INCREMENT example creates a table called *contacts* which has 4 columns and one primary key:

- The first column is called *contact_id* which is created as an INT datatype (maximum 11 digits in length) and can not contain NULL values. It is set as an AUTO_INCREMENT field which means that it is an autonumber field (starting at 1, and incrementing by 1, unless otherwise specified.)
- The second column is called *last_name* which is a VARCHAR datatype (maximum 30 characters in length) and can not contain NULL values.
- The third column is called *first_name* which is a VARCHAR datatype (maximum 25 characters in length) and can contain NULL values.
- The fourth column is called *birthday* which is a DATE datatype and can contain NULL values.
- The primary key is called *contacts_pk* and is set to the *contact_id* column.

Set AUTO_INCREMENT starting value

Now that you've created a table using the AUTO_INCREMENT attribute, how can you change the starting value for the AUTO_INCREMENT field if you don't want to start at 1?

You can use the ALTER TABLE statement to change or set the next value assigned by the AUTO_INCREMENT.

Syntax

In MySQL, the syntax to change the starting value for an AUTO_INCREMENT column using the ALTER TABLE statement is:

```
ALTER TABLE table_name AUTO_INCREMENT = start_value;
```

table_name

The name of the table whose AUTO_INCREMENT value you wish to change. Since a table in MySQL can only contain one AUTO_INCREMENT column, you are only required to specify the table name that contains the sequence. You do not need to specify the name of the column that contains the AUTO_INCREMENT value.

start_value

The next value in the sequence to assign in the AUTO_INCREMENT column.

Example

Let's look at an example of how to change the starting value for the AUTO_INCREMENT column in a table in MySQL.

For example:

```
ALTER TABLE contacts AUTO_INCREMENT = 50;
```

This MySQL AUTO_INCREMENT example would change the next value in the AUTO_INCREMENT field (ie: next value in the sequence) to 50 for the *contact_id* field in the *contacts* table.

PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985-1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). It has become famous because of its apparent and easily understandable syntax, portability and easy to learn.

Facts About Python

- Python is derived from programming languages such as ABC, Modula 3, small talk, Algol-68.
- Python page is a file with a `.py` extension that contains could be the combination of HTML Tags and Python scripts.
- In December 1989 the creator developed the 1st python interpreter as a `hobby` and then on 16 October 2000, Python 2.0 was released with many new features.
- On 3rd December 2008, `Python 3.0` was released with more testing and includes new features.
- Python is an open source scripting language.
- Python is free to download and use.
- Python is one of the official languages at Google.

Features of Python

- **Interpreted Language:** Python is processed at runtime by Python Interpreter.
- **Object-Oriented Language:** It supports object-oriented features and techniques of programming.
- **Interactive Programming Language:** Users can interact with the python interpreter directly for writing programs.
- **Easy language:** Python is easy to learn language especially for beginners.
- **Straightforward Syntax:** The formation of python syntax is simple and straightforward which also makes it popular.
- **Easy to read:** Python source-code is clearly defined and visible to the eyes.
- **Portable:** Python codes can be run on a wide variety of hardware platforms having the same interface.
- **Extendable:** Users can add low level-modules to Python interpreter.
- **Scalable:** Python provides an improved structure for supporting large programs then shell-scripts.

Python is used to create web and desktop applications, and some of the most popular web applications like `Instagram`, `YouTube`, `Spotify` all has been developed in Python, and you can also develop next big thing by using Python.

PYTHON Installation:

Python source code is available under the GNU General Public License (GPL).

Python interpreter is free, and downloads are available for all major platforms (Windows, Mac OS, and Linux) in the form of source and binary. You can download it from the Python Website: python.org.

Python GUI

There are various GUI based Python IDE that python programmers can use for better coding experience.

Names of some Python interpreters are:

- PyCharm
- Python IDLE
- The Python Bundle
- pyGUI
- Sublime Text etc.

Interpreter:

Interpreter can be termed as system software which has the capability to read and execute the program, rather you can say interpret programs. This interpretation includes the source code of high-level language, programs that are pre-compiled as well as scripts. It is to be noted that, interpreter interprets program line-by-line, which means it translates one statement at a single go. This feature makes easy for programmers to check any particular line at the time of debugging, but slows down the overall time of execution of the entire program.

Basics of PYTHON

Python is an interpreted programming language. Python source code is compiled to bytecode as a `.pyc` file, and this bytecode can be interpreted.

There are two modes for using the Python interpreter:

1. Interactive Mode
2. Script Mode

Interactive Mode

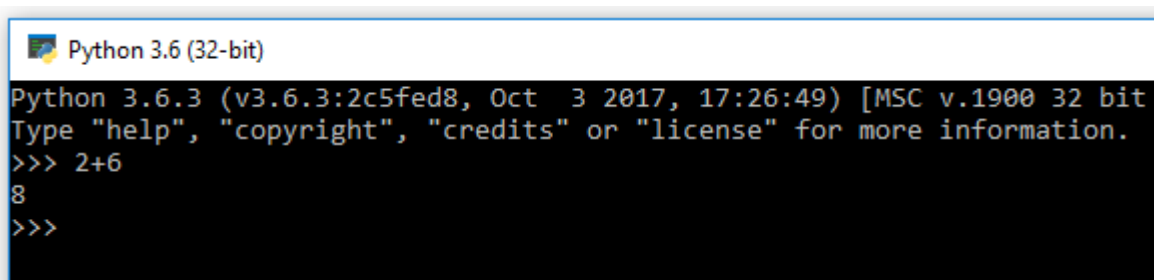
Without passing python script file to the interpreter, directly execute code to Python prompt.

Example:

```
>>>2+6
```

Output:

```
8
```



```
Python 3.6 (32-bit)
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+6
8
>>>
```

The chevron at the beginning of the 1st line, i.e., the symbol >>> is a prompt the python interpreter uses to indicate that it is ready. If the programmer types 2+6, the interpreter replies 8.

Script Mode

Alternatively, programmers can store Python script source code in a file with the .py extension, and use the interpreter to execute the contents of the file. To execute the script by the interpreter, you have to tell the interpreter the name of the file. For example, if you have a script name MyFile.py and you're working on Unix, to run the script you have to type:

```
python MyFile.py
```

Working with the interactive mode is better when Python programmers deal with small pieces of code as you can type and execute them immediately, but when the code is more than 2-4 lines, using the script for coding can help to modify and use the code in future.

Python - Variable Types

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example –

```
#!/usr/bin/python
counter = 100      # An integer assignment
miles   = 1000.0  # A floating point
name    = "John"  # A string
print counter
print miles
print name
```

Here, 100, 1000.0 and "John" are the values assigned to *counter*, *miles*, and *name* variables, respectively. This produces the following result –

```
100
1000.0
John
```

Multiple Assignment

Python allows you to assign a single value to several variables simultaneously. For example –

```
a = b = c = 1
```

Here, an integer object is created with the value 1, and all three variables are assigned to the same memory location. You can also assign multiple objects to multiple variables. For example –

```
a,b,c = 1,2,"john"
```

Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value "john" is assigned to the variable c.

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

Python has five standard data types –

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Number data types store numeric values. Number objects are created when you assign a value to them. For example –

```
var1 = 1  
var2 = 10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is –

```
del var1[,var2[,var3[...varN]]]
```

You can delete a single object or multiple objects by using the del statement. For example –

```
del var  
del var_a, var_b
```

Python supports four different numerical types –

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Examples

- Here are some examples of numbers –

int	long	float	complex
10	51924361L	0.0	3.14j
100	-0x19323L	15.20	45.j
-786	0122L	-21.9	9.322e-36j
080	0xDEFABCECBDAECBFBAEI	32.3+e18	.876j
-0490	535633629843L	-90.	-.6545+0j
-0x260	-052318172735L	-32.54e100	3e+26j
0x69	-4721885298529L	70.2-E12	4.53e-7j

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L.
- A complex number consists of an ordered pair of real floating-point numbers denoted by $x + yj$, where x and y are the real numbers and j is the imaginary unit.

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator (`[]` and `[:]`) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example –

```
#!/usr/bin/python
str = 'Hello World!'

print str      # Prints complete string
print str[0]   # Prints first character of the string
print str[2:5] # Prints characters starting from 3rd to 5th
```

```
print str[2:]    # Prints string starting from 3rd character
print str * 2    # Prints string two times
print str + "TEST" # Prints concatenated string
```

This will produce the following result –

```
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example –

```
#!/usr/bin/python
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list      # Prints complete list
print list[0]   # Prints first element of the list
print list[1:3] # Prints elements starting from 2nd till 3rd
print list[2:]  # Prints elements starting from 3rd element
print tinylist * 2 # Prints list two times
print list + tinylist # Prints concatenated lists
```

This produce the following result –

```
['abcd', 786, 2.23, 'john', 70.2000000000000003]
abcd
[786, 2.23]
```

```
[2.23, 'john', 70.200000000000003]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']
```

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as **read-only** lists. For example –

```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print tuple          # Prints complete list
print tuple[0]      # Prints first element of the list
print tuple[1:3]    # Prints elements starting from 2nd till 3rd
print tuple[2:]     # Prints elements starting from 3rd element
print tinytuple * 2 # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

This produce the following result –

```
('abcd', 786, 2.23, 'john', 70.200000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.200000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed. Similar case is possible with lists –


```
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000    # Invalid syntax with tuple
list[2] = 1000    # Valid syntax with list
```

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces (`{ }`) and values can be assigned and accessed using square braces (`[]`). For example –

```
#!/usr/bin/python

dict = {}
dict['one'] = "This is one"
dict[2]    = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}

print dict['one']    # Prints value for 'one' key
print dict[2]       # Prints value for 2 key
print tinydict      # Prints complete dictionary
print tinydict.keys() # Prints all the keys
print tinydict.values() # Prints all the values
```

This produce the following result –

```
This is one
This is two
{'dept': 'sales', 'code': 6734, 'name': 'john'}
['dept', 'code', 'name']
['sales', 6734, 'john']
```

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

Python - Basic Operators

Operators are the constructs which can manipulate the value of operands.

Consider the expression $4 + 5 = 9$. Here, 4 and 5 are called operands and + is called operator.

Types of Operator

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python - Decision Making

Decisions in a program are used when the program has conditional choices to execute code block. Let's take an example of traffic lights, where different colors of lights lit up at different situations based on the conditions of the road or any specific rule.

It is the prediction of conditions that occur while executing a program to specify actions. Multiple expressions get evaluated with an outcome of either TRUE or FALSE. These are logical decisions, and Python also provides decision-making statements that to make decisions within a program for an application based on the user requirement.

Python provides various types of conditional statements:

Statement	Description
if Statements	It consists of a Boolean expression which results is either TRUE or FALSE followed by one or more statements.
if else Statements	It also contains a Boolean expression. The if statement is followed by an optional else statement & if the expression results in FALSE, then else statement gets executed. It is also called alternative execution in which there are two possibilities of the condition determined in which any one of them will get executed.

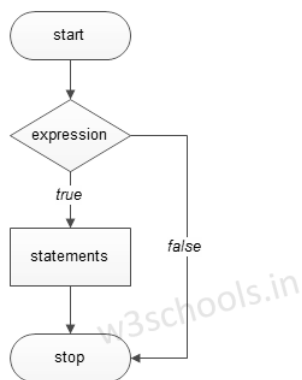
Nested Statements

We can implement if statement and or if-else statement inside another if or if - else statement. Here more than one if conditions are applied & there can be more than one if within elif.

if Statement

The decision-making structures can be recognized and understood using flowcharts.

Figure - If condition Flowchart:



Syntax:

if expression:

#execute your code

Example:

```
a = 15
```

```
if a > 10:
```

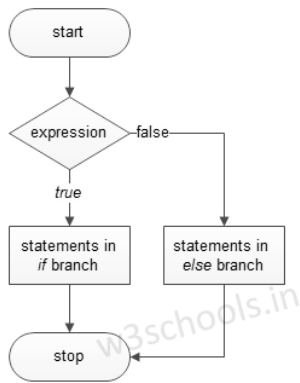
```
    print("a is greater")
```

Output:

```
a is greater
```

if else Statements

Figure - If else condition Flowchart:



Syntax:

```
if expression:
```

```
    #execute your code
```

```
else:
```

```
    #execute your code
```

Example:

```
a = 15
```

```
b = 20
```

```
if a > b:
```

```
    print("a is greater")
```

```
else:
```

```
    print("b is greater")
```

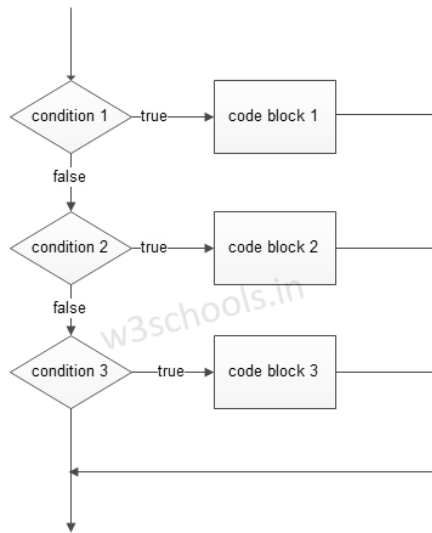
Output:

```
b is greater
```

elif Statements

elif - is a keyword used in Python replacement of else if to place another condition in the program. This is called chained conditional.

Figure - elif condition Flowchart:



Syntax:

if expression:

`#execute your code`

elif expression:

`#execute your code`

else:

`#execute your code`

Example:

```
a = 15
```

```
b = 15
```

```
if a > b:
```

```
    print("a is greater")
```

```
elif a == b:
```

```
    print("both are equal")
```

```
else:
```

```
    print("b is greater")
```

Output:

```
both are equal
```

Single Statement Condition

If the block of an executable statement of if - clause contains only a single line, programmers can write it on the same line as a header statement.

Example:

```
a = 15  
  
if (a == 15): print("The value of a is 15")
```

PYTHON LOOPS:

In programming, loops are a sequence of instructions that does a specific set of instructions or tasks based on some conditions and continue the tasks until it reaches certain conditions.

It is seen that in programming, sometimes we need to write a set of instructions repeatedly - which is a tedious task and the processing also takes time. So in programming, we use iteration technique to repeat the same or similar type of tasks based on the specified condition.

Statements are executed sequentially, but there sometimes occur such cases where programmers need to execute a block of code several times. The control structures of programming languages allow us to execute a statement or block of statement repeatedly.

Types of Loops in Python

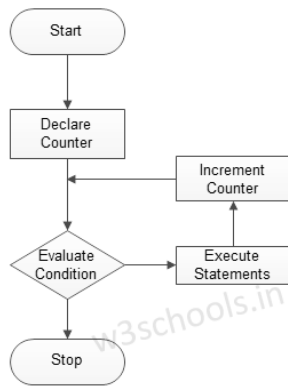
Python provides three types of looping techniques:

Loop	Description
for Loop	This is traditionally used when programmers had a piece of code and wanted to repeat that 'n' number of times.
while Loop	The loop gets repeated until the specific Boolean condition is met.
Nested Loops	Programmers can use one loop inside another; i.e., they can use for loop inside while or vice - versa or for loop inside for loop or while inside while.

for Loop

The graphical representation of the logic behind for looping is shown below:

Figure - for loop Flowchart:



Syntax:

for iterating_var in sequence:

#execute your code

Example 01:

```

for x in range (0,3) :
    print ('Loop execution %d' % (x))
  
```

Output:

Loop execution 0
 Loop execution 1
 Loop execution 2

Example 02:

```

for letter in 'TutorialsCloud':
    print ('Current letter is:', letter)
  
```

Output:

Current letter is : T
 Current letter is : u
 Current letter is : t
 Current letter is : o
 Current letter is : r
 Current letter is : i
 Current letter is : a
 Current letter is : l

Current letter is : s

Current letter is : C

Current letter is : l

Current letter is : o

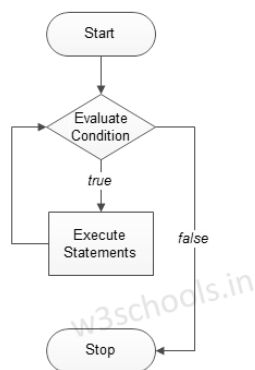
Current letter is : u

Current letter is : d

while Loop

The graphical representation of the logic behind while looping is shown below:

Figure - while loop Flowchart:



Syntax:

while expression:

```
#execute your code
```

Example:

```
#initialize count variable to 1
```

```
count = 1
```

```
while count < 6 :
```

```
    print (count)
```

```
    count+=1
```

```
#the above line means count = count + 1
```

Output:

1

2

3
4
5

Nested Loops

Syntax:

```
for iterating_var in sequence:
```

```
    for iterating_var in sequence:
```

```
        #execute your code
```

```
    #execute your code
```

Example:

```
for g in range(1, 6):
```

```
    for k in range(1, 3):
```

```
        print ("%d * %d = %d" % ( g, k, g*k))
```

Output:

```
1 * 1 = 1
```

```
1 * 2 = 2
```

```
2 * 1 = 2
```

```
2 * 2 = 4
```

```
3 * 1 = 3
```

```
3 * 2 = 6
```

```
4 * 1 = 4
```

```
4 * 2 = 8
```

```
5 * 1 = 5
```

```
5 * 2 = 10
```

Loop Control Statements

These statements are used to change execution from its normal sequence.

Python supports three types of loop control statements:

Python Loop Control Statements

Control Statements	Description
Break statement	It is used to exit a while loop or a for loop. It terminates the looping & transfers execution to the statement next to the loop.
Continue statement	It causes the looping to skip the rest part of its body & start re-testing its condition.
Pass statement	It is used in Python to when a statement is required syntactically, and the programmer does not want to execute any code block or command.

Break statement

Syntax:

```
break
```

Example:

```
count = 0
while count <= 100: print (count) count += 1 if count >= 3:
    break
```

Output:

```
0
1
2
```

Continue statement

Syntax:

```
continue
```

Example:

```
for x in range(10):
    #check whether x is even
    if x % 2 == 0:
        continue
    print (x)
```

Output:

```
1  
3  
5  
7  
9
```

Pass Statement

Syntax:

```
pass
```

Example:

```
for letter in 'TutorialsCloud':  
    if letter == 'C':  
        pass  
        print ('Pass block')  
    print ('Current letter is:', letter)
```

Output:

```
Current letter is : T  
Current letter is : u  
Current letter is : t  
Current letter is : o  
Current letter is : r  
Current letter is : i  
Current letter is : a  
Current letter is : l  
Current letter is : s  
Pass block  
Current letter is : C  
Current letter is : l  
Current letter is : o  
Current letter is : u  
Current letter is : d
```

PYTHON FUCNTIONS:

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like `print()`, etc. but you can also create your own functions. These functions are called *user-defined functions*.

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (()).

- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

Syntax

```
def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]
```

By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Example

The following function takes a string as input parameter and prints it on standard screen.

```
def printme( str ):
    "This prints a passed string into this function"
    print str
    return
```

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call `printme()` function –

```
#!/usr/bin/python

# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print str
    return;

# Now you can call printme function
printme("I'm first call to user defined function!")
printme("Again second call to the same function")
```

When the above code is executed, it produces the following result –

```
I'm first call to user defined function!
Again second call to the same function
```

Pass by reference vs value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example –

```
#!/usr/bin/python
# Function definition is here
def changeme( mylist ):
    "This changes a passed list into this function"
    mylist.append([1,2,3,4]);
    print "Values inside the function: ", mylist
    return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Here, we are maintaining reference of the passed object and appending values in the same object. So, this would produce the following result –

```
Values inside the function: [10, 20, 30, [1, 2, 3, 4]]
Values outside the function: [10, 20, 30, [1, 2, 3, 4]]
```

There is one more example where argument is being passed by reference and the reference is being overwritten inside the called function.

```
#!/usr/bin/python
# Function definition is here
def changeme( mylist ):
    "This changes a passed list into this function"
    mylist = [1,2,3,4]; # This would assign new reference in mylist
    print "Values inside the function: ", mylist
    return
# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

The parameter *mylist* is local to the function *changeme*. Changing *mylist* within the function does not affect *mylist*. The function accomplishes nothing and finally this would produce the following result –

```
Values inside the function: [1, 2, 3, 4]
Values outside the function: [10, 20, 30]
```

Function Arguments

You can call a function by using the following types of formal arguments –

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Required arguments

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

To call the function *printme()*, you definitely need to pass one argument, otherwise it gives a syntax error as follows –

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print str
    return;
# Now you can call printme function
printme()
```

When the above code is executed, it produces the following result –

```
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

Keyword arguments

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters. You can also make keyword calls to the *printme()* function in the following ways –

```
#!/usr/bin/python
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print str
    return;
# Now you can call printme function
printme( str = "My string")
```

When the above code is executed, it produces the following result –

```
My string
```

The following example gives more clear picture. Note that the order of parameters does not matter.

```
#!/usr/bin/python

# Function definition is here
def printinfo( name, age ):
    "This prints a passed info into this function"
    print "Name: ", name
    print "Age ", age
    return;
```

```
# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

When the above code is executed, it produces the following result –

```
Name: miki
Age 50
```

Default arguments

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed –

```
#!/usr/bin/python

# Function definition is here
def printinfo( name, age = 35 ):
    "This prints a passed info into this function"
    print "Name: ", name
    print "Age ", age
    return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```

When the above code is executed, it produces the following result –

```
Name: miki
Age 50
Name: miki
Age 35
```

Variable-length arguments

You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.

Syntax for a function with non-keyword variable arguments is this –

```
def functionname([formal_args,] *var_args_tuple ):
    "function_docstring"
    function_suite
    return [expression]
```

An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Following is a simple example –

```
#!/usr/bin/python
# Function definition is here
def printinfo( arg1, *vartuple ):
    "This prints a variable passed arguments"
    print "Output is: "
    print arg1
    for var in vartuple:
        print var
```

```
    return;

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

When the above code is executed, it produces the following result –

```
Output is:
10
Output is:
70
60
50
```

The *Anonymous* Functions

These functions are called anonymous because they are not declared in the standard manner by using the *def* keyword. You can use the *lambda* keyword to create small anonymous functions.

- Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.
- An anonymous function cannot be a direct call to print because lambda requires an expression
- Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.
- Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

Syntax

The syntax of *lambda* functions contains only a single statement, which is as follows –

```
lambda [arg1 [,arg2,.....argn]]:expression
```

Following is the example to show how *lambda* form of function works –

```
#!/usr/bin/python

# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;

# Now you can call sum as a function
print "Value of total : ", sum( 10, 20 )
print "Value of total : ", sum( 20, 20 )
```

When the above code is executed, it produces the following result –

```
Value of total : 30
Value of total : 40
```


The *return* Statement

The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as `return None`.

All the above examples are not returning any value. You can return a value from a function as follows –

```
#!/usr/bin/python

# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2
    print "Inside the function : ", total
    return total;

# Now you can call sum function
total = sum( 10, 20 );
print "Outside the function : ", total
```

When the above code is executed, it produces the following result –

```
Inside the function : 30
Outside the function : 30
```

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python –

- Global variables
- Local variables

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

Following is a simple example –

```
#!/usr/bin/python

total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
    # Add both the parameters and return them."
    total = arg1 + arg2; # Here total is local variable.
    print "Inside the function local total : ", total
    return total;
```

```
# Now you can call sum function
sum( 10, 20 );
print "Outside the function global total : ", total
```

When the above code is executed, it produces the following result –

```
Inside the function local total : 30
Outside the function global total : 0
```

PYTHON – MODULES:

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example

The Python code for a module named *aname* normally resides in a file named *aname.py*. Here's an example of a simple module, *support.py*

```
def print_func( par ):
    print "Hello : ", par
    return
```

The *import* Statement

You can use any Python source file as a module by executing an import statement in some other Python source file. The *import* has the following syntax –

```
import module1[, module2[,... moduleN]
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module *support.py*, you need to put the following command at the top of the script –

```
#!/usr/bin/python

# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Zara")
```

When the above code is executed, it produces the following result –

```
Hello : Zara
```

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

The *from...import* Statement

Python's *from* statement lets you import specific attributes from a module into the current namespace. The *from...import* has the following syntax –

```
from modname import name1[, name2[, ... nameN]]
```

For example, to import the function `fibonacci` from the module `fib`, use the following statement –

```
from fib import fibonacci
```

This statement does not import the entire module `fib` into the current namespace; it just introduces the item `fibonacci` from the module `fib` into the global symbol table of the importing module.

The *from...import ** Statement

It is also possible to import all names from a module into the current namespace by using the following import statement –

```
from modname import *
```

This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

PYTHON – FILES I/O

Printing to the Screen

The simplest way to produce output is using the *print* statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows –

```
#!/usr/bin/python

print "Python is really a great language,", "isn't it?"
```

This produces the following result on your standard screen –

```
Python is really a great language, isn't it?
```

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are –

- `raw_input`
- `input`
-

The `raw_input` Function

The `raw_input([prompt])` function reads one line from standard input and returns it as a string (removing the trailing newline).

```
#!/usr/bin/python

str = raw_input("Enter your input: ");
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this –

```
Enter your input: Hello Python
Received input is : Hello Python
```

The `input` Function

The `input([prompt])` function is equivalent to `raw_input`, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
#!/usr/bin/python
str = input("Enter your input: ");
print "Received input is : ", str
```

This would produce the following result against the entered input –

```
Enter your input: [x*5 for x in range(2,10,2)]
Received input is : [10, 20, 30, 40]
```

Opening and Closing Files

Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a **file** object.

The `open` Function

Before you can read or write a file, you have to open it using Python's built-in `open()` function. This function creates a **file** object, which would be utilized to call other support methods associated with it.

Syntax

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details –

- **file_name** – The file_name argument is a string value that contains the name of the file that you want to access.
- **access_mode** – The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).
- **buffering** – If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file –

1 r

Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

2 rb

Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

3 r+

Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

4 rb+

Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

5 w

Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

6 wb

Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

7 w+

Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

8 wb+

Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

9 a

Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

10 ab

Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

11 a+

Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

12 ab+

Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The *file* Object Attributes

Once a file is opened and you have one *file* object, you can get various information related to that file.

Here is a list of all attributes related to file object –

Sr.No.	Attribute & Description
1	file.closed Returns true if file is closed, false otherwise.
2	file.mode Returns access mode with which file was opened.
3	file.name Returns name of the file.
4	file.softspace Returns false if space explicitly required with print, true otherwise.

Example

```
#!/usr/bin/python
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed
print "Opening mode : ", fo.mode
print "Softspace flag : ", fo.softspace
```

This produces the following result –

```
Name of the file:  foo.txt
Closed or not :  False
Opening mode :  wb
Softspace flag :  0
```

The `close()` Method

The `close()` method of a *file* object flushes any unwritten information and closes the file object, after which no more writing can be done.

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the `close()` method to close a file.

Syntax

```
fileObject.close();
```

Example

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name

# Close opened file
fo.close()
```

This produces the following result –

```
Name of the file:  foo.txt
```

Reading and Writing Files

The *file* object provides a set of access methods to make our lives easier. We would see how to use `read()` and `write()` methods to read and write files.

The `write()` Method

The `write()` method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.

The `write()` method does not add a newline character (`'\n'`) to the end of the string –

Syntax

```
fileObject.write(string);
```

Here, passed parameter is the content to be written into the opened file.

Example

```
#!/usr/bin/python

# Open a file
fo = open("foo.txt", "wb")
fo.write( "Python is a great language.\nYeah its great!!\n");

# Close opened file
fo.close()
```

The above method would create *foo.txt* file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

```
Python is a great language.  
Yeah its great!!
```

The *read()* Method

The *read()* method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax

```
fileObject.read([count]);
```

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if *count* is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file *foo.txt*, which we created above.

```
#!/usr/bin/python  
  
# Open a file  
fo = open("foo.txt", "r+")  
str = fo.read(10);  
print "Read String is : ", str  
# Close opened file  
fo.close()
```

This produces the following result –

```
Read String is : Python is
```

File Positions

The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

Example

Let us take a file *foo.txt*, which we created above.

```
#!/usr/bin/python  
  
# Open a file  
fo = open("foo.txt", "r+")  
str = fo.read(10);  
print "Read String is : ", str
```



```
# Check current position
position = fo.tell();
print "Current file position : ", position

# Reposition pointer at the beginning once again
position = fo.seek(0, 0);
str = fo.read(10);
print "Again read String is : ", str
# Close opened file
fo.close()
```

This produces the following result –

```
Read String is : Python is
Current file position : 10
Again read String is : Python is
```

Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

The *rename()* Method

The *rename()* method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file *test1.txt* –

```
#!/usr/bin/python

import os

# Rename a file from test1.txt to test2.txt

os.rename( "test1.txt", "test2.txt" )
```

The *remove()* Method

You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

Following is the example to delete an existing file *test2.txt* –

```
#!/usr/bin/python

import os

# Delete file test2.txt
os.remove("text2.txt")
```

Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

The *mkdir()* Method

You can use the *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

Syntax

```
os.mkdir("newdir")
```

Example

Following is the example to create a directory *test* in the current directory –

```
#!/usr/bin/python
import os

# Create a directory "test"
os.mkdir("test")
```

The *chdir()* Method

You can use the *chdir()* method to change the current directory. The *chdir()* method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax

```
os.chdir("newdir")
```

Example

Following is the example to go into *"/home/newdir"* directory –

```
#!/usr/bin/python
import os

# Changing a directory to "/home/newdir"
os.chdir("/home/newdir")
```

The *getcwd()* Method

The *getcwd()* method displays the current working directory.

Syntax

```
os.getcwd()
```

Example

Following is the example to give current directory –

```
#!/usr/bin/python
import os

# This would give location of the current directory
os.getcwd()
```

The `rmdir()` Method

The `rmdir()` method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

Syntax

```
os.rmdir('dirname')
```

Example

Following is the example to remove `"/tmp/test"` directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

```
#!/usr/bin/python
import os
# This would remove "/tmp/test" directory.
os.rmdir( "/tmp/test" )
```

File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows –

- File Object Methods: The *file* object provides functions to manipulate files.
- OS Object Methods: This provides methods to process files as well as directories.

PYTHON – EXCEPTION HANDLING

Sometimes we want to catch some or all errors that can possibly get generated; and as a programmer, we want to be as specific as possible. So, python allows programmers to deal with errors smoothly.

Exceptions are events that are used to modify the flow of control through a program when the error occurs. Exceptions get triggered automatically on finding errors in Python.

These exceptions are processed using five statements. These are:

1. `try/except`: catch the error and recover from exceptions hoist by programmers or Python itself.
2. `try/finally`: Whether exception occurs or not, it automatically performs the clean-up action.
3. `assert`: triggers an exception conditionally in the code.
4. `raise`: manually triggers an exception in the code.
5. `with/as`: implement context managers in older versions of Python such as - Python 2.6 & Python 3.0.

The last was an optional extension to Python 2.6 & Python 3.0.

Why are Exceptions Used?

Exceptions allow us to jump out of random illogical large chunks of codes in case of errors. Let us take a scenario that you have given a function to do a specific task. If you go there and found those things missing that are required to do that specific task, what will you do? Either you stop working or think about a solution - where to find those items to perform the task. The same thing happens here in case of Exceptions also. Exceptions allow programmers jump to an exception handler in a single step, abandoning all function calls. You can think exceptions to an optimized quality go-to statement, in which the program error that occurs at runtime gets easily managed by the exception block. When the interpreter encounters an error, it lets the execution go to the exception part to solve and continue the execution instead of stopping.

While dealing with exceptions, the exception handler creates a mark & executes some code. Somewhere further within that program the exception is raised that solves the problem & makes Python jump back to the marked location; by not throwing away/skipping any active functions that were called after the marker was left.

Roles of an Exception Handler in Python

- **Error handling:** The exceptions get raised whenever Python detects an error in a program at runtime. As a programmer, if you don't want the default behavior then code a 'try' statement to catch and recover the program from an exception. Python will jump to the 'try' handler when the program detects an error the execution will be resumed.
- **Event Notification:** Exceptions are also used to signal suitable conditions & then passing result flags around a program & text them explicitly.
- **Terminate Execution:** There may arise some problems or errors in programs that it needs a termination. So try/finally is used that guarantees that closing-time operation will be performed. The 'with' statement offers an alternative for objects that support it.
- **Exotic flow of Control:** Programmers can also use exceptions as a basis for implementing unusual control flow. Since there is no 'go to' statement in Python so exceptions can help in this respect.

A Simple Program to Demonstrate Python Exception Handling

Example 01:

```
(a,b) = (6,0)
try:# simple use of try-except block for handling errors
    g = a/b
except ZeroDivisionError:
    print ("This is a DIVIDED BY ZERO error")
```

Output:

```
This is a DIVIDED BY ZERO error
```

The above program can also be written like this:

Example 02:

```
(a,b) = (6,0)
try:
    g = a/b
except ZeroDivisionError as s:
    k = s
    print (k)
#Output will be: integer division or modulo by zero
```

Output:

division by zero

The 'try - Except' Clause with No Exception

The structure of such type of 'except' clause having no exception is shown with an example.

```
try:
# all operations are done within this block.
.....
except:
# this block will get executed if any exception encounters.
.....
else:
# this block will get executed if no exception is found.
.....
```

All the exceptions get caught where there is try/except the statement of this type. Good programmers use this technique of exception to make the program fully executable.

'except' Clause with Multiple Exceptions

```
try:
# all operations are done within this block.
.....
except ( Exception1 [, Exception2[,....Exception N ] ] ) :
# this block will get executed if any exception encounters from the above lists of
exceptions.
.....
else:
# this block will get executed if no exception is found.
.....
```

The 'try - Finally' Clause

```
try:
# all operations are done within this block.
.....
# if any exception encounters, this block may get skipped.
finally:
.....
# this block will definitely be executed.
```

PHYTHON – OOP

The python is an Object-oriented programming Language. This means there exists a concept called 'class' that let's programmer structure the codes of software in a fashioned way. Because of the use of classes and objects, the programming became easy to understand and code.

Defining Class and Object

A class is a technique to group functions and data members and put them in a container so that they can be accessed later by using dot (.) operator. Objects are the basic runtime entities of object-oriented programming. It defines the instance of a class. Objects get their variables and functions from classes and the class we will be creating are the templates made to create the object.

Object-Oriented Terminologies

- **class:** Classes are a user-defined data type that is used to encapsulate data and associated functions together. It also helps in binding data together into a single unit.
- **Data Member:** A variable or memory location name that holds value to do a specific task within a program.
- **Member Function:** They are the functions; usually a block of a code snippet that is written to re-use it.
- **Instance variable:** A variable that is defined inside a method of a class.
- **Function Overloading:** This technique is used to assign multiple tasks to a single function & the tasks are performed based on the number of argument or the type of argument the function has.
- **Operator Overloading:** It is the technique of assigning multiple function/tasks to a particular operator.
- **Inheritance:** It is the process of acquiring the properties of one class to another, i.e., one class can acquire the properties of another.
- **Instantiation:** It is the technique of creating an instance of a class.

Program for Class in Python

Example:

```
class karl :  
    varabl = 'Hello'  
def function(self) :  
    print ("This is a message Hello")
```

Another program to explain functions inside a class:

Example:

```
class karl(object) :  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
def sample(self) :  
    print ("This is just a sample code")
```

In the above code, we created a class name karl using 'class' keyword. And two functions are used namely __init__() (for setting the instance variable) & sample(). Classes are used instead of modules because programmers can take this class 'karl' & use it or modify it as many times as we want & each one won't interfere with each other. Importing a module brings the entire program into use.

Creating Objects (Instance of A Class)

Let's see an example to show how to create an object:

Example:

```
class student:
    def __init__(self, roll, name):
        self.r = roll
        self.n = name
        print ((self.n))

#...
stud1 = student(1, "Alex")
stud2 = student(2, "Karlos")
print ("Data successfully stored in variables")
```

Output:

```
Alex
Karlos
Data successfully stored in variables
```

Accessing Object Variables

We can access the object's variable using dot (.) operator.

The syntax is:

```
my object_name.variable_name
```

Example:

```
print object.varabl
```

Accessing Attributes

Object attributes can also be accessed by using dot operator.

Example:

```
stud1.display()

stud2.display()

print ("total number of students are: %d" % student.i)
```

Use of Pre-defined Functions

Instead of using general statements to access attributes, programmers can use the following functions:

- `getattr(obj, name [,default])` : used to access object's attribute.
- `hasattr(object, name)`: used for checking whether the attribute exists or not.
- `setattr(obj, name, value)` : set or create an attribute, if doesn't exist.
- `delattr(obj, name)` : used to delete an attribute.

Built-in Class Attributes

All the Python built-in class attributes can be accessed using dot (.) operator like other attributes.

The built-in class attributes are:

- `__dict__`: This attribute is a dictionary that contains class's-namespace.
- `__doc__`: Used for class documentation string.
- `__name__`: used as class-name.
- `__module__`: used to define module name for the class in which it is defined. In interactive mode it is `__main__`.
- `__bases__`: An empty tuple containing the base-class.

PHP Tutorial

The PHP Hypertext Pre-processor (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases.

PHP is basically used for developing web based software applications.

PHP Introduction

PHP started out as a small open source project that evolved as more and more people found out how useful it was. Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

- PHP is a recursive acronym for "PHP: Hypertext Preprocessor".
- PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.
- It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.
- PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.
- PHP is forgiving: PHP language tries to be as forgiving as possible.
- PHP Syntax is C-Like.

Common uses of PHP:

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, thru email you can send data, return data to the user.
- You add, delete, modify elements within your database thru PHP.
- Access cookies variables and set cookies.
- Using PHP, you can restrict users to access some pages of your website.
- It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible:

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

"Hello World" Script in PHP:

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML (or XHTML if you're cutting-edge) you'll have PHP statements like this:

```
<html>
<head>
<title>Hello World</title>
<body>
  <?php echo "Hello, World!";?>
</body>
</html>
```

It will produce following result:

```
Hello, World!
```

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags ate are recognised by the PHP Parser.

```
<?php PHP code goes here ?>
<? PHP code goes here ?>
<script language="php"> PHP code goes here </script>
```

Most common tag is the <?php...?> and we will also use same tag in our tutorial.

From the next chapter we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

PHP Environment Setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server** - PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely available Apache Server. Download Apache for free here: <http://httpd.apache.org/download.cgi>
- **Database** - PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database. Download MySQL for free here: <http://www.mysql.com/downloads/index.html>
- **PHP Parser** - In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser. This tutorial will guide you how to install PHP parser on your computer.

PHP Parser Installation:

Before you proceed it is important to make sure that you have proper environment setup on your machine to develop your web programs using PHP.

Type the following address into your browser's address box.

```
http://127.0.0.1/info.php
```

If this displays a page showing your PHP installation related information then it means you have PHP and Webserver installed properly. Otherwise you have to follow given procedure to install PHP on your computer.

Apache Configuration:

If you are using Apache as a Web Server then this section will guide you to edit Apache Configuration Files.

PHP.INI File Configuration:

The PHP configuration file, php.ini, is the final and most immediate way to affect PHP's functionality.

Windows IIS Configuration:

To configure IIS on your Windows machine you can refer your IIS Reference Manual shipped along with IIS.

PHP Syntax Overview

This chapter will give you an idea of very basic syntax of PHP and very important to make your PHP foundation strong.

Escaping to PHP:

The PHP parsing engine needs a way to differentiate PHP code from other elements in the page. The mechanism for doing so is known as 'escaping to PHP.' There are four ways to do this:

Canonical PHP tags:

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

Short-open (SGML-style) tags:

Short or short-open tags look like this:

```
<?...?>
```

Short tags are, as one might expect, the shortest option. You must do one of two things to enable PHP to recognize the tags:

- Choose the --enable-short-tags configuration option when you're building PHP.
- Set the short_open_tag setting in your php.ini file to on. This option must be disabled to parse XML with PHP because the same syntax is used for XML tags.

ASP-style tags:

ASP-style tags mimic the tags used by Active Server Pages to delineate code blocks. ASP-style tags look like this:

```
<%...%>
```

To use ASP-style tags, you will need to set the configuration option in your php.ini file.

HTML script tags:

HTML script tags look like this:

```
<script language="PHP">...</script>
```

Commenting PHP Code:

A *comment* is the portion of a program that exists only for the human reader and stripped out before displaying the programs result. There are two commenting formats in PHP:

Single-line comments: They are generally used for short explanations or notes relevant to the local code. Here are the examples of single line comments.

```
<?
# This is a comment, and
# This is the second line of the comment
// This is a comment too. Each style comments only
print "An example with single line comments";
?>
```

Multi-lines printing: Here are the examples to print multiple lines in a single print statement:

```
<?
# First Example
print <<<END
This uses the "here document" syntax to output
multiple lines with $variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon no extra whitespace!
END;
# Second Example
print "This spans
multiple lines. The newlines will be
output as well";
?>
```

Multi-lines comments: They are generally used to provide pseudocode algorithms and more detailed explanations when necessary. The multiline style of commenting is the same as in C. Here are the example of multi lines comments.

```
<?
/* This is a comment with multiline
   Author : Mohammad Mohtashim
   Purpose: Multiline Comments Demo
   Subject: PHP
*/
print "An example with multi line comments";
?>
```

PHP is whitespace insensitive:

Whitespace is the stuff you type that is typically invisible on the screen, including spaces, tabs, and carriage returns (end-of-line characters).

PHP whitespace insensitive means that it almost never matters how many whitespace characters you have in a row. one whitespace character is the same as many such characters

For example, each of the following PHP statements that assigns the sum of 2 + 2 to the variable \$four is equivalent:

```
$four = 2 + 2; // single spaces
$four <tab>=<tab2<tab>+<tab>2 ; // spaces and tabs
$four =
2+
2; // multiple lines
```

PHP is case sensitive:

Yeah it is true that PHP is a case sensitive language. Try out following example:

```
<html>
<body>
<?
$capital = 67;
print("Variable capital is $capital<br>");
print("Variable CaPiTaL is $CaPiTaL<br>");
?>
</body>
</html>
```

This will produce following result:

```
Variable capital is 67
Variable CaPiTaL is
```

Statements are expressions terminated by semicolons:

A *statement* in PHP is any expression that is followed by a semicolon (;). Any sequence of valid PHP statements that is enclosed by the PHP tags is a valid PHP program. Here is a typical statement in PHP, which in this case assigns a string of characters to a variable called \$greeting:

```
$greeting = "Welcome to PHP!";
```

Expressions are combinations of tokens:

The smallest building blocks of PHP are the indivisible tokens, such as numbers (3.14159), strings (.two.), variables (\$two), constants (TRUE), and the special words that make up the syntax of PHP itself like if, else, while, for and so forth

Braces make blocks:

Although statements cannot be combined like expressions, you can always put a sequence of statements anywhere a statement can go by enclosing them in a set of curly braces.

Here both statements are equivalent:

```
if (3 == 2 + 1)
    print("Good - I haven't totally lost my mind.<br>");

if (3 == 2 + 1)
{
    print("Good - I haven't totally");
    print("lost my mind.<br>");
}
```

Running PHP Script from Command Prompt:

Yes you can run your PHP script on your command prompt. Assuming you have following content in test.php file

```
<?php
    echo "Hello PHP!!!!!";
?>
```

Now run this script as command prompt as follows:

```
$ php test.php
```

It will produce following result:

```
Hello PHP!!!!!
```

Hope now you have basic knowledge of PHP Syntax.

PHP Variable Types

The main way to store information in the middle of a PHP program is by using a variable.

Here are the most important things to know about variables in PHP.

- All variables in PHP are denoted with a leading dollar sign (\$).
- The value of a variable is the value of its most recent assignment.
- Variables are assigned with the = operator, with the variable on the left-hand side and the expression to be evaluated on the right.
- Variables can, but do not need, to be declared before assignment.
- Variables in PHP do not have intrinsic types - a variable does not know in advance whether it will be used to store a number or a string of characters.
- Variables used before they are assigned have default values.
- PHP does a good job of automatically converting types from one to another when necessary.
- PHP variables are Perl-like.

PHP has a total of eight data types which we use to construct our variables:

- **Integers:** are whole numbers, without a decimal point, like 4195.
- **Doubles:** are floating-point numbers, like 3.14159 or 49.1.
- **Booleans:** have only two possible values either true or false.
- **NULL:** is a special type that only has one value: NULL.
- **Strings:** are sequences of characters, like 'PHP supports string operations.'
- **Arrays:** are named and indexed collections of other values.
- **Objects:** are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources:** are special variables that hold references to resources external to PHP (such as database connections).

The first five are *simple types*, and the next two (arrays and objects) are compound - the compound types can package up other arbitrary values of arbitrary type, whereas the simple types cannot.

We will explain only simple data type in this chapters. Array and Objects will be explained separately.

Integers:

They are whole numbers, without a decimal point, like 4195. They are the simplest type .they correspond to simple whole numbers, both positive and negative. Integers can be assigned to variables, or they can be used in expressions, like so:

```
$int_var = 12345;
$another_int = -12345 + 12345;
```

Integer can be in decimal (base 10), octal (base 8), and hexadecimal (base 16) format. Decimal format is the default, octal integers are specified with a leading 0, and hexadecimals have a leading 0x.

For most common platforms, the largest integer is $(2^{31} - 1)$ (or 2,147,483,647), and the smallest (most negative) integer is $-(2^{31} - 1)$ (or -2,147,483,647).

Doubles:

They like 3.14159 or 49.1. By default, doubles print with the minimum number of decimal places needed. For example, the code:

```
$many = 2.2888800;
$many_2 = 2.2111200;
$few = $many + $many_2;
print(.$many + $many_2 = $few<br>.);
```

It produces the following browser output:

```
2.28888 + 2.21112 = 4.5
```

Boolean:

They have only two possible values either true or false. PHP provides a couple of constants especially for use as Booleans: TRUE and FALSE, which can be used like so:

```
if (TRUE)
    print("This will always print<br>");
else
```

```
    print("This will never print<br>");
```

Interpreting other types as Booleans:

Here are the rules for determine the "truth" of any value not already of the Boolean type:

- If the value is a number, it is false if exactly equal to zero and true otherwise.
- If the value is a string, it is false if the string is empty (has zero characters) or is the string "0", and is true otherwise.
- Values of type NULL are always false.
- If the value is an array, it is false if it contains no other values, and it is true otherwise. For an object, containing a value means having a member variable that has been assigned a value.
- Valid resources are true (although some functions that return resources when they are successful will return FALSE when unsuccessful).
- Don't use double as Booleans.

Each of the following variables has the truth value embedded in its name when it is used in a Boolean context.

```
$true_num = 3 + 0.14159;
>true_str = "Tried and true"
>true_array[49] = "An array element";
>false_array = array();
>false_null = NULL;
>false_num = 999 - 999;
>false_str = "";
```

NULL:

NULL is a special type that only has one value: NULL. To give a variable the NULL value, simply assign it like this:

```
$my_var = NULL;
```

The special constant NULL is capitalized by convention, but actually it is case insensitive; you could just as well have typed:

```
$my_var = null;
```

A variable that has been assigned NULL has the following properties:

- It evaluates to FALSE in a Boolean context.
- It returns FALSE when tested with IsSet() function.

Strings:

They are sequences of characters, like "PHP supports string operations". Following are valid examples of string

```
$string_1 = "This is a string in double quotes";
$string_2 = "This is a somewhat longer, singly quoted string";
$string_39 = "This string has thirty-nine characters";
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?
$variable = "name";
$literally = 'My $variable will not print!\n';
print($literally);
$literally = "My $variable will print!\n";
print($literally);
?>
```

This will produce following result:

```
My $variable will not print!\n
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with \$) are replaced with string representations of their values.

The escape-sequence replacements are:

- `\n` is replaced by the newline character
- `\r` is replaced by the carriage-return character
- `\t` is replaced by the tab character
- `\$` is replaced by the dollar sign itself (`$`)
- `\"` is replaced by a single double-quote (`"`)
- `\\` is replaced by a single backslash (`\`)

Here Document:

You can assign multiple lines to a single string variable using here document:

```
<?php

$channel =<<<_XML_
<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
</channel>
_XML_ ;

echo <<<END
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!
<br />

END;

print $channel;
?>
```

This will produce following result:

```
This uses the "here document" syntax to output
multiple lines with variable interpolation. Note
that the here document terminator must appear on a
line with just a semicolon. no extra whitespace!

<channel>
<title>What's For Dinner<title>
<link>http://menu.example.com/<link>
<description>Choose what to eat tonight.</description>
```

Variable Scope:

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

- [Local variables](#)
- [Function parameters](#)
- [Global variables](#)
- [Static variables](#)

Variable Naming:

Rules for naming a variable is:

- Variable names must begin with a letter or underscore character.
- A variable name can consist of numbers, letters, underscores but you cannot use characters like + , - , % , (,) . & , etc

There is no size limit for variables.

PHP Constants

A constant is a name or an identifier for a simple value. A constant value cannot change during the execution of the script. By default a constant is case-sensitive. By convention, constant identifiers are always uppercase. A constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. If you have defined a constant, it can never be changed or undefined.

To define a constant you have to use `define()` function and to retrieve the value of a constant, you have to simply specifying its name. Unlike with variables, you do not need to have a constant with a `$`. You can also use the function `constant()` to read a constant's value if you wish to obtain the constant's name dynamically.

constant() function:

As indicated by the name, this function will return the value of the constant.

This is useful when you want to retrieve value of a constant, but you do not know its name, i.e. It is stored in a variable or returned by a function.

constant() example:

```
<?php
define("MINSIZE", 50);

echo MINSIZE;
echo constant("MINSIZE"); // same thing as the previous line

?>
```

Only scalar data (boolean, integer, float and string) can be contained in constants.

Differences between constants and variables are:

- There is no need to write a dollar sign (\$) before a constant, where as in Variable one has to write a dollar sign.
- Constants cannot be defined by simple assignment, they may only be defined using the `define()` function.
- Constants may be defined and accessed anywhere without regard to variable scoping rules.
- Once the Constants have been set, may not be redefined or undefined.

Valid and invalid constant names:

```
// Valid constant names
define("ONE", "first thing");
define("TWO2", "second thing");
define("THREE_3", "third thing")
// Invalid constant names
define("2TWO", "second thing");
define(" THREE ", "third value");
```

PHP Magic constants:

PHP provides a large number of predefined constants to any script which it runs.

There are five magical constants that change depending on where they are used. For example, the value of `LINE` depends on the line that it's used on in your script. These special constants are case-insensitive and are as follows:

A few "magical" PHP constants are given below:

Name	Description
<code>__LINE__</code>	The current line number of the file.
<code>__FILE__</code>	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, <code>__FILE__</code> always contains an absolute path whereas in older versions it contained relative path under some circumstances.
<code>__FUNCTION__</code>	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
<code>__CLASS__</code>	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
<code>__METHOD__</code>	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

PHP Operator Types

What is Operator? Simple answer can be given using expression $4 + 5$ is equal to 9. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Arithmetic Operators:

There are following arithmetic operators supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Comparison Operators:

There are following comparison operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators:

There are following logical operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A or B) is true.
&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Assignment Operators:

There are following assignment operators supported by PHP language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ will assign value of $A + B$ into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	$C *= A$ is equivalent to $C = C * A$
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	$C /= A$ is equivalent to $C = C / A$
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	$C \% = A$ is equivalent to $C = C \% A$

Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax:

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Categories:

All the operators we have discussed above can be categorised into following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

Precedence of PHP Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example $x = 7 + 3 * 2$; Here x is assigned 13, not 20 because operator $*$ has higher precedence than $+$ so it first get multiplied with $3*2$ and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR		Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /=	

PHP Decision Making

The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following thredecision making statements:

- **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** - is used with the if...else statement to execute a set of code if **one** of several condition are true
- **switch statement** - is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

```
if (condition)
    code to be executed if condition is true;
else
    code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
    echo "Have a nice weekend!";
else
    echo "Have a nice day!";
?>
</body>
</html>
```

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

```
<html>

<body>
<?php
$d=date("D");
if ($d=="Fri")
{
    echo "Hello!<br />";
    echo "Have a nice weekend!";
    echo "See you on Monday!";
}
?>
</body>
</html>
```

The Elself Statement

If you want to execute some code if one of several conditions are true use the elseif statement

Syntax

```
if (condition)
    code to be executed if condition is true;
elseif (condition)
    code to be executed if condition is true;
else
    code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
    echo "Have a nice weekend!";
elseif ($d=="Sun")
    echo "Have a nice Sunday!";
else
    echo "Have a nice day!";
?>
</body>
</html>
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

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;
}
```

Example

The *switch* statement works in an unusual way. First it evaluates given expression then seeks a label to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the labels match then statement will execute any specified default code.

```

<html>
<body>
<?php
$d=date("D");
switch ($d)
{
case "Mon":
    echo "Today is Monday";
    break;
case "Tue":
    echo "Today is Tuesday";
    break;
case "Wed":
    echo "Today is Wednesday";
    break;
case "Thu":
    echo "Today is Thursday";
    break;
case "Fri":
    echo "Today is Friday";
    break;
case "Sat":
    echo "Today is Saturday";
    break;
case "Sun":
    echo "Today is Sunday";
    break;
default:
    echo "Wonder which day is this ?";
}
?>
</body>
</html>

```

PHP Loop Types

Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

- **for** - loops through a block of code a specified number of times.
- **while** - loops through a block of code if and as long as a specified condition is true.
- **do...while** - loops through a block of code once, and then repeats the loop as long as a special condition is true.
- **foreach** - loops through a block of code for each element in an array.

We will discuss about **continue** and **break** keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.

Syntax

```

for (initialization; condition; increment)
{
    code to be executed;
}

```

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it \$i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop:

```
<html>
<body>
<?php
$a = 0;
$b = 0;

for( $i=0; $i<5; $i++ )
{
    $a += 10;
    $b += 5;
}
echo ("At the end of the loop a=$a and b=$b" );
?>
</body>
</html>
```

This will produce following result:

```
At the end of the loop a=50 and b=25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

Syntax

```
while (condition)
{
    code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
<body>
<?php
$i = 0;
$num = 50;

while( $i < 10)
{
    $num--;
    $i++;
}
echo ("Loop stopped at i = $i and num = $num" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 1 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

```
do
{
    code to be executed;
}while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10:

```
<html>
<body>
<?php
$i = 0;
$num = 0;
do
{
    $i++;
}while( $i < 10 );
echo ("Loop stopped at i = $i" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to \$value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
foreach (array as value)
{
    code to be executed;
}
```

Example

Try out following example to list out the values of an array.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
    echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.

The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.

Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
<body>

<?php
$i = 0;

while( $i < 10)
{
    $i++;
    if( $i == 3 ) break;
}
echo ("Loop stopped at i = $i" );
?>
</body>
</html>
```

This will produce following result:

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.

Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
<body>
<?php
$array = array( 1, 2, 3, 4, 5);
foreach( $array as $value )
{
    if( $value == 3 )continue;
    echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP Arrays

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

- **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion
- **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

Numeric Array

These arrays can store numbers, strings and any object but their index will be prepresented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value )
{
    echo "Value is $value <br />";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";

$numbers[2] = "three";
$numbers[3] = "four";
$numbers[4] = "five";

foreach( $numbers as $value )
{
    echo "Value is $value <br />";
}
?>
</body>
</html>
```

This will produce following result:

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE: Don't keep associative array inside double quote while printing otherwise it would not return any value.

Example

```

<html>
<body>
<?php
/* First method to associate create array. */
$salaries = array(
    "mohammad" => 2000,
    "qadir" => 1000,
    "zara" => 500
);

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ". $salaries['qadir'] . "<br />";
echo "Salary of zara is ". $salaries['zara'] . "<br />";

/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ". $salaries['qadir'] . "<br />";

echo "Salary of zara is ". $salaries['zara'] . "<br />";
?>
</body>
</html>

```

This will produce following result:

```

Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low

```

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.


```

<html>
<body>
<?php
    $marks = array(
        "mohammad" => array
            (
                "physics" => 35,
                "maths" => 30,
                "chemistry" => 39
            ),
        "qadir" => array
            (
                "physics" => 30,
                "maths" => 32,
                "chemistry" => 29
            ),
        "zara" => array
            (
                "physics" => 31,
                "maths" => 22,
                "chemistry" => 39
            )
    );
    /* Accessing multi-dimensional array values */
    echo "Marks for mohammad in physics : " ;
    echo $marks['mohammad']['physics'] . "<br />";
    echo "Marks for qadir in maths : ";
    echo $marks['qadir']['maths'] . "<br />";
    echo "Marks for zara in chemistry : " ;
    echo $marks['zara']['chemistry'] . "<br />";
?>
</body>
</html>

```

This will produce following result:

```

Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39

```

PHP Strings

They are sequences of characters, like "PHP supports string operations".

Following are valid examples of string

```

$string_1 = "This is a string in double quotes";
$string_2 = "This is a somewhat longer, singly quoted string";
$string_39 = "This string has thirty-nine characters";
$string_0 = ""; // a string with zero characters

```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```

<?
$variable = "name";
$literally = 'My $variable will not print!\\n!';
print($literally);
$literally = "My $variable will print!\\n";
print($literally);
?>

```

This will produce following result:

```
My $variable will not print!\n
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with \$) are replaced with string representations of their values.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character
- \\$ is replaced by the dollar sign itself (\$)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator:

```
<?php
$string1="Hello World";
$string2="1234";
echo $string1 . " " . $string2;
?>
```

This will produce following result:

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

```
<?php
echo strlen("Hello world!");
?>
```

This will produce following result:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
echo strpos("Hello world!","world");
?>
```

This will produce following result:

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

PHP File Inclusion

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required then instead of changing thousand of files just change included file.

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -
<a href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have following content.

```
<html>
<body>
<?php include("menu.php"); ?>
<p>This is an example to show how to include PHP file!</p>
</body>
</html>
```

This will produce following result

[Home](#) - [ebXML](#) - [AJAX](#) - [PERL](#)

This is an example to show how to include PHP file. You can include mean.php file in as many as files you like!

The require() Function

The require() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file then the **require()** function generates a fatal error and halt the execution of the script.

So there is no difference in require() and include() except they handle error conditions. It is recommended to use the require() function instead of include(), because scripts should not continue executing if files are missing or misnamed.

You can try using above example with require() function and it will generate same result. But if you will try following two examples where file does not exist then you will get different results.

```
<html>
<body>
<?php include("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This will produce following result

This is an example to show how to include wrong PHP file!

Now lets try same example with require() function.

```
<html>
<body>
<?php require("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This time file execution halts and nothing is displayed.

NOTE: You may get plain warning messages or fatal error messages or nothing at all. This depends on your PHP Server configuration.

PHP Files & I/O

This chapter will explain following functions related to files:

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If files does not exist then it attempts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If files does not exist then it attempts to create a file.

If an attempt to open a file fails then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The file's length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
<html>

<head>
<title>Reading a file using PHP</title>
</head>
<body>

<?php
$filename = "/home/user/guest/tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
    echo ( "Error in opening file" );
    exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );

fclose( $file );

echo ( "File size : $filesize bytes" );
echo ( "<pre>$text</pre>" );
?>

</body>
</html>
```

Writing a file

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing will stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exists()** function which takes file name as an argument

```

<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
    echo ( "Error in opening new file" );
    exit();
}
fwrite( $file, "This is a simple test\n" );
fclose( $file );
?>

<html>
<head>
<title>Writing a file using PHP</title>
</head>
<body>

<?php
if( file_exist( $filename ) )
{
    $filesize = filesize( $filename );
    $msg = "File created with name $filename ";
    $msg .= "containing $filesize bytes";
    echo ( $msg );
}

else
{
    echo ( "File $filename does not exist" );
}
?>
</body>
</html>

```

PHP Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you:

- Creating a PHP Function
- Calling a PHP Function

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Creating PHP Function:

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage() and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

```

<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>

<?php
/* Defining a PHP Function */
function writeMessage()
{
    echo "You are really a nice person, Have a nice time!";
}
/* Calling a PHP Function */
writeMessage();
?>
</body>
</html>

```

This will display following result:

```
You are really a nice person, Have a nice time!
```

PHP Functions with Parameters:

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

```

<html>
<head>
<title>Writing PHP Function with Parameters</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
    $sum = $num1 + $num2;
    echo "Sum of the two numbers is : $sum";
}
addFunction(10, 20);
?>
</body>
</html>

```

This will display following result:

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference:

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.


```

<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function addFive($num)
{
    $num += 5;
}

function addSix(&$num)
{
    $num += 6;
}
$orignum = 10;
addFive( &$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );

echo "Original Value is $orignum<br />";
?>
</body>
</html>

```

This will display following result:

```

Original Value is 15
Original Value is 21

```

PHP Functions returning value:

A function can return a value using the **return** statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```

<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function addFunction($num1, $num2)
{
    $sum = $num1 + $num2;
    return $sum;
}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value
?>
</body>
</html>

```

This will display following result:

```

Returned value from the function : 30

```

Setting Default Values for Function Parameters:

You can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case use does not pass any value to this function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function printMe($param = NULL)
{

    print $param;
}
printMe("This is test");
printMe();
?>

</body>
</html>
```

This will produce following result:

```
This is test
```

Dynamic Function Calls:

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

```
<html>
<head>
<title>Dynamic Function Calls</title>
</head>
<body>
<?php
function sayHello()
{
    echo "Hello<br />";
}
$function_holder = "sayHello";
$function_holder();
?>
</body>
</html>
```

This will display following result:

```
Hello
```

Chapter 2. Setting up a Web Server

Table of Contents

2.1	Wampserver and Apache HTTP on Windows.....	1
2.1.1	Requirements.....	1
2.1.2	Installing Wampserver.....	2
2.1.3	Setting up Server Passwords	3
2.1.4	Testing Applications.....	5
2.2	Apache Tomcat on Windows	6
2.2.1	Requirements.....	6
2.2.2	Tomcat Setup.....	7
2.2.3	Testing Applications.....	10
2.3	Lampserver and Apache HTTP on Ubuntu 15.04	10
2.3.1	Requirements.....	10
2.3.2	Installing Apache, PHP and MySQL.....	11
2.3.3	Testing Applications.....	13
2.4	Apache Tomcat on Ubuntu 15.04.....	13
2.4.1	Testing Applications.....	14

Objectives

At the end of this unit you will be able to:

- install and setup Wamp server and Apache server on Windows;
- install and setup Tomcat Server on Windows;
- install and setup Lamp server and Apache server on Ubuntu 15.04; and
- install and setup Tomcat on Ubuntu 15.04.

2.1 Wampserver and Apache HTTP on Windows

In this section, you will learn how to set up a Web Server on a Windows PC. The steps in this section will illustrate how to use Apache HTTP. The next section will illustrate the setup for Apache Tomcat. Apache is a popular Web Server that allows users to easily set up their own Web Servers. It has the advantage of being open-source and hence is free to download. Apache is the basic software needed to support running of HTML and related content. Additional software, such as Tomcat, can be installed to complement the Web Server. Tomcat is a server that is meant to run applications written in Java and JSP (Java Server Pages).

Some popular options for deploying Apache, and optionally PHP and MySQL on Windows are Apache Lounge, XAMPP and Wampserver. Wampserver was used for this example. WAMP is an acronym that stands for “Windows, Apache, MySQL, and PHP”.

2.1.1 Requirements

To illustrate the steps below a Windows 7 64-bit computer was used. The Windows computer was connected to a local area network (LAN) that has Internet access. You also need to know the IP address of your

Web Servers

computer. You can find your IP address by typing 'ipconfig' at a command prompt. Find the entry labeled 'Ethernet adapter Local Area Network' and take note of the IPv4 address.

It is recommended to disable the Windows Firewall before starting the Web Server setup. The steps below are for a fresh installation of Wampserver (assumes that Wampserver had not been installed before).

2.1.2 Installing Wampserver

Download WAMP from <http://www.wampserver.com/en/> . You will have the option of choosing 64-bit or 32-bit installation depending on your PC. This example uses the 64-bit installation. Locate the downloaded Wampserver file and click on it. This will open an installation wizard as shown in Figure 1. Follow the instruction wizard and leave the default settings as they are. After successful installation you will get the window shown in Figure 2. Leave the 'launch WampServer 2 now' box checked and click on 'Finish' button (in future you can start WampServer by clicking on your Start menu and clicking on its menu). On your toolbar, you should now see a 'W' shaped icon. On left-clicking this icon, you get the pop-up management console in Figure 3. Click on 'Start all services' and then check the 'W' icon on your toolbar. If the 'W' icon is green it means that all services are running. If it is red it means that no services are running. If it is orange it means that some services are running. If everything was installed correctly you should see a window such as the one in Figure 4.

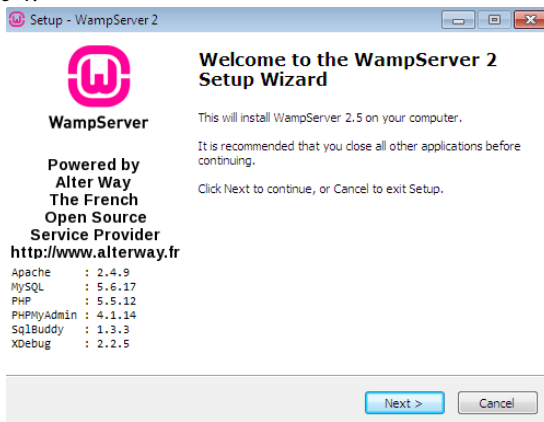


Figure 1: Welcome window to WAMP setup

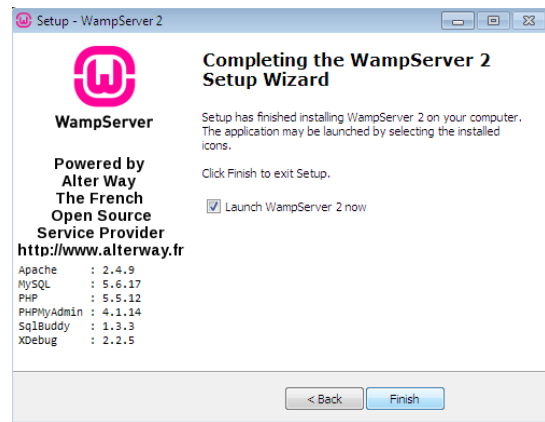


Figure 2: Finished installation

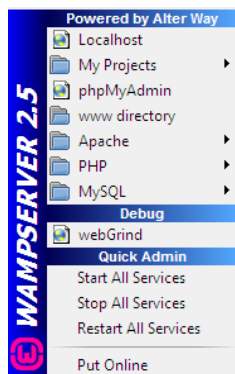


Figure 3: WampServer Management Console

Web Servers

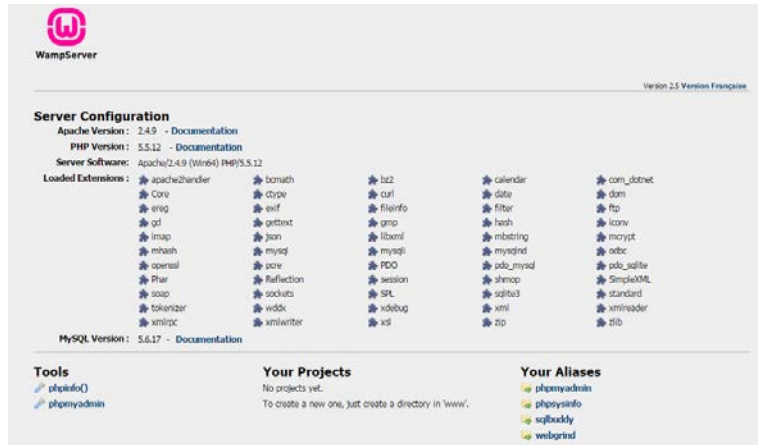


Figure 4: Localhost home screen

2.1.3 Setting up Server Passwords

In this step, you will learn how to create passwords for your server and also passwords to protect the files that you may want to share from your Web Server.

i. Setting up MySQL and PHP Admin Password

On your local host home screen (Figure 4) click on 'phpmyadmin' under 'Tools'. The interface window opens showing the WAMP configuration page. At the bottom of this page, a message indicates that MySQL is running without a password (Figure 5).

Click on users and then check the box next to 'root localhost' and then click on 'Edit Privileges' (Figure 6). Scroll down to change the password and then click on 'Go' to save the changes. If you try to click on any other menu on the interface, for example on the SQL menu, you will get an error. Let us fix this by also changing the PHP admin password to match the MySQL password.

Open Windows Explorer. Navigate to the C:\wamp\apps\phpmyadminx.x.x\ folder (Figure 7). Inside that folder open **config.inc.php** – ideally using Notepad or any other html editor.

Search for the line `$cfg['blowfish_secret'] = ''`; – if you're using notepad it might be easier to just search for the word 'blowfish'. Change the line `$cfg['blowfish_secret'] = 'abcdef'`; to `$cfg['blowfish_secret'] = 'my passphrase'`; where **my passphrase** is your own password – **not the same one that you specified for root in MySQL**.

Now search for the phrase `['auth_type'] = 'config'`. Change 'config' to 'cookie'. Now search for `$cfg['Servers'][$i]['password'] = ''`; Replace the " with 'mysql-password'; where mysql-password is the MySQL password you created earlier.

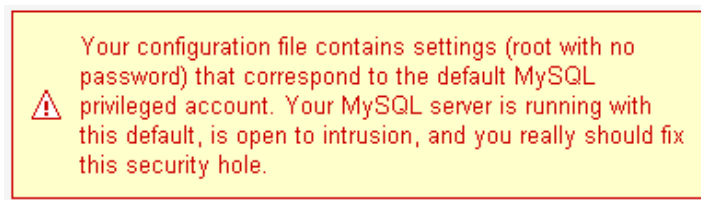


Figure 5: No Password set for WampServer

Web Servers

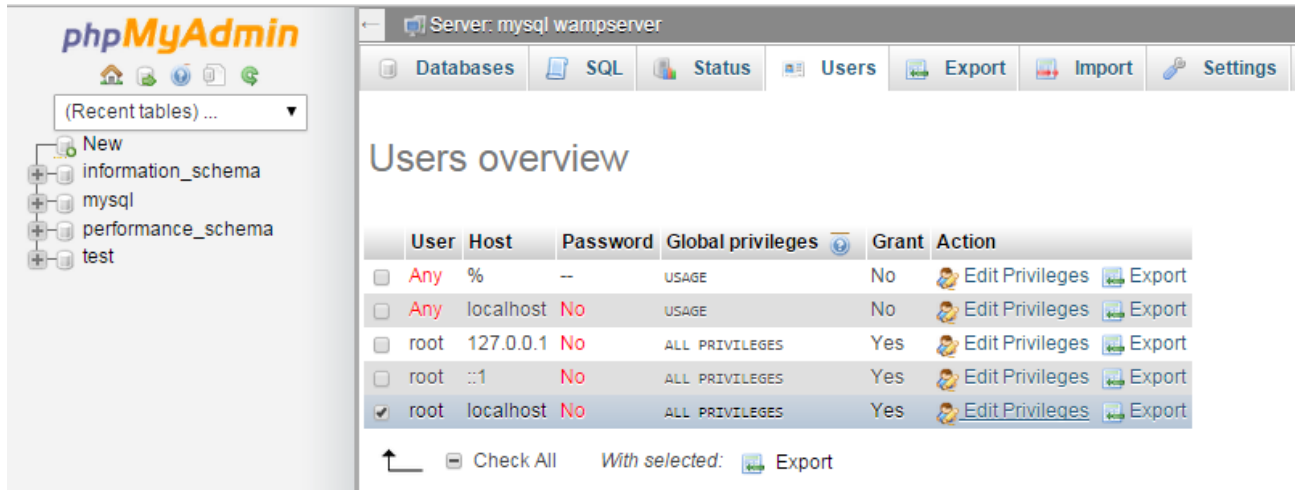


Figure 6: Changing root user privilege

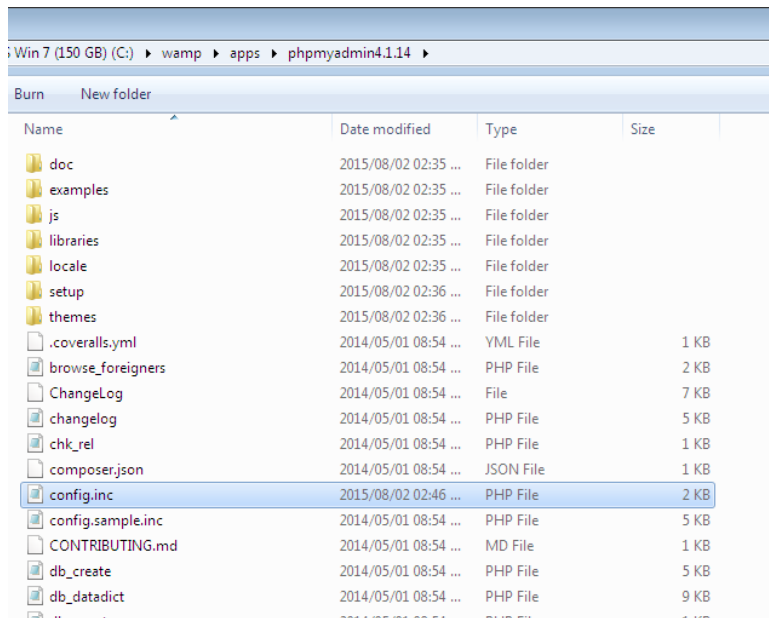


Figure 7: Access config.inc folder

Save the changes you have made and exit out of the editor. Go back to your toolbar where wampserver is located and click on 'restart all services'. Refresh localhost on your browser. Click on 'phpmyadmin'. You should now be prompted for a username and password. Your username is 'root' (since we did not change it from the default one) and your password is the MySQL password that you created.

ii. Setting up a password to access files stored in the Web Server

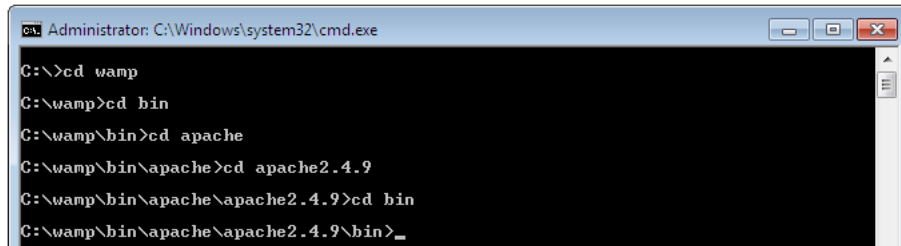
Now in case you want to share files from your server, you do not want just anyone to be able to access the files. So let us password-protect your files. For example, assume that you would like to store music files on your server and share them with others. First, decide where you would like to store your music files. In this illustration, the folder 'www' found in C:\wamp is used to store the music directory. We are assuming that you want your users to be able to access any folders that are stored inside the 'www' folder.

Next, using a command prompt, access the bin directory in the Apache folder (Figure 8). Then type:

Web Servers

```
htpasswd -c "C:\wamp\my-password-file.txt" username
```

my-password-file.txt is the file where you create the password to access the 'www' folder. **username** is the username that you create to access the 'www' folder. Pick a username of your choice. **Note that the password file is not created inside the music folder.** After you press enter, you will be prompted to enter and re-enter a password. Create a password of your choice. This is the password that will be associated with the username you have just created, and the combination will be used to access the 'www' folder.



```
Administrator: C:\Windows\system32\cmd.exe
C:\>cd wamp
C:\wamp>cd bin
C:\wamp\bin>cd apache
C:\wamp\bin\apache>cd apache2.4.9
C:\wamp\bin\apache\apache2.4.9>cd bin
C:\wamp\bin\apache\apache2.4.9\bin>_
```

Figure 8: Accessing Apache folder via command prompt

Now we want to apply the login to your music folder. Open a new file in a plain text editor like Notepad. Copy and paste the following into it:

```
AuthType Basic
```

```
AuthName "This is a private area, please log in"
```

```
AuthUserFile "c:\wamp\my_password_file.txt"
```

```
AuthGroupFile /dev/null
```

```
require valid-user
```

You can replace the message **'this is a private area, please log in'** with your own security message that you would like users to see. Save this file in the 'www' folder which is our server root folder. There are two important things to note when saving this file:

1. Save this file as `.htaccess` (including the dot).
2. If using an html editor such as Notepad put quotation marks around the name, thus `".htaccess"`. This way, it will not be saved as a text file.

Now when you refresh your localhost on the browser, you should be prompted for login details.

2.1.4 Testing Applications

Sharing files stored on your WampServer

If you would like to share the music files that you have stored on your Web Server, simply create a directory called 'music' inside the 'www' folder (Figure 9). Put the music files that you would like to share in the 'music' folder. Type <http://localhost/music/> on your address bar and you should be able to see a listing of your music.

Web Servers

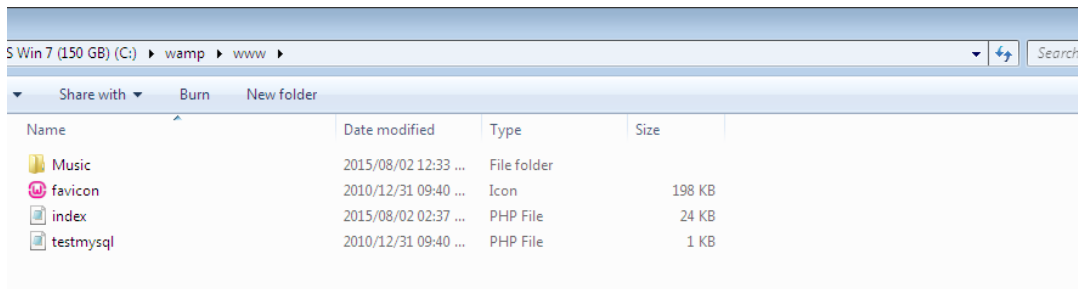


Figure 9: Music folder to be shared

Take note of your computer's IP address. Go to your wampserver toolbar, left-click and click 'Put online'. Your server can now be accessed on the Web.

Share the address <http://xxx.xxx.xx.xxx/music/> with someone else to try on another computer. 'xxx' represents the numbers in your IP address. The user will be prompted for the login details that you setup in the last section. Note that this may not work if you are sharing the address with someone outside of your network who is behind a firewall.

We shall soon see how to write HTML files that can be accessed via the Apache HTML server.

The next section illustrates how to set up an Apache Tomcat server on Windows.

Testing a 'Hello World' Application

Let us try a simple example to test that the websites we will create will work on this web server. Open Notepad and type the following code to print 'hello world' and save it as `hello.php` in the 'www' folder. Then access this file from your browser using your ip address or `localhost/hello.html`. You should get a 'hello world' output on your browser.

```
<html>
<head>
  <title>HTML Test</title>
</head>
<body>
  Hello world
</body>
</html>
```

2.2 Apache Tomcat on Windows

Apache Tomcat is a Java-capable HTTP server, which is able to execute special Java programs known as Java Servlet and Java Server Pages (JSP). It is also able to execute HTML files just like Apache HTTP.

2.2.1 Requirements

To illustrate the steps below a Windows 7 64-bit computer was used. The Windows computer had Internet access. Tomcat 8 was used for this installation. You need to have the latest Java JDK version installed. The recommended JDK version for Tomcat 8 is `jdk1.8`. Make sure that your computer is updated with this version. Go to java.com to download the latest version of Java.

The steps below are for a fresh installation of Apache Tomcat (assumes that Tomcat had not been installed before).

2.2.2 Tomcat Setup

i. Download and Install Tomcat

Go to <http://tomcat.apache.org/> and download the latest version of Tomcat (Tomcat 9 at the time of writing this). Under Download on the left click on 'Tomcat 9' and under Binary Distribution click on 'Core' and you will see various packages. Click on the package applicable to you. For a Windows 64-bit computer click on '64-bit Windows zip'. Download this file.

Unzip the downloaded file to a directory of your choice. It is recommended not to unzip to the desktop as it is a difficult directory to locate from a command prompt. For this illustration a folder named 'tomcat' was created under drive D, hence D:/tomcat. The zipped file was extracted to this location. It is recommended to rename the unzipped file from the default name, for example, rename to 'apachetomcat' (Figure 10).

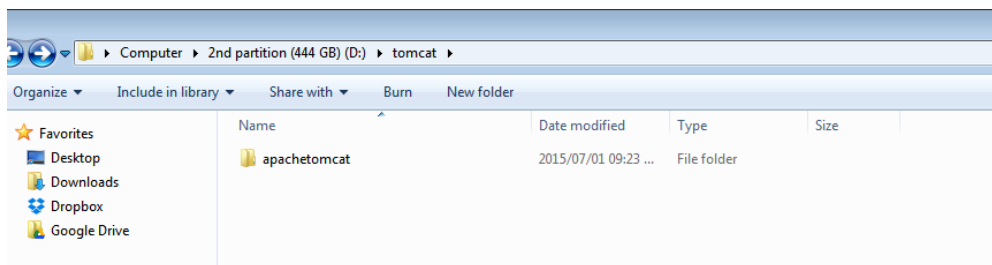


Figure 10: Creating folders to store tomcat files

ii. Create an Environment Variable

First, take note of the directory into which JDK was installed. The default is "C:\Program Files\Java\jdk1.8.0_{xx}", where {xx} is the latest upgrade number. It is important to verify your JDK installed directory before you proceed further. Start the command prompt and type 'set JAVA_HOME' to check if the environmental variable had been set. If not, you will get the message 'Environment Variable JAVA_HOME not set'. If JAVA_HOME is set, check if it is set to your JDK installed directory correctly (For example in Figure 11). If not, proceed to set the environmental variable as below.

To set the environment variable JAVA_HOME in Windows 7: Press "Start" button > Control Panel > System & Security > System > Advanced system settings > Switch to "Advanced" tab > Environment Variables > System Variables > "New" (or "Edit" for modification) > In "Variable Name", enter "JAVA_HOME" > In "Variable Value", enter your JDK installed directory (e.g., "c:\Program Files\Java\jdk1.8.0_51"). Click OK. To verify, **restart** the command prompt and type 'set JAVA_HOME'. You should now see the output as in Figure 11.

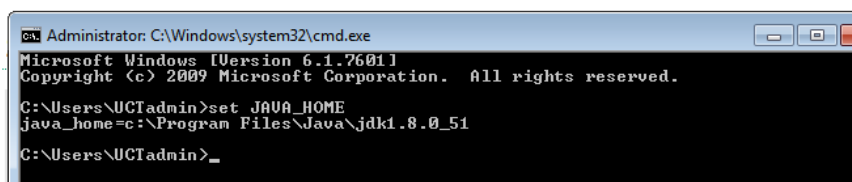


Figure 11: Checking environmental variables for JAVA_HOME

iii. Configure Tomcat Server

The Tomcat configuration files are located in the "conf" sub-directory of your Tomcat installed directory, e.g. "D:\tomcat\apachetomcat\conf". There are 4 configuration XML files: server.xml, web.xml, context.xml and tomcat-users.xml. Make a BACKUP of the configuration files before you proceed.

Set the TCP Port Number

Use an HTML text editor (e.g., NotePad++) to open the configuration file "server.xml", under the "conf" sub-directory of Tomcat installed directory.

The default TCP port number configured in Tomcat is 8080, you may choose any number between 1024 and 65535, which is not used by an existing application. We shall choose 8888 in this article. Locate the following lines that define the HTTP connector, and change port="8080" to port="8888". Save the file and exit.

```
<Connector port="8888" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

Enabling Directory Listing

Again, use an HTML text editor to open the configuration file "web.xml", under the "conf" sub-directory of Tomcat installed directory.

We shall enable directory listing by changing "listings" from "false" to "true" for the "default" servlet. Locate the following lines that define the "default" servlet; and change the "listings" from "false" to "true". Save and exit.

```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
    <init-param>
        <param-name>debug</param-name>
        <param-value>0</param-value>
    </init-param>
    <init-param>
        <param-name>listings</param-name>
        <param-value>true</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Enabling Automatic Reload

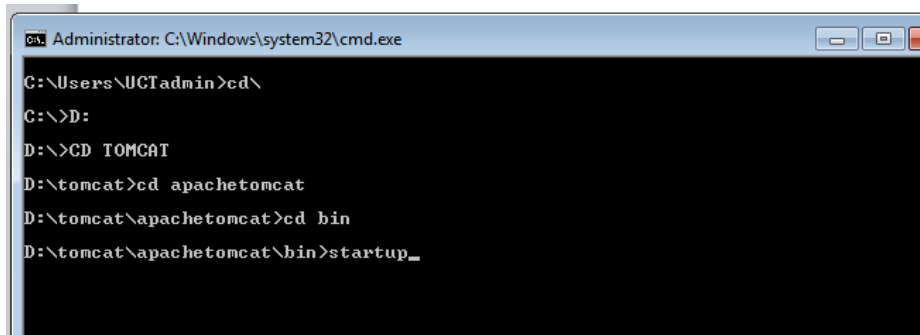
We shall add the attribute reloadable="true" to the <Context> element to enable automatic reload after code changes. Again, this is handy for test system but not for production, due to the overhead of detecting changes. Open context.xml and locate the <Context> start element, and change it to <Context reloadable="true">. Save and exit.

```
<Context reloadable = "true">
..... </Context>
```

iv. Start Tomcat Server

Launch a CMD shell. Set the current directory to the tomcat directory\bin", and run "startup.bat" as in Figure 12:

Web Servers



```
Administrator: C:\Windows\system32\cmd.exe
C:\Users\UCTadmin>cd\
C:\>D:
D:\>CD TOMCAT
D:\tomcat>cd apachetomcat
D:\tomcat\apachetomcat>cd bin
D:\tomcat\apachetomcat\bin>startup_
```

Figure 12: Starting tomcat from command prompt

A new Tomcat console window appears (look out for the Tomcat's port number (double check that Tomcat is running on port 8888). Future error messages will be sent to this console. Output messages from related Java programs are also sent to this console. If you want to shut down the server type 'shutdown' in place of 'startup'.

Start a browser. Issue URL "http://localhost:8888" to access the Tomcat server's welcome page. For users on the other machines over the net, they have to use the server's IP address or DNS domain name or hostname in the format of "http://serverHostnameOrIPAddress:8888". Note that this may not work if you are sharing the address with someone outside of your network who is behind a firewall. If everything is setup correctly, you should see the screen in Figure 13.

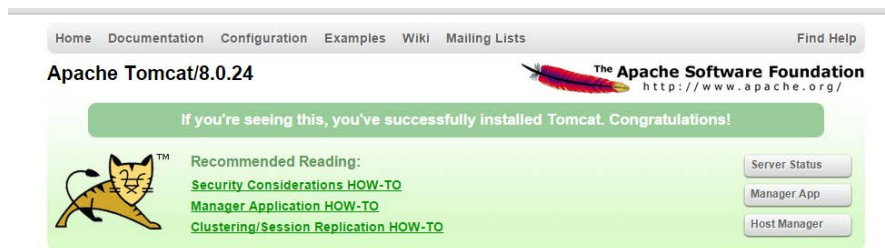


Figure 13: Tomcat home screen on local host

Install Tomcat's Sample Web Application

Go to: <http://localhost:8888/docs/>

Click the link: "3. First web application"

Click "Example App" under contents on the left side of the screen.

Click on the link "here" to download their "Sample Application". (Download link:

<http://localhost:8888/docs/appdev/sample/sample.war>)

Save to this location: C:\Tomcat\apachetomcat\webapps

Give the Tomcat container a minute and it will automatically extract the WAR file and create a Web Application called "Sample".

Test your install: <http://localhost:8888/sample>

If you wish to create your own directory under webapps, choose a *name* for your webapp. Let's call it "myapps".

Go to Tomcat's "webapps" sub-directory. Create the following directory structure for you webapp "myapps":

1. Under Tomcat's "webapps", create your webapp *root* directory "myapps" (i.e., "tomcat\apachetomcat\webapps\myapps").
2. Under "myapps", create a sub-directory "WEB-INF" (case sensitive, a "dash" not an underscore) (i.e., "tomcat\apachetomcat\webapps\myapps\WEB-INF").

3. Under "WEB-INF", create a sub-sub-directory "classes" (case sensitive, plural) (i.e., "tomcat\apachetomcat\webapps\myapps\WEB-INF\classes").

Restart your tomcat server to pick up the changes. Then on your browser type <http://localhost:8888/myapps/>

You should see the directory listing of the directory "tomcat\apachetomcat\webapps\myapps", which shall be empty (provided you have enabled directory listing in web.xml earlier).

We shall soon see how to write HTML and PHP files that can be accessed via the Tomcat server.

2.2.3 Testing Applications

Sharing files stored on your Tomcat Server

If you would like to share the music files that you have stored on your Web Server, simply create a directory called 'music' insider the 'myapps' folder. Put the music files that you would like to share in the 'music' folder. Type <http://localhost:8888/myapps/music/> on your address bar and you should be able to see a listing of your music.

Testing a 'Hello World' Application on Tomcat Server

Let us try a simple example to test that the websites we will create will work on this web server. Open Notepad and type the following code to print 'hello world' and save is as hello.html in the 'myapps' folder. Then access this file from your browser using your ip address or <http://localhost:8888/myapps/hello.html>. You should get 'hello world' output on your browser.

```
<html>
<head>
<title>HTML Test</title>
</head>
<body>
Hello world
</body>
</html>
```

2.3 Lampserver and Apache HTTP on Ubuntu 15.04

In this section, you will learn how to set up a Web Server on Ubuntu 15.04. The steps in this section will illustrate how to use Apache HTTP. The next section will illustrate the setup for Apache Tomcat. LAMP stack is a group of open source software used to get web servers up and running. The acronym stands for Linux, Apache, MySQL or MariaDB, and PHP. Since I assume that your computer is already running Ubuntu, the Linux part is taken care of.

2.3.1 Requirements

To illustrate the steps below a 64-bit computer running Ubuntu 15.04 was used. The computer was connected to a local area network (LAN) with Internet access. You also need to know your server IP address. You can find out your IP address by right clicking the network icon in the notification area and clicking Connection Information, or by running *ifconfig -a* on the terminal. To log into your server, you will need to know the password for the "root" user's account. First, run the command below to update your server, before which you will be prompted to enter your root password.

```
sudo apt-get update
```

2.3.2 Installing Apache, PHP and MySQL

Run the commands below to install Apache2 Web Server and wait for completion of installation.

```
sudo apt-get install apache2
```

Run the commands below to start the Apache2 Web Server.

```
sudo service apache2 start
```

At this time, if you browse to the server using its IP address, you'll see Apache2 default page for Ubuntu as in Figure 14. This is how you also tell the server is up and functioning.

The next step is to install PHP as well as its module to enable PHP apps or web services to function. There are hundreds of PHP modules, but these few will get most web services started. To install PHP and other modules, run the commands below.

```
sudo apt-get install php5 php5-mysql php5-curl php5-gd php5-snmp php5-mcrypt  
php5-tidy php5-xmllrpc libapache2-mod-php5
```

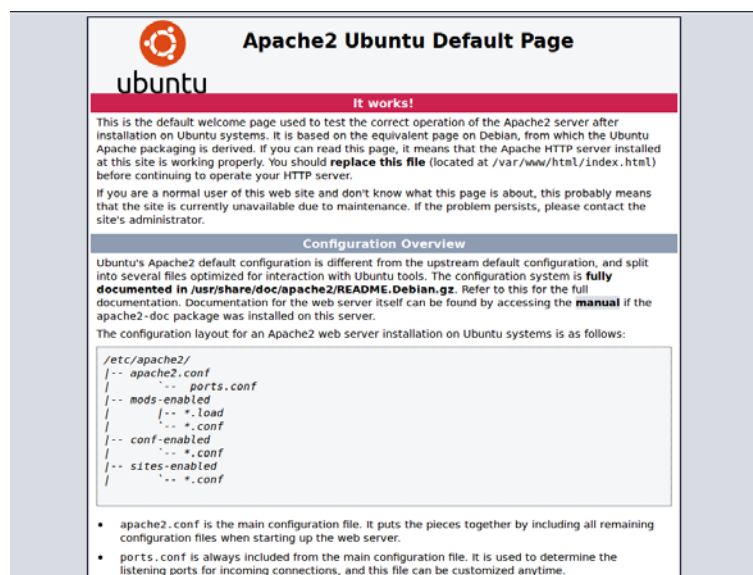


Figure 14: Successful Apache Installation

After installing the above modules, go to Apache root directory. It can be found at `/var/www/html` in Ubuntu. There create a file called **phpinfo.php**. Then in that file, add the lines below.

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

Save the file, restart Apache and browser to the server IP address followed by `phpinfo.php`. (ex. <http://Your IP address/phpinfo.php>). There you'll find PHP information page such as Figure 15. This is how you also know PHP is functioning.

Web Servers

Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/05-opcache.ini, /etc/php5/apache2/conf.d/10-pdo.ini, /etc/php5/apache2/conf.d/20-json.ini
PHP API	20131106
PHP Extension	20131226
Zend Extension	220131226
Zend Extension Build	API220131226.NTS
PHP Extension Build	API20131226.NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	enabled
Registered PHP Streams	https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

This program makes use of the Zend Scripting Language Engine:
Zend Engine v2.6.0, Copyright (c) 1998-2014 Zend Technologies
with Zend OPcache v7.0.4-dev, Copyright (c) 1999-2014, by Zend Technologies




Figure 15: Successful PHP Installation

The next step is to install MySQL. Run the commands below. Wait for completion.

```
sudo apt-get install mysql-server mysql-client
```

After installing the database server, you can start it using the commands below.

```
sudo systemctl start mysql
```

When it's started, run the commands below to configure the database server.

```
sudo mysql_secure_installation
```

When prompted, follow the options below and type your root password.

Next, choose Yes for the rest of the prompts until you're done.

- Enter current password for root (enter for none): Type root password
- Change the root password? **N**
- Remove anonymous users? **Y**
- Disallow root login remotely? **Y**
- Remove test database and access to it? **Y**
- Reload privilege tables now? **Y**

Restart Apache2 and you're done.

At this time, Apache2, PHP5 and other modules and MySQL database server should be installed and functioning. Your Ubuntu server is ready for any open source application that supports the LAMP stack.

You may wish to install PhpMyAdmin, which is a web interface through which you can easily manage/administer your MySQL/MariaDB databases. The installation can be completed with the following command:

```
sudo apt-get install phpmyadmin
```

Upon installation you will be asked to select the web server you are using. Select "Apache" and continue. Next you will be asked if you wish to configure phpmyadmin with dbconfig-common. Select "Yes". You need to perform one more step so that you can be able to access phpmyadmin from your Apache server. Run the following command.

```
sudo ln -s /usr/share/phpmyadmin /var/www/html
```

Web Servers

If you now access phpmyadmin on your browser through <http://Your IP address/phpmyadmin> you should see the same window in Figure 16 and you can be able to log in using your MySQL username and password that you created.



Figure 16: Phpadmin home page

2.3.3 Testing Applications

Sharing files stored on your Tomcat Server

If you would like to share the music files that you have stored on your Web Server, create a directory called 'music' insider the 'var/www/html' folder.

Put the music files that you would like to share in the 'music' folder. Type <http://localhost/music/> on your address bar and you should be able to see a listing of your music.

Testing a 'Hello World' Application on Tomcat Server

Let us try a simple example to test that the websites we will create will work on this web server. Type the following command on the terminal to open the *gedit* editor.

```
sudo gedit
```

In the *gedit* editor, type the following code to print 'hello world' and save is at hello.html under var/www/html Then access this file from your browser using your ip address or localhost/hello.html. You should get 'hello world' as output in your browser.

```
<html>
<head>
<title>HTML Test</title>
</head>
<body>
Hello world
</body>
</html>
```

2.4 Apache Tomcat on Ubuntu 15.04

Make sure that Java JDK is installed on your machine. For this example, JDK-7 was installed. First, head-on-over to the Apache Tomcat 8 Download site. Then, under the heading 8.0.28 (the current version as of November 2015), or whichever is the newest version at the time you read this chapter, you'll see Binary Distributions. Under Binary Distributions you'll see Core and then under Core, you will see tar.gz. Right click on tar.gz and copy the URL link location.

Web Servers

In the terminal use the URL you copied thus:

```
wget http://apache.is.co.za/tomcat/tomcat-8/v8.0.28/bin/apache-tomcat-8.0.28.tar.gz
```

After the download completes, decompress the file using the following command:

```
tar xvzf apache-tomcat-8.0.28.tar.gz
```

Now, move the file into a proper location using the following command. I am moving it to a folder called *opt*.

```
mv apache-tomcat-8.0.28 /opt
```

Now let's set the environment variables in *.bashrc* by typing the following command:

```
vim ~/.bashrc
```

Once in the vim editor, type 'i' in order to enter the insert mode. Add this information to the end of the file. After that press **esc** on the keyboard to go back to normal mode. Next press ':' (the colon) and this will drop the cursor to the bottom of the terminal. Here you save your changed by typing 'w' and press enter, and then you can type 'q' to quit vim.

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
export CATALINA_HOME=/opt/ apache-tomcat-8.0.28
```

Make the changes effective by running the following command:

```
. ~/.bashrc
```

Tomcat should now be installed and configured for your server. To activate Tomcat, run the following script on the terminal. Note that the commands are case sensitive.

```
$CATALINA_HOME/bin/startup.sh
```

You can verify that Tomcat is installed correctly by typing <http://localhost:8080> in your browser. You should see the window indicating successful installation.

2.4.1 Testing Applications

Sharing files stored on your Tomcat Server

If you would like to share the music files that you have stored on your Web Server, simply create a directory called 'music' insider the 'webapps' folder. Put the music files that you would like to share in the 'music' folder. Type <http://localhost/music/> on your address bar and you should be able to see a listing of your music.

Testing a 'Hello World' Application on Tomcat Server

Let us try a simple example to test that the websites we will create will work on this web server. Type the following command on the terminal to open the *gedit* editor.

```
sudo gedit
```

In the *gedit* editor, type the following code to print 'hello world' and save is at *hello.html* under the *webapps* folder. Then access this file from your browser using your ip address or *localhost/hello.html*. You should get the output in Figure 17.

```
<html>
<head>
<title>HTML Test</title>
</head>
```


Web Servers

```
<body>  
Hello world  
</body>  
</html>
```

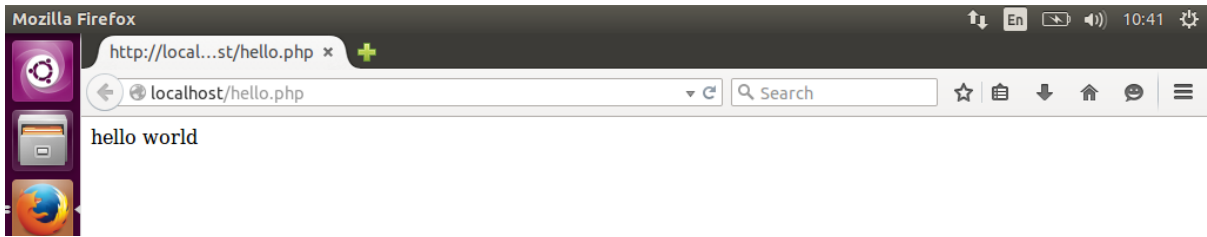


Figure 17: Hello World Output

Eclipse IDE Tutorial

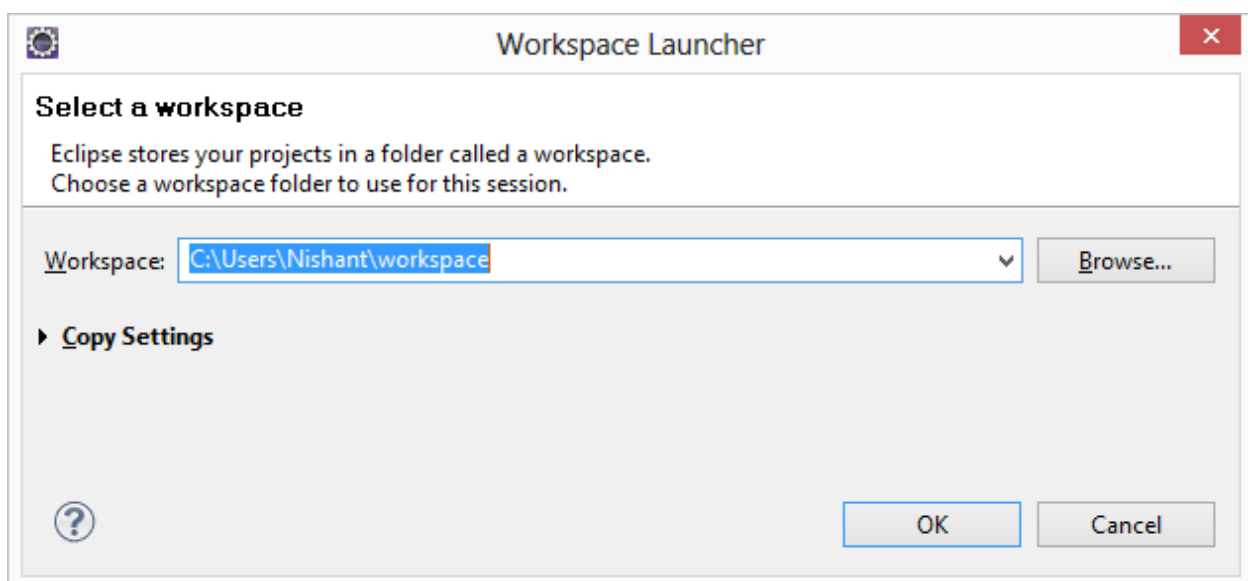
To start Eclipse, double-click on the eclipse.exe (Microsoft Windows) or eclipse (Linux / Mac) file in the directory where you unpacked Eclipse.

The system will prompt you for a *workspace*. The *workspace* is the physical location (file path) you are working in. Your projects, source files, images and other artifacts can be stored and saved in your workspace. The *workspace* also contains preferences settings, plug-in specific metadata, logs etc.

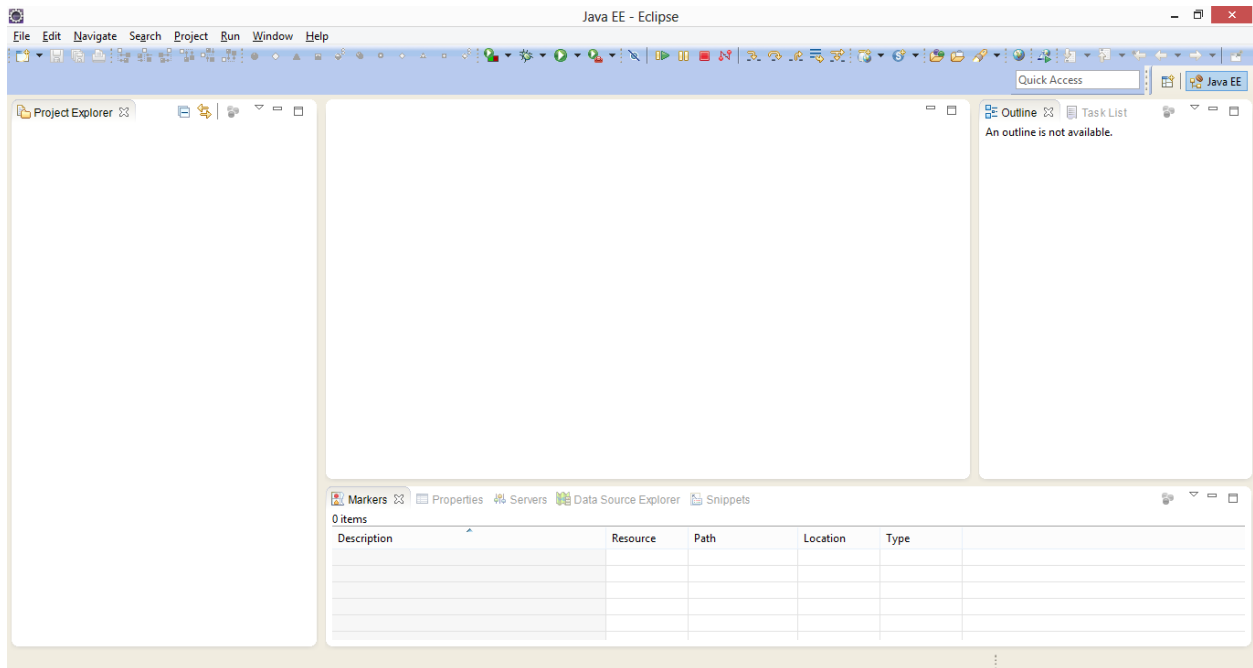
You typically use different *workspaces* if you require different settings for your project or if you want to divide your projects into separate directories.

You can choose the workspace during startup of Eclipse or via the menu (*File* → *Switch Workspace* → *Others*).

Select an empty directory and click the *OK* button.



Once you select the workspace, the application will look similar to the following screenshot.



Eclipse projects

An Eclipse project contains source, configuration and binary files related to a certain task and groups them into buildable and reusable units. An Eclipse project can have *natures* assigned to it which describe the purpose of this project. For example, the Java *nature* defines a project as Java project. Projects can have multiple natures combined to model different technical aspects.

Projects in Eclipse cannot contain other projects.

Views and editors - parts

Parts are user interface components which allow you to navigate and modify data. A part can have a dropdown menu, context menus and a toolbar.

Parts can be freely positioned in the user interface.

Parts are typically classified into *views* and *editors*. The distinction into views and editors is not based on technical differences, but on a different concept of using and arranging these parts.

A view is typically used to work on a set of data, which might be a hierarchical structure. If data is changed via the view, this change is typically directly applied to the underlying data structure. A view sometimes allows us to open an editor for a selected set of data.

An example for a view is the *Project Explorer*, which allows you to browse the files of Eclipse projects. If you change data in the *Project Explorer*, e.g., renaming a file, the file name is directly changed on the file system.

Editors are typically used to modify a single data element, e.g., the content of a file or a data object. To apply the changes made in an editor to the data structure, the user has to explicitly save the editor content.

For example, the *Java* editor is used to modify Java source files. Changes to the source file are applied once the user selects the *Save* command. A dirty editor tab is marked with an asterisk to the left of the modified name of the file.

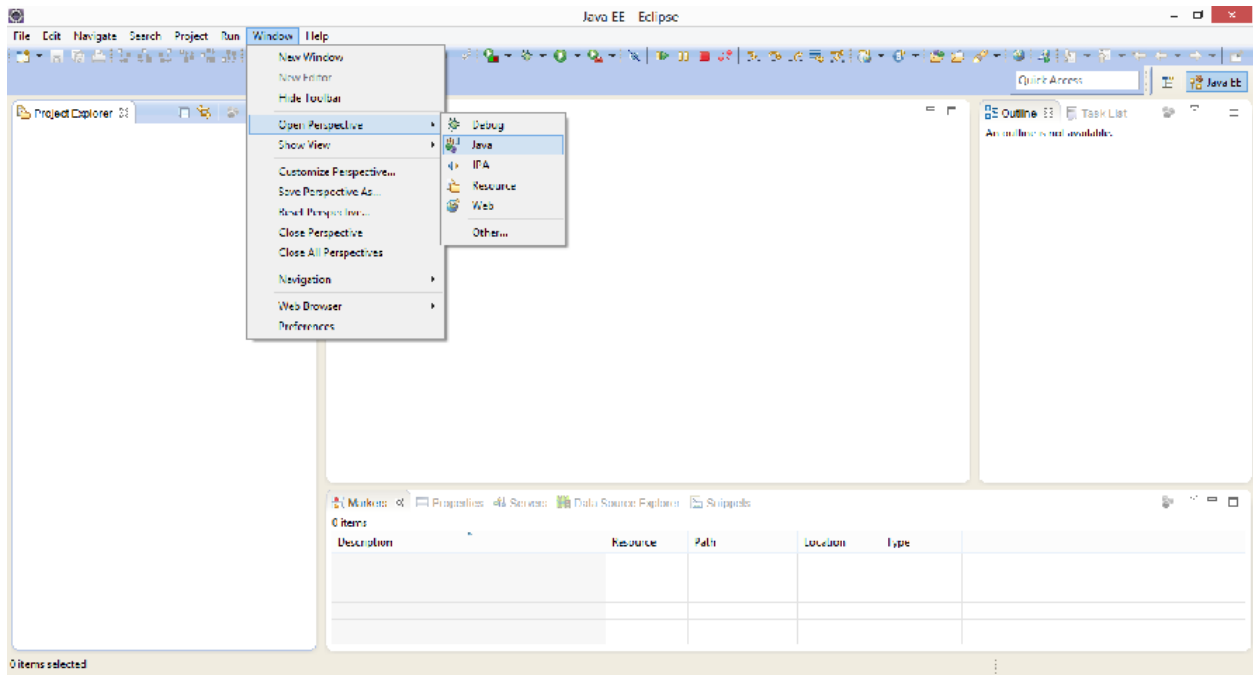
Perspective

A *perspective* is a visual container for a set of parts. Perspectives can be used to store different arrangements of parts. For example, the Eclipse IDE uses them to layout the views appropriate to the task (development, debugging, review, ...) the developer wants to perform.

Open editors are typically shared between perspectives, i.e., if you have an editor open in the *Java* perspective for a certain class and switch to the *Debug* perspective, this *editor* stays open.

You can switch *Perspectives* via the *Window* → *Open Perspective* → *Other...* menu entry.

The main perspectives used for Java development are the *Java* perspective and the *Debug* perspective.



You can change the layout and content within a perspective by opening or closing parts and by re-arranging them.

To open a new part in your current perspective, use the *Window* → *Show View* → *Other...* menu entry.

Perspectives in Eclipse

Eclipse provides different *perspectives* for different tasks. The available *perspectives* depend on your installation.

For Java development you usually use the *Java Perspective*, but Eclipse has much more predefined *perspectives*, e.g., the *Debug perspective*.

Eclipse allows you to switch to another *perspective* via the *Window* → *Open Perspective* → *Other...* menu entry.

Java perspective and Package Explorer

The default *perspective* for Java development can be opened via *Window* → *Open Perspective* → *Java*.

On the left hand side, this perspective shows the *Project Explorer* view, which allows you to browse your *projects* and to select the components you want to open in an editor via a double-click.

For example, to open a Java source file, open the tree under *src*, select the corresponding *.java* file and double-click it. This will open the file in the default Java *editor*.

The following picture shows the Eclipse IDE in its standard Java *perspective*. The *Project Explorer* view is on the left. In the middle you see the open *editors*. Several *editors* are stacked in the same container and you can switch between them by clicking on the corresponding tab. Via drag and drop you can move an editor to a new position in the Eclipse IDE.

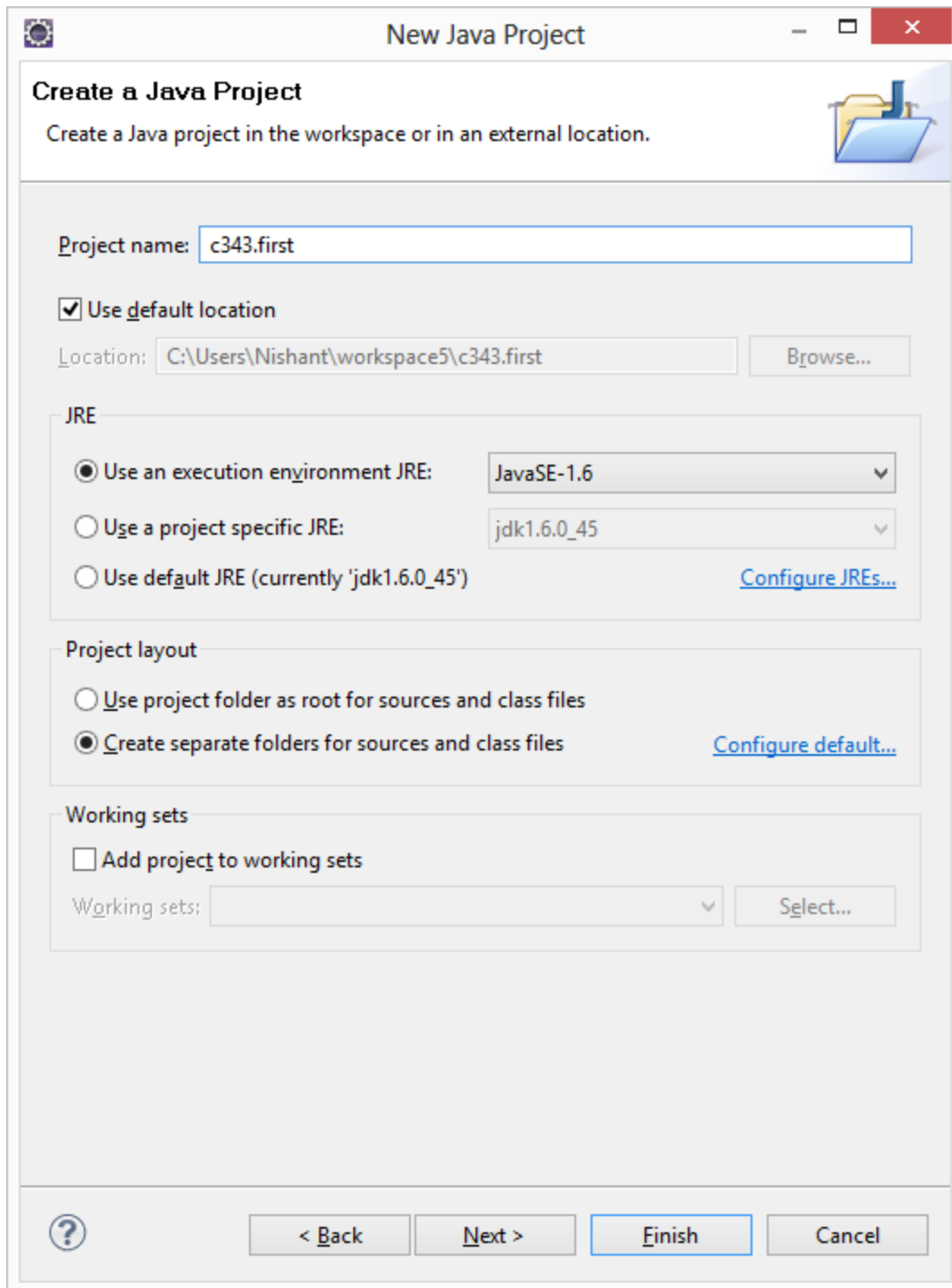
To the right and below the editor area you find more *views* which were considered useful by the developer of the perspective. For example, the *Javadoc* view shows the Javadoc of the selected class or method.

Create your first Java program

Create project

Select *File* → *New* → *Java project* from the menu.

Enter *c343.first* as the project name. Select the *Create separate folders for sources and class files* flag.

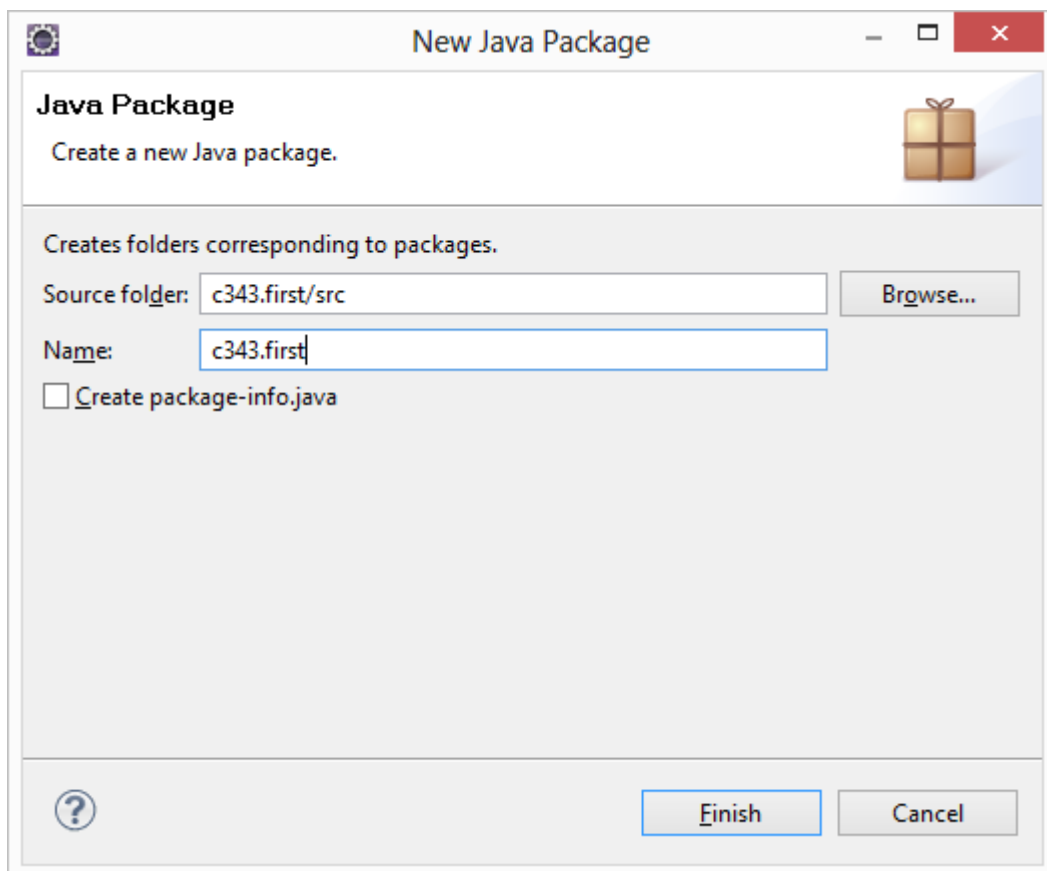


Press the *Finish* button to create the project. A new project is created and displayed as a folder. Open the *c343.first* folder and explore the content of this folder.

Create package

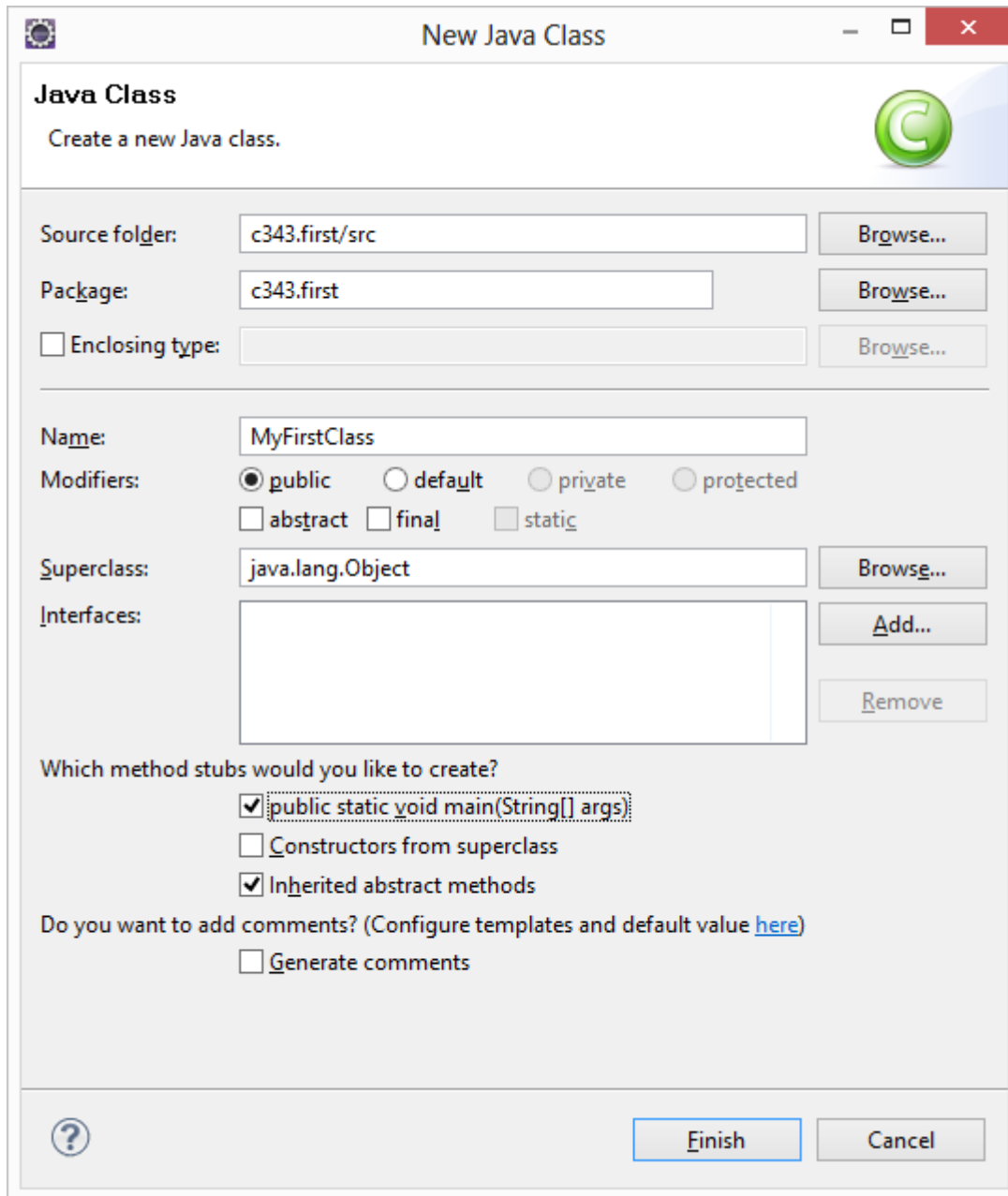
In the following step you create a new package. A good convention for the project and package name is to use the same name for the top level package and the project. For example, if you name your project `com.example.javaproject` you should also use `com.example.javaproject` as the top-level package name.

To create the `c343.first` package, select the `src` folder, right-click on it and select *New* → *Package*. Enter the name of your new package in the dialog and press the *Finish* button.



Create Java class

Right-click on your package and select *New* → *Class*. Enter `MyFirstClass` as the class name and select the *public static void main (String[] args)* checkbox. Press the *Finish* button.



This creates a new file and opens the Java *editor*. Change the class based on the following listing.

```
package c343.first;

public class MyFirstClass {

    public static void main(String[] args) {
        System.out.println("Hello Eclipse!");
    }
}
```

You could also directly create new packages via this dialog. If you enter a new package in this dialog, it is created automatically.

Run your project in Eclipse

Now run your code. Either right-click on your Java class in the *Package Explorer* or right-click in the Java class and select *Run-as → Java application*.

Eclipse will run your Java program. You should see the output in the *Console* view.

Congratulations! You created your first Java project, a package, a Java class and you ran this program inside Eclipse.