Module - V

PHP & POSTGRESQL (psql)

[1] Pogresql =

* powerful, open source, general purpose & obj-relational DB system
* Features -

1) Data integrity -
* Supports various constraints like unique, primary key, foreign key, etc.
* psql adheres (follow closely) to ACID proprties (Atomicity, consistency, Isolation, Durability) to guarantee that transactions are processed reliably.

2) scalability -
* means the ability of the DB system to handle a growing amount of wrk.
* Spports horizontal & vertical scalability to add more servers & increasing the resources of the existing ~~sys~~ srvr.

3) security -
* Supports secure & encrypted SSL (secure Socket layer) connections & provides

various authentication methods.

4) Transaction sprt -
provide full sprt for ACID (pro) & ensures effective transaction isolation using the multiversion concurrency control Method (MVCC)

5) cross-platform -
runs on Unix os, Linux, macOS as well as windows systms.

6) Availability

7) Indepndnce - developed by intrnational community, not a company.

[2] Datatypes :- (11)

I Numeric types :
- Smallint - 2 bytes
- big int - 8 "
- integr - 4 "
- decimal - variable
- numeric - "
- Smallserial - 2 bytes
- bigserial - 8 "
- Serial - 4 "

II Monetary Types :
money type stores a currncy amount with a fixed fractional precision.
money - 8 bytes

III Character type :
- varchar(n) → variable length
- char(n) → fixed length
- text → variable unlimited length

IV Binary type :
bytea type used to store binary str.

V Date/time type

VI Boolean type - 1 byte - T/F

VII Enumerated type :
enum types comprises a static, ordered set of values.
eg → Create Type week AS ENUM ('Mon', 'Tue', ...);

VIII Geometric type :
repsnts 2D spatial obj.
point - 16 bytes
line - 32 "
box - 32 "
circle - 24 "

IX Network type:
- cidr – 7 / 19 bytes
- inet – 7 / 19 bytes
- macaddr – 6 bytes

X Bit Str type:
used to store bit masks., they are either 0/1.
2 bit SQL types → bit(n)
→ bit varying(n).
n → +ve int.

XI Array type:
eg → create table savings (
name text,
saving integer[],
scheme text[][]
);
insert into savings values
('AK', '{20000, 14600, 23500, 13250}',
'{{"FD", "MF"}, {"FD", "prpty"}}');

Select name from savings where saving[2] > saving[4];

[3] commands =

* \d → describe all the tables in a DB.
shx → \d tablename;
* \c dbname → connect to DB

I CREATE DATABASE:
create a PSql DB in shell prompt
shx → Create database dbname;
eg → postgres=# create database testdb;

II CREATE TABLE:
shx → create table table_name (
col1 datatype,
col2 datatype
.. ..
primary key (one/more cols)
);
eg → create table company (
id int primary key NOT NULL,
name text NOT NULL,
Age int NOT NULL,
Address char(50),
salary real
);

III INSERT QUERY:
shx → ① insert into tablename (col, col2..)
values (val1, val2...);
② insert into tablename values
(val1, val2 ....);

Exists → to list down all the records.

* \? → to get full list of psql cmnds.
* \l → to list all DB in cmt psql

IV SELECT QUERY: used to fetch the data frm a DB table, which returns data in the form of rslt table.
These tables → result-sets.
Slx → Select * from tablename;
eg → Select id, name, salary from Company;

a) WHERE clause:
used to specify a condin while the data frm single table / joining with multiple tables.
Slx → Select col, col2.. from tblename WHERE [search condin];
eg → ① Select * from company WHERE Age >= 25 AND salary >= 30000;
② select * from company WHERE Age IS NOT NULL;
③ select name from company WHERE name like 'ja%'; (starting with 'ja')
④ select name, age from company WHERE Age IN (25, 27) (age value is either 25 or 27)
⑤ ..... WHERE AGE NOT IN (25, 27) (age value neither 25 nor 27)
⑥ Select age from company WHERE EXISTS (select age from company WHERE salary >2000);

b) ORDER BY clause:
used to sort the data in asc/dsnd ording, based on 1/more cols.
Slx → select col list from tablename WHERE condin [ORDERBY col1, col2..] [ASC / DESC];
eg → Select name, age from Company ORDERBY name DESC;

c) GROUP BY clause:
to grp togthr those rows in a table that hve identical data.
Slx → Select col list from tbname WHERE [condin] GROUP BY col1, col2.. ORDERBY col1, col2..
eg → Select name, count(name), sum(salary) from company GROUP BY name;

d) HAVING clause:
- to pick out particular rows where the ()'s rslt meets some condin.
- This clause must follow GROUP BY in a query & must also proceede the ORDER BY clause if used.
- Slx → Select col1, col2 from table1, table2 WHERE [condin]

* \help → to get full list of SQL commds.
* \help create table;
* \dt → to list all tables in crnt DB.

GROUP BY col1, col2 ..
HAVING [condition]
ORDER BY col1, col2 .. ;

eg → Select name, Count(name) from company GROUP BY name HAVING count(name)<2;
(display record for which the name count is <2)

④ DISTINCT keyword:

used in conjunction with Select (5) to eliminate all the duplicate records & fetching only unique records.

Syn → select Distinct col1, col2 ..
from tbname WHERE [condition]

eg → select name from company ORDER BY name ASC;
(produce more than 1 same entry).
Select Distinct name from company;
(donot have duplicate entry)

V SELECT INTO (5):

* Allows you to create a new table & inserts data returned by a query.
* Unlike Select (5), the select into (8) does not return data to the client.

decimal(10,2)
(decimal point-ல முன் 10 to decimal-ல 8 and 2 digit store)

* Syn → select collist
INTO [temporary | TEMP | UNLOGGED] [TABLE] new tablename
from tablename
WHERE condition
* eg → create table prod (
id serial primary key,
name varchar(255),
price decimal(10,2)
);
insert into prod(name, price) values ('prod A', 10), ('prod B', 20);
SELECT name, price INTO product from prod WHERE price>200;
[This will create a new table 'product' & insert all of the products from prod table that cost > 200]

VI CREATE TABLE AS (5):

* Creates a table & fills it with data computed by a Select cmnd.
* Syn → create table tblename [(Colname [,...])] AS query.
* eg → create table comp AS
Select * from comp where salary <20000;

VII) DELETE query:

* used to dlt the existing records frm a table.
* You can use 'where' clause with delete query to dlt the selected rows.
* S/x → DELETE from tablename
  WHERE condin
* eg → ① delete from company where id=2;
  ② delete from company;

VII) UPDATE query:

* used to modify the existing record in a table.
* S/x → Update tblename
  SET Col=val1, col2=val2 ...
  WHERE condin
* eg → ① update company set salary=15000 where id=3;
  ② update company set add='Texas', salary=20000;
  [To modify all address & salary columns values in company table].

[4] Integration of Php-PSQL =

I Establishing Connection - Pg_connect():

* pg_connect used to open a db connection.
* It takes a connection str as its argumnts & returns a db connection handle.
* S/x → pg_connect (string $connection_string [,int $connect_type])
  where $connection_string is the connecting str
* they can be empty to use all default paramtr.
* contain 1 / more paramtr settings (keyword=value)
* $connect_type is optional & can be either PGSQL_CONNECT_FORCE_NEW or PGSQL_CONNECT_ASYNC.
* keyword options used in $connection_string —
a) dbname : db to connect to
b) user : username to use whn connecting
c) password : password for specified user.
d) host : name of srvr to connect to
e) hostaddr : IP address of srvr to connect to
f) port : TCP/IP port to connect to on the srvr.

port = 5432 (default)

* eg → <body>
<?php
$db_handle = pg_connect ("host = localhost . dbname = testdb user = postgres password = postgres");
(conn" ok) → if ($db_handle) {
echo "connected";
→ else
echo "error on connecting";
pg_close ($db_handle);
?>
</body>

* ()s for retrieving info about a conn" –
a) pg_dbname () – returns cnt dbname
s/x → pg_dbname (dbname);
b) pg_host ()
c) pg_options ()
d) pg_port ()
e) pg_tty () – returns TTY name.

* persistent connection – pg_pconnect():
pg_pconnect() creates a persistnt connection that can be reused by othr scripts & pg_connect () creates a new connection each time the script runs, & this connection is not reused by othr scripts.

s/x → resource pg_connect (string $connection_str [, int $connection_type])

pg_connection_status:

* returns the status of the specified connection (conn" ok check)
* s/x → int pg_connection_status (resource $connection);
* returns 2 possible values
→ PGSQL_CONNECTION_OK → 1
→ PGSQL_CONNECTION_BAD → 0
* eg → $db_handle = pg_connect ("....");
$stat = pg_connection_status ($db_handle);
if ($stat == PGSQL_CONNECTION_OK)
echo 'connected';
else
die('prbm occured in code');
pg_close ($db_handle);

die() used to display a msg & exit the script.

III pg_last_error() :

* returns the most recently occurring error msg from the connection.
* Syn → Str pg_last_error ([resource $connection]);
* eg → $db_handle = pg_connect("....") OR die("connection prbm");
  $res = pg_query("DROP TABLE nonexistenttable");
  echo pg_last_error();
  → ERROR: table 'nonExistTable' does not exist
* Other error handling ()s —
  - pg_result_error()
  - " _error_field()
  - " _status() —
    returns info regarding the status of rslt.

IV pg_close() :

* PHP will automatically close any open connection at the end of script's execution.
* Syn → pg_close ($db_handle);
* pg_close() is an immediate explicit cmnd.

V Building queries :

* queries can be built using PHP Str ()s.
* eg → $name = strtolower ($name);
  $query = "SELECT * FROM CUSTOMER.
  WHERE NAME = '$name' ";
  (msg/o $name convrt to lowrcase then return)

VI Execute queries : (2 methods)

1) pg_query() = responsible for Sndng query Str to psql Servr & returning the rslt set.
   Syn → resource pg_query ([resource $connection], Str $query);
   eg → $db_handle = ...
   $query = " select *.... = $name ";
   $result = pg_query ($db_handle, $query);

2) pg_execute() =
   pg_prepare() :
* creates a prepared statemnt for later Execution with pg_execute / pg_send_execute();
* This failure allows cmnds that will be used repeatedly to be parsed & planned

jst once, rathr than each time they are executed.

* $rs → pg_prepare ($connection, $statmntname, $query);

$statmntname → name to give the prepared statmt
$query → parameterized SQL (s). If any parmtrs are used, they are refferred to as $1, $2, etc~

* eg → pg_prepare ($dbhandle, "my_query", 'select * from shop where name = $1');

* pg_execute():

- Snds a reqst to execute a prepared statmnt with gvn paramtrs & wait for the rslt.
- $rs → resource pg_execute ([resource $conn"], str $statmntname, array $parmtrs);
  $parmts → an array of parmtrs values to substitute for $1, $2...
- eg →

① $dbconn = pg_connect("...");
// prepare a query
$query = "INSERT INTO product (pid, name, price, description) values ($1, $2, $3, $4)";
$result = pg_prepare ($dbconn, "my_query", $query);
// execute the prepared query

$result = pg_execute ($dbconn, "my_query", array ("c34", "toothpaste", "25", "best in market"));

② $dbhandle = pg_connect("...");
$result = pg_prepare ($dbhandle, "my_query", 'select * from shop where name = $1');
$result = pg_execute ($dbhandle, "my_query", array ("Ansar"));
$result = pg_execute ($dbhandle, "my_query", array ("fazil"));

[5] working with result sets: (5)

* The return value of a call to pg_query() or pg_execute() is a pointer to a resltset.
* resltset stores the rslt of the query as returned by the db.
* To retrieve the rsltset following () are used —

1) pg_fetch_row(): used to fetch a row frm a psql db rsltset.
- whn you execute a query using pg_query() or similar (), you get a result resource.

you can then use pg_fetch_row()
to fetch individual rows from this rslt set.
- It returns an array, indexed frm 0 upwards with each value represented as a str.
- Str → array pg_fetch_row (resource $result [, int $row])
- eg → $conn = pg_connect ("dbname=publisher");

Checks whether the conn is successful →
if (!conn) {
    echo "An error occured";
    exit;
}

(exit() should be given if, biz the fun after it, biz the exit() terminates the pgm & pgm will not continue executing after it() is executed)

$result = pg_query ($conn, "select author, email from authors");

Checks whether the query is successful or not →
if (!$result) {
    echo "An error occured";
    exit;
}

while ($row = pg_fetch_row ($result)) {
    echo "Author: $row[0]
    Email: $row[1]";
    echo "
";
}

author → row[0]
email → row[1]

→ Author: Anjar Email: ~~
Author: fazil Email: ~~

2) pg_fetch_array():

* returns an array that corresponds to the fetched row.
* rslts can be stored with numeric indices (field nos) & also with associative indices (field name).
* Str → pg_fetch_array (resource $result [, int $row [, int $mode = PGSQL_BOTH]]).
- $mode controls how the returned array is indexed.
- mode is a constnt & can take following value - PGSQL_ASSOC, PGSQL_NUM & PGSQL_BOTH.
a) PG_NUM - returns an array with numerical indices like pg_fetch_row
b) PG_ASSOC - returns associative indx with key represented by fieldname & value by field contnts
c) PG_BOTH - Default, numeric indx & associative indx are possible.

* eg → (1) $arr = pg_fetch_array ($result, 0, PGSQL_NUM);
echo $arr[0];
echo $arr[1];

② $arr = pg_fetch_array ($result, NULL, PGSQL_ASSOC);
echo $arr["author"];
echo $arr["email"];

③ $arr = pg_fetch_array ($result); Both
echo $arr["author"];
echo $arr[1];

3) pg_fetch_assoc(): returns an associative array that corresponds to fetched row.
eg → like pg_fetch_row() → while → pg_fetch_assoc → $row = [email]

4) pg_fetch_object(): returns an obj with (pro) that corresponds to fetched row's field names.
Identical to pg_fetch_array() except that an obj is returned rathr than an array.
eg → while($row = pg_fetch_object ($result)){
echo $row → author
$row → email;
echo "
";
}

5) pg_fetch_all():

Selected → when we select → select * from ~
Affected → update / dlt → 2 rows affected

* returns an array that contains all records in the rslt resource.
* Shx → array pg_fetch_all (resource $result [, int $mode = PGSQL_ASSOC]).
* eg → ~~$arr = pg_fetch_all ($result);~~ ~~print_r ($arr);~~
* eg → $arr = pg_fetch_all ($result);
foreach ($arr as $row){
echo $row[author]
$row[email];
}.

[6] Rows Selected & Rows Affected =

* Fns to return the nos of rows selected or affected by a query execution.

1) pg_num_rows():

* returns the nos of rows in psql rslt resource.
* on error, it returns -1.
* eg → $result = pg_query ($conn, "SELECT *");
$rows = pg_num_rows ($result);
echo $rows;
(Select * from company → query → ~~~ → rows ~~~ ~~~ ())

pg_get_result() → retrive the rslt of query that was sent asynchronously using pg_send_query().

2) pg_affected_rows():
* returns the no of tuples affected by ins, update & dlt queries.
* Slx → int pg_affected_rows (resource $result);
* If no tuple is affected return 0.
* eg → $result = pg_query ($conn, "insert into authors values ('orwell', 2002)");
$res = pg_affected_rows ($result);
echo $res row(s) affected;
→ 1 row affected.

3) pg_num_fields() (colmns)
* returns the no of colmns in psql result resource.
* int pg_num_fields (resource $result);
* on error, return 0.

[7] Insert, update & dlting data =
* Data can be inserted using INSERT query & executed using pg_query(), pg_execute(), & pg_send_query().
* Data can be updated using UPDATE & dlted using DELETE query.

pg_send_query() → snds query to psql server bt doesn't wait for the rslt. it allows pgm to continue executing other tasks while waiting for the query to be processed on the srvr.

1 Inserting data:

```
<body>
<form action="test.php" method="GET">
<table>
<tr>
<td> Eid </td>
<td><input type="text" name="id"></td>
</tr>
<tr>
<td> name </td>
<td> <input type="text" name="name"></td>
</tr>
</table>
</form> </body>
```

This is insert.html.

```
<body>
<?php
$db_handle = pg_connect ("....");
if (!$db_handle)
    die ("\n error occurred");
$query = "insert into company (id, name)
    values ('$_GET[id]', '$_GET[name]')";
pg_send_query ($db_handle, $query);
$result = pg_get_result ($db_handle);
```

update & } executed using pg_query(),
delete } pg_send_query() & pg_execute().

```
if(pg_result_error($result)!=NULL){
    echo "error in inserting";
else{
    echo "data inserted";
} ?>
<input type="button" name=click
  value="Add more" onclick="parent.location
    ='insert.html'" >.
</body>
```

This is test.php.

II update data :

```
<?php
$db_handle = pg_connect("...");
if(!db_handle)
    die("....");
$query = "update company set name='Amal',
        age=30, address='kollam'
        where id=1011";
$result = pg_query($db_handle, $query);
if(pg_affected_rows($result)==0)
    echo "no record found";
else
    echo "data updated";
```

III deleting data :

```
<?php
$db_handle = pg_connect("...");
if(!db_handle)
    die("...");
$query = "delete from company
        where id=1011";
$result = pg_query($db_handle, $query);
if(pg_affected_rows($result)==0)
    echo "no record found";
else
    echo "data deleted";
?>
```

[8] Freeing Result sets =

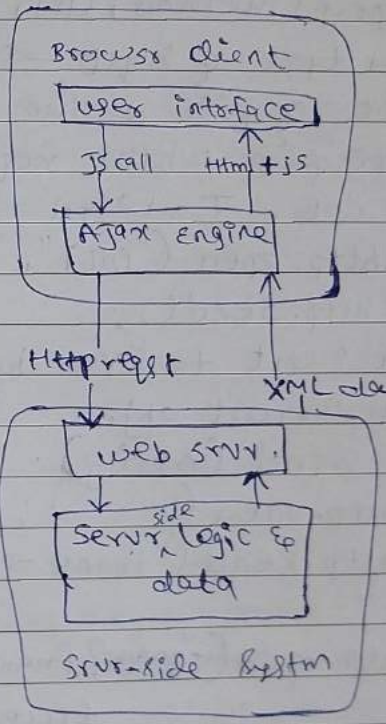
* pg_free_result() frees the mly & data associated with the specified psql query rslt resource.
* only called if mly consumption during script execution is a prbm otherwise all rslt mly will be automatically freed whn the scripts ends.
* Syn → boolean pg_free_result(resource $result);

AJAX made by google.

[9] Introduction to AJAX =

* AJAX → Asynchronous JS and XML.
* It is a technique for creating fast & dynamic web pgs with the hlp of XML, html, CSS & js.
* uses Html for content, CSS for presntation, along with doc obj model & js for dynamic content display.
* Allows web pgs to be updated asynchronsly by exchanging small amounts of data with srvr behind the scenes. means, that is possible to update parts of a webpg, without reloading the whole pg.
* used to communicate with the srvr without refreshing the web pg & thus Tse user experience.
* Google Suggest is using AJAX to create a very dynamic web intrface.
* AJAX is not a prgming lang, bt a way to use existing standards like js & XML.
* js hlps in making a request to web srvr.
* This request is asynchronous & snd from a web browsr to a webSrvr with the hlp of an obj → xmlhttpRequest then Srvr returns requested data using XML
* eg → facebook, youtube, google map.

→ working of AJAX:



[10] Sending request & retrieving response =

Steps—

1) Creating an instance of XMLHttpRequest obj

to send an HttpRequest from a client to a server.

s/x → variable = new XMLHttpRequest();

2) Sending the request to server using open() method of XMLHttpRequest() obj

s/x → open(method, url, async)

method → type of req — GET / POST

url → loc of file on the srvr

Async → specifies whthr req is handled or not, T → asyn & F → synchronous

eg → xmlhttp.open("GET", "file.php", true);
xmlhttp.send();

3) Sending a req to srvr by using send() of XMLHttpRequest obj.

s/x → send(string).

eg → xmlhttp.open(....);
xmlhttp.send("fname=Run&lname=AK");
(or)
xmlhttp.open("GET", "greet.php?fname=Run&lname=AK", true);

[11] XMLHttpRequest obj:

* JS uses a spl XML obj built into the browser to make HTTP requests to the srvr & receive data in response.
* XML-req used either asyn / syn.
If XML-req used syn, then aftr snding the req to the srvr, the user has to wait till the response is received frm srvr.
* eg of syn →

```
xmlRequest = new XMLHttpRequest();
xmlRequest.open("GET", 'add.xml', false);   // request creation
xmlRequest.send(null);                      // and request
xmlRequest = xmlRequest.responseXML;
```

(response in the form of XML is stored in a variable)

* eg of asyn →

```
xmlRequest = new XMLHttpRequest();
"  .onreadystatechange = asyncHandlr;
"  .open("GET", "add.xml", true);
"  .send(null);
function asyncHandlr() {
    if (xmlRequest.readyState == 4)
        objXML = xmlRequest.responseXML;
}
```

* (pro) of XMLHttpRequest —
1) onreadystatechange : It is an evnt handler proprty of XMLHttpReq obj in js. It specifies a fn() to be called

whnevr the readystate property of xml_req changes.

2) readyState: This property holds the status of XMLHttpRequest.

5 possible values for readyState —

value	state	Description
0	UNSENT	client has been created, open() not called yet.
1	OPENED	open() has been called.
2	HEADERS_RECEIVED	send() has been called, & headers & status are available
3	LOADING	downloading; responseText holds partial data.
4	DONE	The op^r is complete

eg→
```
const xhr = new XMLHttpRequest();
console.log("UNSENT", xhr.readyState);
xhr.open("GET", "xmc.php", true);
console.log("OPENED", xhr.readyState);
xhr.onprogress = () => {
    console.log("LOADING", xhr.readyState);
};
xhr.onload = () => {
    console.log("DONE", xhr.readyState);
};
```

→ UNSENT 0
OPENED 1
LOADING 3
DONE 4.

* onprogress & onload are event handler (pro) of XMLHttpRequest obj.

onprogress → It is called whnevr xml_req changes while the request is in progress.

onload → It is called whn the request completes successfully, (i.e) whn the readstate is 4. & status is 200 (ok).

3) responseText:

* returns the response in the form of str.
* If readstate value is 0, 1, 2 → responseText contains an empty str.
* 3 → it contains incomplete response received by the client.
* 4 → it contains complete response received by the client.

4) responseXML:

* returns the response in XML format.
* This property returns an XML doc obj.

* xmc response is returned only why readystate value is 4.

5) status :

* reprints the http status code
* It is only available whn the readystate value is either 3 / 4.

Http Status code	Http Status txt.
200	text ok
204	text no content
302	text found
404	text not found
503	service unavailable

6) statusText :

* used to retrieve the http status of a request.
* provide textual description of the value supplied by the status property
* This property is only available why readystate value is 3 / 4.
* Error value access exception arises.

7) abort() → cancels the crnt request made by the xmlHttp request obj.

eg → xhr.abort();

8) open()
9) send()

[12] performing an ajax GET request :

* used to get / retrieve some kind of info from the server that does not require any change in DB.

eg → fetching search rslt based on a term, fetching user details based on their id / name.

* eg:→

```
<script>
    function displayFullName(){
        req = new XMLHttpRequest();
        req.open("GET","greet.php?
        fname=Ram&lname=Rahcam", true);
        req.onreadystatechange=function(){
            if(this.readyState===4 &&
            this.status===200)
                doc.getElementByName
                ("result").innerHTML=
                    this.responseText;
        };
        req.send();
    }
</script>
```

```
<?php
 if (isset ($_GET["fname"]) && isset ($_GET
                ["lname"])) {
  $fname = htmlspecialchars ($_GET["fname"]);
  $lname =  "          ($_GET["lname"]);
 // creating fullname by joining fname & lname
  $fullname = $fname . " " . $lname;

  echo "Hello, $fullname !";
} else {
     echo " error ";
}
?>
```

→ Hello Ram Raheem.

* isset() → checks if a var is set & not null
* htmlspecialchars() → converts special char to their corresponding html entities.
  eg → & → &
  " → "

[13] performing Ajax post request :

* used to submit a form data to code &rvr.
* eg →

```
<script>
  function postcmnd () {
    req = new ...
    req.open ("post", "confirmation.php");
    req.onready state change = function () {
        - - - - - -
    },
  };
  // retriving the form data.
  var myform = do.get ... ("myform");
  var formData = new FormData (myform);
  req.send (formdata);
}
```

```
<?php
if ($_SERVER["REQUEST METHOD"] == "post"){
   $name = htmlspecialchars (trim ($_post[
                              "name"]));
    $cmnt = - - - - - -
 // check if form fields are empty
if(!empty ($name) && !empty ($cmnt)) {
     echo "Hi $name ";
     echo "Here's yr cmnd $cmnd";
}
else {
     echo " fill yr form ";
}
```

- $_SEVER["REQEUST_METHOD"] ==> POST →
  Checks if the form was submitted using POST method.
- trim() removes any white space from the beginning & end of the str.