OPEN SOURCE PROGRAMMING

e-NOTES

B.Sc. Computer Science / B.C.A

Dr.P.Rizwan Ahmed
Vice Principal (Academic) & HOD

Department of Computer Applications &
PG Department of Information Technology

Mazharul Uloom College, Ambur-635802
(Managed by Ambur Muslim Educational Society (AMES),
Affiliated to Thiruvalluvar University, Vellore)

Unit - II

Introduction

Linux is a popular version of the UNIX Operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Linux is an open source operating system (OS). An operating system is the software that directly manages a system's hardware and resources, like CPU, memory, and storage. The OS sits between applications and hardware and makes the connections between all of your software and the physical resources that do the work.

Linux Essential Commands

ls command

The ls command lists the directory content. If no directory is specified, the command will display the content of the working directory.

pwd command

The *pwd* command is used to print the path of the current directory.

mkdir command

To create a new directoy, the *mkdir* command is used. You must specify the name of the directory. If no path is specified, the directory is created inside the working directory.

echo command

The *echo* command is used to to output text to the screen. You simply type *echo* and then the text you would like to display.

whoami command

The *whoami* command displays the username of the current user.

cd command

To change the current working directory we use the *cd* command. You must specify the path of the directory you would like to access.

File System Concepts

Linux file system has a hierarchal file structure as it contains a root directory and its subdirectories. All other directories can be accessed from the root directory. A partition usually has only one file system, but it may have more than one file system.

Linux File System or any file system generally is a layer that is under the operating system that handles the positioning of your data on the storage; without it, the system cannot knows which file starts from where and ends where.

Linux File System Types

When you try to install Linux you will see that Linux offers many file systems like these:
Ext, Ext2, Ext3, Ext4, JFS, XFS, btrfs and swap

- **Ext**: an old one and no longer used due to limitations.
- **Ext2**: first Linux file system that allows two terabytes of data allowed.
- **Ext3**: came from Ext2, but with upgrades and backward compatibility.
- **Ext4**: faster and allow large files with significant speed.
- **JFS**: old file system made by IBM. It works very well with small and big files, but it failed and files corrupted after long time use, reports say.
- **XFS**: old file system and works slowly with small files.
- **Btrfs:** made by Oracle. It is not stable as Ext in some distros, but you can say that it is a replacement for it if you have to. It has excellent performance.

Standard Files

stdin

The primary input channel for the program. Default source is the user's keyboard, but it can easily be switched to be from another process via a pipe (|) or a file via a file redirection (<).

stdout

The primary output channel for program's data output. Default destination is the user's screen, but it can easily be switched to another process via a pipe (|) or a file via a file redirection (>).

stderr

Output channel for error messages. Default destination is the user's screen, but it can be switched to another process via a pipe (|) or a file via a file redirection (>).

Linux Security Model

Based on the UNIX model, all files, directories, running processes and system resources on the Linux system are associated with a userand group. The security can be set independently for the user, or owner, and group.

A third set of permissions is maintained for everyone on the system who is neither the owner nor in the group associated with a resource. This is commonly referred to as the permissions for 'other' users.

These security levels, user, group and other, each have a set of permissions associated with them. Typical permissions are read, write and execute and depending on the type of resource, these will determine what a given user is allowed to do with the resource.

In this example, the ls -l command is used to show the ownership and permissions of the file status.sh.

The permissions portion of the output breaks down further to indicate the level of access for the owner, group and other users. In this example, the owner has read, write and execute, the group has read and execute and other has only read.

Each user on the system is associated with a primary group but can also belong to additional groups, adding to the flexibility of Linux security. A user has access rights granted to his or her primary group as well as any additional groups to which the user belongs.

Though we use user and group names are used to interact with the system, these identifiers are tracked within the system by ID numbers. The user ID (UID)and group ID (GID) numbers are associated with user and group names through the /etc/passwd and /etc/group files, respectively.

Vi Editor

The VI editor is the most popular and classic text editor in the Linux family. Below, are some reasons which make it a widely used editor –

1. It is available in almost all Linux Distributions
2. It works the same across different platforms and Distributions
3. It is user-friendly. Hence, millions of Linux users love it and use it for their editing needs

Nowadays, there are advanced versions of the vi editor available, and the most popular one is **VIM** which is **V**i **Im**proved. Some of the other ones are Elvis, Nvi, Nano, and Vile. It is wise to learn vi because it is feature-rich and offers endless possibilities to edit a file.

To work on VI editor, you need to understand **its operation modes**. They can be divided into two main parts.

Command mode

- The vi editor opens in this mode, and it only understands commands
- In this mode, you can, move the cursor and cut, copy, paste the text
- This mode also saves the changes you have made to the file
- Commands are case sensitive. You should use the right letter case.

Editor Insert mode

- This mode is for inserting text in the file.
- You can switch to the Insert mode from the command mode **by pressing 'i' on the keyboard**
- Once you are in Insert mode, any key would be taken as an input for the file on which you are currently working.
- To return to the command mode and save the changes you have made you need to press the Esc key

Partitions Creation

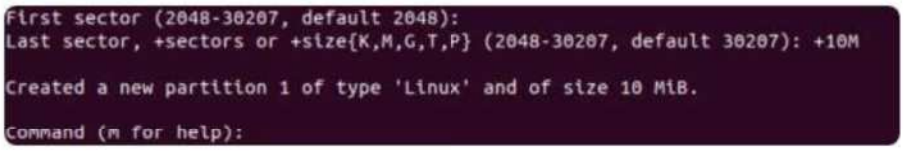
Our main objective here is to create a partition. To create a new partition, we use the **command 'n'**. This will prompt you to specify the type of partition which you wish to create.

If you wish to create a logical partition, choose 'l'. Alternatively, you can **choose 'p'** for a primary partition. For this tutorial, we will create a primary partition.

Now, we will be asked to specify the starting sector for our new partition. Press ENTER to choose the first available free sector on your system. Next, you'll be prompted to select the last sector for the partition.

Either press ENTER to use up all the available space after your first sector or specify the size for your partition

Sector Type



As shown in the screenshot above, we chose to create a 10 MB partition for this demonstration. Here 'M' specifies the unit as megabytes. You can use 'G' for gigabytes.

If you don't specify a unit, the unit will be assumed to be sectors. Hence +1024 will mean 1024 sectors from the starting sector.

Shell Introduction

A **shell program**, sometimes referred to as a shell script, is simply a program constructed of shell commands. Shell programs are interpreted each time they are run. This means each command is processed (i.e. executed) by the shell a single line at a time. This is different from languages such as C or C++, which are translated in their entirety by a compiler program into a binary image. A shell program may be simple and consist of just a few shell commands, or it may be very complex and consist of thousands of shell commands. The complexity of the shell program is in the hand of the programmer. In general, a shell program can be characterized by:

- shell programs consist of one or more primitive shell commands
- shell programs are created using your text editor of choice, e.g. vi or emacs
- shell programs are executed just as shell commands are, by typing the name of the program followed by the *[Enter]* key
- shell programs have permission modes as do any other file, and must have the correct permissions set to execute the program
- as with other programming languages, the shell language has the functionality to allow input & output, iteration, logical decision making, file creation and deletion, and system call capability
- shell programs are free format, as long as the syntax of each shell command is correct. This means that blank lines, indentation and abundant whitespace can be used freely.

As stated, a shell program is merely a file containing shell commands. Thus, if we wanted to write the venerable "hello world" program as a shell script, we could do the following:

1. use the editor to create the program, for simplicity we'll call hw (recall file extensions are not mandatory)

$ vi hw *[Enter]*

2. insert the following shell command in the hw file:

```
echo "Hello World!"
```

3. save and exit the editor program
4. run the hw program[3]
5. $ hw *[Enter]*
   ksh: hw: cannot execute - Permission denied

6. once the problem above is fixed, we run the script and see the following:
7. $ hw *[Enter]*
   Hello World!

You have now written your first successful shell program.

String Processing

Shell programming is heavily dependent on string processing. The term string is used generically to refer to any sequence of characters; typical examples of strings might be a line of input or a single argument to a command. Users enter responses to prompts, file names are generated, and commands produce output. Recurring throughout this is the need to determine whether a given string conforms to a given pattern; this process is called pattern matching. The shell has a fair amount of built-in pattern matching functionality.

Investigation and Managing Processes

What is a process?

A process is an instance of a running program. It is the job of Kernel to manage CPU time and other resources among multiple processes. Each process is given a unique PID (process ID). PID is allotted by Kernel when a process is born.

Listing Process:

ps (displays the following)

PID	TTY	TIME	CMD
22794	tty1	00:00:00	bash
22809	tty1	00:00:00	ps

Finding Process:

ps axu | grep tty2
(shows all process, running on all terminal by all users. Use grep for filtering)

ps -eaf | grep sshd

pgrep -u username
(to see PID related to specific user)

#pidof crond
(to see PID of specific process)

Signals: Signals are messages that are sent to processes with a command like **kill**. The advantage of signals is that they can be sent to a process even if it is not attached to a terminal. So if a web server or a graphical program whose interface has "frozen" can still be shut down by sending appropriate signal.

Sending signals to processes:

Process normally terminate on their own when they have completed.

Interactive application may need the user to issue a **quit** command.

CTRL + C also terminate the process, which send an interrupt (INT) signal to a process.

If a process doesn't terminate using above method, you can use **KILL** signal.
(Remember it should only be used when the above method doesn't work. Using KILL signal on a routine basis may cause zombie process and loose data)

5. Scheduling Priority:

Every running process has a scheduling priority, Whoever has got higher priority, gets more attention of the CPU.

The priority of process is set by altering niceness value.

The niceness value defaults to zero(0) but can be set from -20 (least value, highest piority) to 19 (highest value, least priority)

To set the niceness value to a specific value when starting a process, use nice -n as following:
nice -n 13 vim file.txt (to run vim at 13 niceness value)
nice -n -13 vim file.txt (to run vim at -13 niceness value)

To modify the niceness value of running processes, use renice command as following:
renice 15 PID (Where PID is process ID of running process)
renice -15 PID

Interactive Process Management Tools

Running top presents a list of the processes running on your system, updated every 5 seconds, you can use keystrokes to kill, renice, colorize processes, define the order to display etc. Press the ? key while in top to view the complete list of hotkeys.

Job Control

Jobs running in the foreground can be suspended: temporarily stopped, without being killed. To suspend a job, Press Ctrl + z. Once a process is suspended, it can be resume in the background using the bg command or resumed in the foreground using the fg command. When the job resumes, it will continue executing from the point at which it was suspended. It will not have to start over from the beginning.

man mkdir (press **ctrl + z** to suspend this job)
[1]+ stopped

bg	to resume the recent suspended job in background
bg 2	to resume the job 2 in background
jobs	to see all suspended job
fg 2	to resume the job 2 in foreground

Network Client

Network Client is a software that runs on a client computer and allows it to establish connectivity with services running on server computers. In Microsoft Windows 95 and Windows 98, the network client is one of several components that can be installed to provide connectivity with different kinds of networks. Without the appropriate client software, a workstation cannot access files and print resources or other resources on a network server.

Installing Application

The RPM Package Manager

All software on a Red Hat Enterprise Linux system is divided into RPM packages which can be installed, upgraded, or removed using the RPM package manager.

Package installation is never interactive. In contrast to package management on some other platforms, RPM's design does not provide interactive configuration of software as part of the package load process. RPM can perform configuration actions as part of the installation, but these are scripted not interactive. It is common for packages to install with reasonable default configurations applying. On the other hand some software installs in an un-configured state.

Applies to all software. On some other common platforms, the package management system applies only to part of the installed software. The scope of RPM include core operating systems as well as services and applications. It is common and desirable to run Red Hat Enterprise Linux systems such that all software installed falls under the management of RPM. This is a great aid for management and configuration control, since one framework applies for the management of every installed file.

No such thing as a patch. It is common on other platforms to have operating system updates released as software objects (eg. “service packs”) which represent incremental changes to a large number of installed component packages. RPM never does this. If part of any given software package is changed as part of an errata or bug fix, then that entire package will be re-released in

its entirety at a new version. The implications are that the installed state of an RPM-managed system can be described as the version number of all the installed components.

Software to be installed using rpm is distributed through rpm package files, which are essentially compressed archives of files and associated dependency information. Package files are named using the following format:
name-version-release.architecture.rpm
The version refers to the open source version of the project, while the release refers to Red Hat internal patches to the open source code.

The Yum Package Management Tool
The command-line utility yum gives you an easy way to manage the packages on your system:

```
[root@stationX ~]# yum install firefox
```

The above command will search the configured repositories for a package named firefox, and if found will install the latest version, pulling in dependencies if needed.

```
[root@stationX ~]# yum remove mypackage
```

The above command will try to remove the package named mypackage from your system. If any other package depends on mypackage yum will prompt you about this, giving you the option to remove those packages as well.

```
[root@stationX ~]# yum update [mypackage...]
```

If any packages are specified on the command-line yum will search the configured repositories for updated versions of those packages and install them. When no packages are specified yum will search for updates to all of your currently installed packages.

```
[root@stationX ~]# yum list available
```

The above command will list all packages in the yum repositories available to be installed.

Unit - III

Introduction to MY SQL

MySQL is an open-source relational database management system that works on many platforms. It provides multi-user access to support many storage engines and is backed by Oracle. So, you can buy a commercial license version from Oracle to get premium support services.

The features of MySQL are as follows:

- **Ease of Management** – The software very easily gets downloaded and also uses an event scheduler to schedule the tasks automatically.
- **Robust Transactional Support** – Holds the ACID (Atomicity, Consistency, Isolation, Durability) property, and also allows distributed multi-version support.
- **Comprehensive Application Development** – MySQL has plugin libraries to embed the database into any application. It also supports stored procedures, triggers, functions, views and many more for application development.
- **High Performance** – Provides fast load utilities with distinct memory caches and table index partitioning.
- **Low Total Cost Of Ownership** – This reduces licensing costs and hardware expenditures.
- **Open Source & 24 * 7 Support** – This RDBMS can be used on any platform and offers 24*7 support for open source and enterprise edition.
- **Secure Data Protection** – MySQL supports powerful mechanisms to ensure that only authorized users have access to the databases.
- **High Availability** – MySQL can run high-speed master/slave replication configurations and it offers cluster servers.
- **Scalability & Flexibility** – With MySQL you can run deeply embedded applications and create data warehouses holding a humongous amount of data.

The show Databases and Table

Show Database

The most common way to get a list of the MySQL databases is by using the mysql client to connect to the MySQL server and run the SHOW DATABASES command.

Access the MySQL server using the following command and enter your MySQL user password when prompted:

```
mysql -u user -p
```

From within the MySQL shell execute the following command:

```
Mysql> SHOW DATABASES;
```

The command will print a list of all the databases for which the user have some kind of a privilege granted to . The output will be similar to this:

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| opencart           |
+--------------------+
2 rows in set (0.00 sec)
```

Show Table

To get a list of the tables in a MySQL database, use the mysql client tool to connect to the MySQL server and run the SHOW TABLES command.

Execute the following command to get a list of all tables and views in the current database:

```
Mysql>SHOW TABLES;
```

The output will look something like this:

```
+---------------------------+
| Tables_in_database_name   |
+---------------------------+
| actions                   |
| permissions               |
| permissions_roles         |
| permissions_users         |
| roles                     |
| roles_users               |
| settings                  |
| users                     |
+---------------------------+
8 rows in set (0.00 sec)
```

The USE command

The USE statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The named database remains the default until the end of the session or another USE statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

The database name must be specified on a single line. Newlines in database names are not supported.
Making a particular database the default by means of the USE statement does not preclude accessing tables in other databases. The following example accesses the author table from the db1 database and the editor table from the db2 database:

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
  WHERE author.editor_id = db2.editor.editor_id;
```

Create Database and Tables

To create a new database in MySQL, you use the CREATE DATABASE statement with the following syntax:

```
CREATE DATABASE [IF NOT EXISTS] database_name
[CHARACTER SET charset_name]
[COLLATE collation_name]
```

Code language: SQL (Structured Query Language) (sql)

First, specify the database_name following the CREATE DATABASE clause. The database name must be unique within the MySQL server instance. If you try to create a database with a name that already exists, MySQL issues an error.

Second, to avoid an error in case you accidentally create a database that already exists, you can specify the IF NOT EXISTS option. In this case, MySQL does not issue an error but terminates the CREATE DATABASE statement instead.

Third, specify the character set and collation for the new database at creation time. If you omit the CHARACTER SET and COLLATE clauses, MySQL uses the default character set and collation for the new database.

The CREATE TABLE statement allows you to create a new table in a database.
The following illustrates the basic syntax of the CREATE TABLE statement:

```
CREATE TABLE [IF NOT EXISTS] table_name(
   column_1_definition,
   column_2_definition,
   ...,
   table_constraints
) ENGINE=storage_engine;
```

Example

```
CREATE TABLE IF NOT EXISTS tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    start_date DATE,
    due_date DATE,
    status TINYINT NOT NULL,
    priority TINYINT NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)  ENGINE=INNODB;
```

Describe Table

Even though DESCRIBE and EXPLAIN statements are synonyms, the DESCRIBE statement is used more to obtain information about a table structure while EXPLAIN statement is used to obtain a query execution plan.

The DESCRIBE statement is a shortcut for SHOW COLUMN statement:

```
DESCRIBE table_name;
```

is equivalent to this SHOW COLUMN statement:

```
SHOW COLUMNS FROM table_name;
```

Or you can also use the short form of describe:

```
DESC table_name
```

Select, Insert, Update, and Delete statement

Select

In MySQL, The SELECT statement is the most commonly used MySQL statement. You can use the SELECT statement to fetch or get data from one or more tables within the database. With SELECT statement of MySQL, you can fetch all fields of table or specified fields.

Syntax

In MySQL SELECT statement syntax is :-

```
SELECT field1, field2 … field n
FROM Table
[WHERE conditions];
```

Params

Fields :- It's a database column name. You can fetch one or more fields using SELECT statement.

Table :- It's a database table name.

WHERE :- It's a option. If You want to filter some data that time specify any condition using the WHERE clause.

SELECT Statement Example

```
SELECT users.id, users.name, users.age
FROM users
```

WHERE status = active;

Insert

In MySQL, You can use the INSERT INTO statement to insert data to your database table. With INSERT INTO statement of MySQL, you insert single row or multiple rows into database table.

Syntax

In MySQL INSERT INTO statement syntax is :-

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

Params

Fields :- It's a database column name. You can insert value into in it.
table_name :- It's a database table name.
Value :- It's your column values, that you want to insert into database

INSERT INTO Statement Example

Here, we will insert single row into database table name users.
INSERT INTO users(id,name) VALUES (1, 'test');
Here, we will insert multiple rows into database table name users.

```
INSERT INTO users
 (id, first_name, last_name)
 VALUES
 (5, 'wis', 'check'),
 (6, 'joy', 'son'),
 (7, 'kemal', 'case');
```

Update

In MySQL, You can use the UPDATE statement to update or change data to your database table. With UPDATE statement of MySQL, you can change or modify table data by using MySQL WHERE Clause.

Syntax
In MySQL UPDATE statement syntax is :-

```
UPDATE table_name SET field1=new-value1, field2=new-value2
```

[WHERE Clause]

Params
table_name: – This is the name of a database table.
Fields: – This is a database column name. Where you can update the value in it
Value: – These are your column values, which you want to update in the database.
WHERE: – In MySQL, where the clause is used to update specific rows in a table.

UPDATE Statement Example
Here, we will update data into database table using UPDATE AND WHERE Clause statement of MySQL.

```
UPDATE users
SET first_name = 'Michel'
WHERE id = 501;
```

Delete

In MySQL, You can use the DELETE statement to delete/remove all the data from database table.

You can delete/remove records single data from the databse table on the basis of conditions.
Syntax
In MySQL DELETE statement syntax is :-

```
DELETE FROM table_name;
```

OR

```
DELETE FROM table_name [WHERE Clause];
```

Params
table_name: – This is the name of a database table.
WHERE: – In MySQL, where clause is used to delete specific rows in a database table.

DELETE Statement Example
Here, we will delete/remove data into database table using DELETE AND WHERE Clause statement of MySQL.

```
DELETE FROM users
WHERE id = 15;
```

DELETE All Record From Table Example
In MySQL, You can use the following syntax for delete all records from databse table.

```
DELETE FROM users;
```

Table Joins

MySQL **JOINS** are used to retrieve data from multiple tables. A MySQL JOIN is performed whenever two or more tables are joined in a SQL statement.

There are different 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)

So let's discuss MySQL JOIN syntax, look at visual illustrations of MySQL JOINS, and explore MySQL JOIN examples.

INNER JOIN (simple join)

Chances are, you've already written a statement that uses a MySQL INNER JOIN. It is the most common type of join. MySQL INNER JOINS return all rows from multiple tables where the join condition is met.

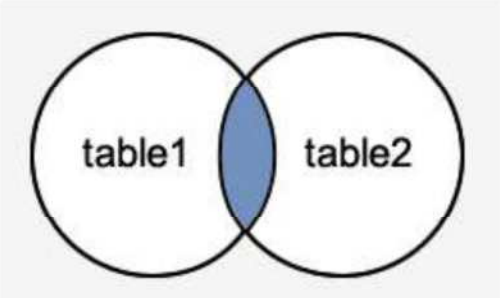
Syntax

The syntax for the INNER JOIN in MySQL is:

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

Visual Illustration

In this visual diagram, the MySQL INNER JOIN returns the shaded area:



The MySQL INNER JOIN would return the records where *table1* and *table2* intersect.

Example

Here is an example of a MySQL INNER JOIN:

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

This MySQL INNER JOIN example would return all rows from the suppliers and orders tables where there is a matching supplier_id value in both the suppliers and orders tables.

Let's look at some data to explain how the INNER JOINS work:

We have a table called *suppliers* with two fields (supplier_id and supplier_name). It contains the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have another table called *orders* with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2013/05/12
500126	10001	2013/05/13
500127	10004	2013/05/14

If we run the MySQL SELECT statement (that contains an INNER JOIN) below:

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
INNER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

supplier_id	name	order_date
10000	IBM	2013/05/12
10001	Hewlett Packard	2013/05/13

The rows for *Microsoft* and *NVIDIA* from the supplier table would be omitted, since the supplier_id's 10002 and 10003 do not exist in both tables. The row for 500127 (order_id) from the orders table would be omitted, since the supplier_id 10004 does not exist in the suppliers table.

Old Syntax

As a final note, it is worth mentioning that the MySQL INNER JOIN example above could be rewritten using the older implicit syntax as follows (but we still recommend using the INNER JOIN keyword syntax):

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers, orders
WHERE suppliers.supplier_id = orders.supplier_id;
```

LEFT OUTER JOIN

Another type of join is called a MySQL LEFT OUTER JOIN. This type of join returns all rows from the LEFT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

Syntax

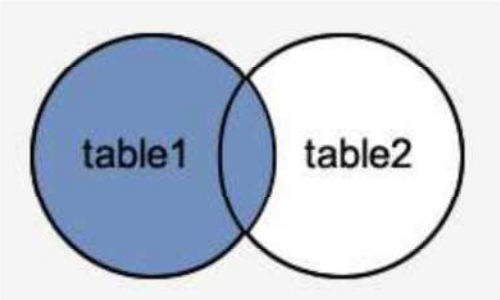
The syntax for the LEFT OUTER JOIN in MySQL is:

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

In some databases, the LEFT OUTER JOIN keywords are replaced with LEFT JOIN.

Visual Illustration

In this visual diagram, the MySQL LEFT OUTER JOIN returns the shaded area:



The MySQL LEFT OUTER JOIN would return the all records from *table1* and only those records from *table2* that intersect with *table1*.

Example

Here is an example of a MySQL LEFT OUTER JOIN:

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
LEFT JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

This LEFT OUTER JOIN example would return all rows from the suppliers table and only those rows from the orders table where the joined fields are equal.

If a supplier_id value in the suppliers table does not exist in the orders table, all fields in the orders table will display as <null> in the result set.

Let's look at some data to explain how LEFT OUTER JOINS work:

We have a table called *suppliers* with two fields (supplier_id and supplier_name). It contains the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

We have a second table called *orders* with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2013/05/12
500126	10001	2013/05/13

If we run the SELECT statement (that contains a LEFT OUTER JOIN) below:

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
LEFT JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

supplier_id	supplier_name	order_date
10000	IBM	2013/05/12
10001	Hewlett Packard	2013/05/13
10002	Microsoft	<null>

10003	NVIDIA	<null>

The rows for *Microsoft* and *NVIDIA* would be included because a LEFT OUTER JOIN was used. However, you will notice that the order_date field for those records contains a <null> value.

RIGHT OUTER JOIN

Another type of join is called a MySQL RIGHT OUTER JOIN. This type of join returns all rows from the RIGHT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal (join condition is met).

Syntax

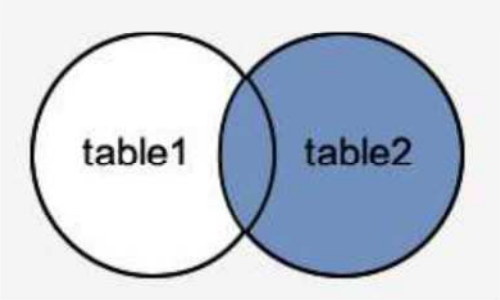
The syntax for the RIGHT OUTER JOIN in MySQL is:

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

In some databases, the RIGHT OUTER JOIN keywords are replaced with RIGHT JOIN.

Visual Illustration

In this visual diagram, the MySQL RIGHT OUTER JOIN returns the shaded area:



The MySQL RIGHT OUTER JOIN would return the all records from *table2* and only those records from *table1* that intersect with *table2*.

Example

Here is an example of a MySQL RIGHT OUTER JOIN:

```
SELECT orders.order_id, orders.order_date, suppliers.supplier_name
FROM suppliers
RIGHT JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

This RIGHT OUTER JOIN example would return all rows from the orders table and only those rows from the suppliers table where the joined fields are equal.

If a supplier_id value in the orders table does not exist in the suppliers table, all fields in the suppliers table will display as <null> in the result set.

Let's look at some data to explain how RIGHT OUTER JOINS work:

We have a table called *suppliers* with two fields (supplier_id and supplier_name). It contains the following data:

supplier_id	supplier_name
10000	Apple
10001	Google

We have a second table called *orders* with three fields (order_id, supplier_id, and order_date). It contains the following data:

order_id	supplier_id	order_date
500125	10000	2013/08/12
500126	10001	2013/08/13
500127	10002	2013/08/14

If we run the SELECT statement (that contains a RIGHT OUTER JOIN) below:

```
SELECT orders.order_id, orders.order_date,
suppliers.supplier_name
FROM suppliers
RIGHT JOIN orders
ON suppliers.supplier_id = orders.supplier_id;
```

Our result set would look like this:

order_id	order_date	supplier_name
500125	2013/08/12	Apple
500126	2013/08/13	Google
500127	2013/08/14	<null>

The row for *500127* (order_id) would be included because a RIGHT OUTER JOIN was used. However, you will notice that the supplier_name field for that record contains a <null> value.

Loading and Dumping A Database

We can load a database or otherwise execute SQL commands from a file. We simply put the commands or database into a file—let's call it mystuff.sql—and load it in with this command:

$ mysql people < mystuff.sql

We can also dump out a database into a file with this command:

$ mysqldump people > entiredb.sql

For fun, try the mysqldump command with the people database (a gentle reminder: the password is LampIsCool):

$ mysqldump -uapache -p people

Enter password:

Notice that this outputs all the SQL needed to create the table and insert all the current records. For more information, see man mysqldump.

Unit - IV

PHP Introduction

PHP (recursive acronym for PHP: Hypertext Preprocessor) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML. PHP is a server side scripting language that is also an interpreted language. It is basically used for developing software applications that have dynamic content and require constant interactions with databases. It is an open source language making it one of the most sought out platforms for developers and companies.

Features and Advantages

- The performance of the scripts written in php is much better than those written in other languages like JSP and ASP.
- Being open source, we can develop and maintain our applications without incurring any cost.
- PHP is a platform independent language, i.e., it can run on any OS without any problem
- PHP code can also be embedded into HTML tags and script.

Commenting your code

A comment is a piece of code that is not executed by the compiler rather it is there just for the human reader to read. Comments in general are used to explain the code better and improve the readability of the code.

There are multiple types of comments

Single Line Comments

Single-line comments are used to provide short explanations or points related to the code.

<? //This is a comment ?>

Multi-line comments

Multiline comments are used to provide more detailed explanations about the code whenever necessary. This system of commenting is same as the system of commenting in C

<? /*This is a multiline comment */ ?>

The Assignment Operator

You have seen the assignment operator in use each time a variable was declared in an example; the assignment operator consists of the single character: =. The assignment operator takes the value of the right-side operand and assigns it to the left-side operand:

```
$name = "jimbo";
```

Operations and Expressions

The variable `$name` now contains the string `"jimbo"`. This construct is also an expression. Although it might seem at first glance that the assignment operator simply changes the variable `$name` without producing a value, in fact, a statement that uses the assignment operator always resolves to a copy of the value of the right operand. Thus

```
echo $name = "jimbo";
```

prints the string `"jimbo"` to the browser while it also assigns the value `"jimbo"` to the `$name` variable.

Arithmetic Operators

The arithmetic operators do exactly what you would expect—they perform arithmetic operations. Table 5.2 lists these operators along with examples of their usage and results.

Table 5.2. Arithmetic Operators

Operator	Name	Example	Sample Result
+	Addition	10+3	13
-	Subtraction	10-3	7
/	Division	10/3	3.3333333333333
*	Multiplication	10*3	30
%	Modulus	10%3	1

The addition operator adds the right-side operand to the left-side operand. The subtraction operator subtracts the right-side operand from the left-side operand. The division operator divides the left-side operand by the right-side operand. The multiplication operator multiplies the left-side operand by the right-side operand. The modulus operator returns the remainder of the left-side operand divided by the right-side operand.

The Concatenation Operator

The concatenation operator is represented by a single period (`.`). Treating both operands as strings, this operator appends the right-side operand to the left-side operand. So

```
"hello"." world"
```

returns

```
"hello world"
```

Note that the resulting space between the words occurs because there is a leading space in the second operand (" world" instead of "world"). The concatenation operator literally smashes together two strings without adding any padding. So, if you tried to concatenate two strings without leading or trailing spaces, such as

```
"hello"."world"
```

you would get this as your result:

```
"helloworld"
```

Regardless of the data types of the operands used with the concatenation operator, they are treated as strings, and the result will always be of the string type. You will encounter concatenation frequently throughout this book when the results of an expression of some kind must be combined with a string, as in

```
$cm = 212;

echo "the width is ".($cm/100)." meters";
```

Combined Assignment Operators

Although there is only one true assignment operator, PHP provides a number of combination operators that transform the left-side operand and return a result, while also modifying the original value of the variable. As a rule, operators use operands but do not change their original values, but combined assignment operators break this rule. A combined assignment operator consists of a standard operator symbol followed by an equal sign. Combination assignment operators save you the trouble of using two operators in two different steps within your script. For example, if you have a variable with a value of 4, and you want to increase this value to 4 more, you might see:

```
$x = 4;

$x = $x + 4; // $x now equals 8
```

However, you can also use a combination assignment operator (+=) to add and return the new value, as shown here:

```
$x = 4;

$x += 4; // $x now equals 8
```

Each arithmetic operator, as well as the concatenation operator, also has a corresponding combination assignment operator. Table 5.3 lists these new operators and shows an example of their usage.

Table 5.3. Some Combined Assignment Operators

Operator	Example	Equivalent To
+=	$x += 5	$x = $x + 5
-=	$x -= 5	$x = $x - 5
/=	$x /= 5	$x = $x / 5
*=	$x *= 5	$x = $x * 5
%=	$x %= 5	$x = $x % 5
.=	$x .= " test"	$x = $x." test"

Each of the examples in Table 5.3 transforms the value of `$x` using the value of the right-side operand. Subsequent uses of `$x` will refer to the new value. For example

```
$x = 4;

$x += 4; // $x now equals 8

$x += 4; // $x now equals 12

$x -= 3; // $x now equals 9
```

These operators will be used throughout the scripts in the book. You will frequently see the combined concatenation assignment operator when you begin to create dynamic text; looping through a script and adding content to a string, such as dynamically building the HTML code to represent a table, is a prime example of the use of a combined assignment operator.

Automatically Incrementing and Decrementing an Integer Variable

When coding in PHP, you will often find it necessary to increment or decrement a variable that is an integer type. You will usually need to do this when you are counting the iterations of a loop. You have already learned two ways of doing this—either by incrementing the value of `$x` using the addition operator

```
$x = $x + 1; // $x is incremented by 1
```

or by using a combined assignment operator

```
$x += 1; // $x is incremented by 1
```

In both cases, the new value is assigned to `$x`. Because expressions of this kind are common, PHP provides some special operators that allow you to add or subtract the integer constant `1` from an integer variable, assigning the result to the variable itself. These are known as the *post-increment* and *post-decrement* operators. The post-increment operator consists of two plus symbols appended to a variable name:

```
$x++; // $x is incremented by 1
```

This expression increments the value represented by the variable `$x` by one. Using two minus symbols in the same way will decrement the variable:

```
$x--; // $x is decremented by 1
```

If you use the post-increment or post-decrement operators in conjunction with a conditional operator, the operand will be modified only after the first operation has finished:

```
$x = 3;

$y = $x++ + 3;
```

In this instance, `$y` first becomes `6` (the result of 3 + 3) and then `$x` is incremented.
In some circumstances, you might want to increment or decrement a variable in a test expression before the test is carried out. PHP provides the pre-increment and pre-decrement operators for this purpose. These operators behave in the same way as the post-increment and post-decrement operators, but they are written with the plus or minus symbols preceding the variable:

```
++$x; // $x is incremented by 1

--$x; // $x is decremented by 1
```

If these operators are used as part of a test expression, incrementing occurs before the test is carried out. For example, in the next fragment, `$x` is incremented before it is tested against `4`.

```
$x = 3;

++$x < 4; // false
```

The test expression returns `false` because `4` is not smaller than `4`.

Comparison Operators

Comparison operators perform comparative tests using their operands and return the Boolean value `true` if the test is successful or `false` if the test fails. This type of expression is useful when using control structures in your scripts, such as `if` and `while` statements. This book covers `if` and `while`statements in Chapter 6, "Flow Control Functions in PHP."

For example, to test whether the value contained in `$x` is smaller than `5`, you can use the less-than operator as part of your expression:

```
$x < 5
```

If `$x` contains the value `3`, this expression will have the value `true`. If `$x` contains `7`, the expression resolves to `false`.

Table 5.4 lists the comparison operators.

Table 5.4. Comparison Operators

Operator	Name	Returns True If...	Example ($x Is 4)	Result
==	Equivalence	Left is equivalent to right	$x == 5	false
!=	Non-equivalence	Left is not equivalent to right	$x != 5	true
===	Identical	Left is equivalent to right and they are the same type	$x === 4	true

Operator	Name	Returns True If...	Example ($x Is 4)	Result
	Non-equivalence	Left is equivalent to right but they are not the same type	`$x === "4"`	`false`
>	Greater than	Left is greater than right	`$x > 4`	`false`
>=	Greater than or equal to	Left is greater than or equal to right	`$x >= 4`	`true`
<	Less than	Left is less than right	`$x < 4`	`false`
<=	Less than or equal to	Left is less than or equal to right	`$x <= 4`	`true`

These operators are most commonly used with integers or doubles, although the equivalence operator is also used to compare strings. Be very sure to understand the difference between the == and = operators. The == operator tests equivalence, whereas the = operator assigns value. Also, remember that === tests equivalence with regards to both value and type.

Creating Complex Test Expressions with the Logical Operators

Logical operators test combinations of Boolean values. For example, the `or` operator, which is indicated by two pipe characters (`||`) or simply the word `or`, returns the Boolean value `true` if either the left or the right operand is true:

```
true || false
```

This expression returns `true`.

The `and` operator, which is indicated by two ampersand characters (`&&`) or simply the word `and`, returns the Boolean value `true` only if both the left and right operands are true:

```
true && false
```

This expression returns the Boolean value `false`. It's unlikely that you will use a logical operator to test Boolean constants because it makes more sense to test two or more expressions that resolve to a Boolean. For example

```
($x > 2) && ($x < 15)
```

returns the Boolean value `true` if `$x` contains a value that is greater than `2` and smaller than `15`. Parentheses are used when comparing expressions to make the code easier to read and to indicate the precedence of expression evaluation. Table 5.5 lists the logical operators.

Table 5.5. Logical Operators

Operator	Name	Returns True If...	Example	Result
`||`	Or	Left or right is true	`true || false`	`true`
`or`	Or	Left or right is true	`true or false`	`true`
`xor`	Xor	Left or right is true but not both	`true xor true`	`false`
`&&`	And	Left and right are true	`true && false`	`false`
`and`	And	Left and right are true	`true and false`	`false`
`!`	Not	The single operand is not true	`! true`	`false`

Expressions

Expressions are the most important building blocks of PHP. In PHP, almost anything you write is an expression. The simplest yet most accurate way to define an expression is "anything that has a value".

The most basic forms of expressions are constants and variables. When you type "$a = 5", you're assigning '5' into$a. '5', obviously, has the value 5, or in other words '5' is an expression with the value of 5 (in this case, '5' is an integer constant).

After this assignment, you'd expect $a's value to be 5 as well, so if you wrote $b = $a, you'd expect it to behave just as if you wrote $b = 5. In other words, $a is an expression with the value of 5 as well. If everything works right, this is exactly what will happen.

Slightly more complex examples for expressions are functions. For instance, consider the following function:

```
<?php
function foo ()
{
    return 5;
}
?>
```

Assuming you're familiar with the concept of functions (if you're not, take a look at the chapter about functions), you'd assume that typing $c = foo() is essentially just like writing $c = 5, and you're right. Functions are expressions with the value of their return value. Since foo() returns 5, the value of the expression 'foo()' is 5. Usually functions don't just return a static value but compute something.

PHP Variables

Variables in a program are used to store some values or data that can be used later in a program. The variables are also like containers that store character values, numeric values, memory addresses, and strings. PHP has its own way of declaring and storing variables. There are few rules, that needs to be followed and facts that need to be kept in mind while dealing with variables in PHP:

- Any variables declared in PHP must begin with a dollar sign (**$**), followed by the variable name.
- A variable can have long descriptive names (like $factorial, $even_nos) or short names (like $n or $f or $x)
- A variable name can only contain alphanumeric characters and underscores (i.e., 'a-z', 'A-Z', '0-9 and '_') in their name. Even it cannot start with a number.
- A constant is used as a variable for a simple value that cannot be changed. It is also case-sensitive.
- Assignment of variables is done with the assignment operator, "equal to (=)". The variable names are on the left of equal and the expression or values are to the right of the assignment operator '='.
- One must keep in mind that variable names in PHP names must start with a letter or underscore and no numbers.
- PHP is a loosely typed language, and we do not require to declare the data types of variables, rather PHP assumes it automatically by analyzing the values. The same happens while conversion. No variables are declared before they are used. It automatically converts types from one type to another whenever required.
- PHP variables are case-sensitive, i.e., $sum and $SUM are treated differently.

Example

```
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";      // outputs "Bob, Joe"

$4site = 'not yet';     // invalid; starts with a number
$_4site = 'not yet';    // valid; starts with an underscore
$täyte = 'mansikka';    // valid; 'ä' is (Extended) ASCII 228.
?>
```

Control –statement

Control statements are a basic component of all modern-day programming languages like PHP, Java, Python, Go, Ruby, NodeJS. These control statements allow software developers and architects to decide how the software or program they develop will behave under different conditions. For instance, on an e-commerce platform, the developers would want the same

system to behave differently for different user roles like buyers as well as sellers. Such kind of distinguished behaviors can only be achieved with control statements.

Different Control Statement in PHP

Like all other languages, PHP provides a few control statements enabling developers to develop different logic to execute in different conditions. PHP core includes the control statements:

- if
- if.else
- if.else..if
- Switch statement

Let us look at each of these control statements with details and understand their implementation through examples.

1. The IF Statement in PHP

The IF statement in PHP is the most simplified control statement of the language. The IF condition works on a Boolean value which is evaluated based on a certain condition and it is used to execute certain lines of code only if a condition is met or is true. The condition provided to the IF statement is first to evaluate, depending on the evaluation a False or True value is generated and on basis of it the code if the IF condition block is either executed or skipped in the program flow.

Let's walk through the syntax of if statement to understand it better:

```
if (my_condition) {
code to execute if the condition supplied is true;
}
```

As shown above, the if statement requires a condition in the () round brackets which should be evaluated. In the curly braces, {} we supply the code spec which should be executed.

et's see an example below:

Code:

```
<?php
$a=20;
$b=10;
if($a>$b)
{
echo "A is greater than B";
}
?>
```

Output:

```
A is greater than B
```

2. The IF-ELSE Statement in PHP

As mentioned before, the IF statement provides a very basic program control. The IF-ELSE statement adds further complexity to the IF statement by defining 2 blocks of code; one to be executed when the condition stands true and other when the condition evaluates to false. Naturally, since the condition can either evaluate to true or false either the code block in IF

would execute or the code block under ELSE would execute. Under no circumstances, both blocks will execute parallel.

Let's review it's syntax below:

```
if (condition top evaluate) {
code to executed if the condition supplied is true;
}
else
{
code to execute if the condition supplied is false;
}
```

Let's understand it's implementation with an example:

Code:

```
<?php
$a=20;
$b=10;
if($a>$b)
{
echo "A is greater than B";
}
else
{
echo "B is greater than B";
}
?>
```

Output:

```
A is greater than B
```

Now, when the values of $a and $b are switched in the above code as shown below:

Code:

```
<?php
$a=10;
$b=20;
if($a>$b)
{
echo "A is greater than B";
}
else
{
echo "B is greater than A";
```

```
}
?>
```

Output:

```
B is greater than A
```

3. The IF – ELSE–IF Statement in PHP

Sometimes a need may arise to use multiple if conditions together, in such cases we can use a combination of multiple if-else statements. We can combine several if-else statements to work together as long as our requirement is met.

The syntax of if-else if combination statement would be as follows:

```
If (condition1)
{
Code to execute;
}
elseIf(condition2)
{
Code to execute if condition 2 is met;
}
Else
{
Code to execute if condition1 and condition2 are not met;
}
```

Let's understand this better with an example:

Code:

```
<?php
$t = date("H"); //collecting the date from server
echo "The time  is " . $t;
echo ", and we will show the following message:";
echo "\n";
if ($t < "10") {  //condition 1
echo "Hello! I hope you have a good morning!";
} elseif ($t < "20") { //condition 2
echo "Hello! I hope you have a good day!";
} else {
echo "Hello! I hope you have a good night!";
}
?>
```

Output:

```
The time  is 09, and we will show the following message:
Hello! I hope you have a good morning!
```

4. Switch Statement in PHP

The Switch Statement or generally known as a switch loop is a very efficient way to work with multiple conditions simultaneously. It allows us to achieve the same functionality as if-else if and else does but can be achieved with lesser lines of code.

Syntax:

```
switch (condition) {
case value1:
code to be executed if n=label1;
break;
case value 2:
code to be executed if n=label2;
break;
case value 3:
code to be executed if n=label3;
break;
...
default:
code to be executed if n is different from all labels;
}
```

The Switch statement first evaluates the "condition" and then executes the code in the block with corresponding label value. If none of the label values match the condition, it executes the code in the default block.

Now let's understand the flow of the switch statement with the below practical example:

Code:

```
<?php
$myfavsport = "cricket";
switch ($myfavsport) {
case "cricket":
echo "Your favorite sport is cricket!";
break;
case " football":
echo "Your favorite sport is football!";
break;
case "throwball":
echo "Your favorite sport is throwball!";
break;
default:
```

```
echo "Your favorite sport is neither cricket, football or
throwball!";
}
?>
```

Output:

```
Your favorite sport is cricket!
```

Arrays

Arrays in PHP, provides you with an outline for the creation of arrays in PHP. An array is a collection of similar datatypes. An array stores multiple values in a single variable. Why there is a need for an array when the work of storing a value can be done by variable also? The answer is because to store values of limited data like count of numbers 5 is possible, but when count increases to say 100 or 200 we need to store 100 values in 100 variables which is a bit difficult thus we store it in an array. This is why arrays are used.

How to Create Arrays in PHP?

Syntax:

```
variablename = array();
```

OR

```
variablename[i] = value;,
```

Where variable name is the name of the variable i is the key or the index value is the element value.

Example to Create an Array

```
$colors = array("Red","Green","Blue");
```

To calculate the length of array we use the count keyword.

```
$length = count($colors); // output is 3
```

Each value in the array is termed as an element of the array. The array index begins with 0. And the index of the last element in an array is the total length of the array minus 1.

In the given example above, the index of Red is 0, Green is 1 and that of Blue is 2. It really becomes easier to access the array with the help of the index or a key. To get the value at each index of an array we loop through the given array. To loop the array we use a foreach loop or for a loop.

Types of Arrays in PHP

There are three types of arrays that you can create. These are:

- **Indexed array** — An array with a numeric key.
- **Associative array** — An array where each key has its own specific value.
- **Multidimensional array** — An array containing one or more arrays within itself.

Indexed Arrays
An indexed or numeric array stores each array element with a numeric index. The following examples shows two ways of creating an indexed array, the easiest way is:

Example
```
<?php
// Define an indexed array
$colors = array("Red", "Green", "Blue");
?>
```
Note: In an indexed or numeric array, the indexes are automatically assigned and start with 0, and the values can be any data type.
This is equivalent to the following example, in which indexes are assigned manually:

Example
```
<?php
$colors[0] = "Red";
$colors[1] = "Green";
$colors[2] = "Blue";
?>
```

Associative Arrays
In an associative array, the keys assigned to values can be arbitrary and user defined strings. In the following example the array uses keys instead of index numbers:

Example
```
<?php
// Define an associative array
$ages = array("Peter"=>22, "Clark"=>32, "John"=>28);
?>
```
The following example is equivalent to the previous example, but shows a different way of creating associative arrays:

Example
```
<?php
$ages["Peter"] = "22";
$ages["Clark"] = "32";
$ages["John"] = "28";
?>
```

Multidimensional Arrays
The multidimensional array is an array in which each element can also be an array and each element in the sub-array can be an array or further contain array within itself and so on. An example of a multidimensional array will look something like this:

Example
```
<?php
```

```
// Define a multidimensional array
$contacts = array(
    array(
        "name" => "Peter Parker",
        "email" => "peterparker@mail.com",
    ),
    array(
        "name" => "Clark Kent",
        "email" => "clarkkent@mail.com",
    ),
    array(
        "name" => "Harry Potter",
        "email" => "harrypotter@mail.com",
    )
);
// Access nested value
echo "Peter Parker's Email-id is: " . $contacts[0]["email"];
?>
```

Functions

IN PHP, many functions are used such as built-in functions and user-defined functions. Each and every function has its own functionality and properties. A function is a set of statements written in the program that can be used multiple times in the code anywhere needed. A function call is required to execute the statements written inside the function. It is a piece of code that takes one or more inputs as a parameter and processes it and returns a value. Programmers simply have to create a function and then call that function in the program wherever required.

Types of Functions in PHP

In PHP, mainly two functions are used by the programmers. They are:

1. User-Defined

These functions are used when the developer or programmer has to execute their own logic of code. These functions are defined using the keyword function and inside the function, a set of statements will be written to execute it when a function call occurs. The function call can be made by just simply calling the function like functionname(), and the function will get executed.

2. Built-in

These functions provide us with built-in library functions. PHP provides these functions in the installation package itself which makes this language more powerful and useful. To use the properties of the function we just need to call the function wherever required to fetch the desired result.

There are many built-in functions used in PHP such as Date, Numeric, String, etc.

String Functions: These functions have a predefined functionality in PHP to work with strings. PHP has various string functions such as strpos(), strncmp(), strrev(), strlen(),

Date Function: These functions are predefined functionality in PHP where the format is a UNIX date and time which is a human-readable format.

Numeric Functions: These functions have their own predefined logic provided by PHP which is used for numeric operations. It will return the result either in Boolean form or in the numeric form. Some of the numeric functions include is_number(), number_format(), round() ,etc.

How Functions are Used in PHP?

As we discussed earlier, in PHP we have two functions i.e. built-in and user-defined. Let's understand more about these functions:

Example #1

For String Functions

Code:

```
<!DOCTYPE html>
<html>
<body>
<?php
print_r(str_split("Hi This is a test sample"));
?>
</body>
</html>
```

Output:

Array ([0] => H [1] => i [2] => [3] => T [4] => h [5] => i [6] => s [7] => [8] => i [9] => s [10] => [11] => a [12] => [13] => t [14] => e [15] => s [16] => t [17] => [18] => s [19] => a [20] => m [21] => p [22] => l [23] => e)

The explanation for the above program: In the above example, the string that we passed inside the function str_split(), splits the string on to a single character and produces the output.

Example #2

Code :

```
<!DOCTYPE html>
<html>
<body>
<?php
echo strcmp("Hi this is test","Hi this is test");
?>
<p>If this function returns 0, the two strings are same.</p>
</body>
</html>
```

Output:

0

If this function returns 0, the two strings are same .

The explanation for the above program: In the above example, the function strcmp () will compare the strings and if the strings are the same it will return zero and if the strings are not equal then it will return some other number.