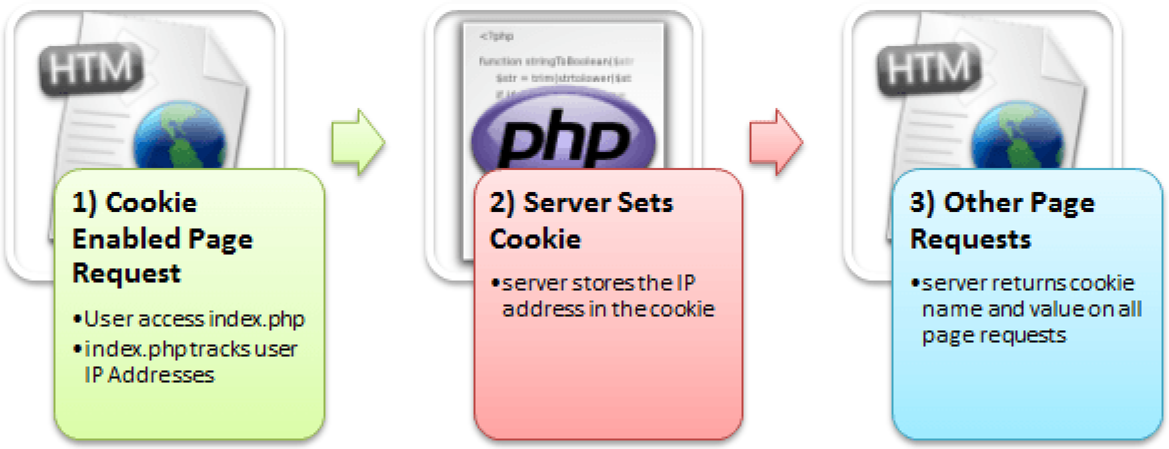
Cookie in php

- A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.
- Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.
- cookies are also used for storing user related data.
- A cookie is a small file with the maximum size of 4KB that the web server stores on the client computer.
- Once a cookie has been set, all page requests that follow return the cookie name and value.
- A cookie can only be read from the domain that it has been issued from.
- Cookies are small files saved on the user's computer
- Cookies can only be read from the issuing domain
- Cookies can have an expiry time, if it is not set, then the cookie expires when the browser is closed
- **The diagram shown below illustrates how cookies work.**


HTM
1) Cookie Enabled Page Request
•User access index.php
•index.php tracks user IP Addresses
php
2) Server Sets Cookie
•server stores the IP address in the cookie
HTM
3) Other Page Requests
•server returns cookie name and value on all page requests

- Here,

  1) A user requests for a page that stores cookies

  2) The server sets the cookie on the user's computer

  3) Other page requests from the user will return the cookie name and value

Uses of Cookies

- cookies allow us to track the state of the application using small files stored on the user's computer.

- Personalizing the user
- Tracking the pages visited by a user

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

setcookie(*name, value, expire, path, domain, secure*);

- **Name** − This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.
- **Value** − This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** − This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.
- **Path** − This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain** − This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security** − This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

isset() function to find out if the cookie is set:

- If you want to **destroy a cookie** before its expiry time, then you set the expiry time to a time that has already passed.
- Modify a Cookie Value
- To modify a cookie, just set (again) the cookie using the setcookie() function:

Modify a Cookie Value

- To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<?php
$name = "user";
$value = "Alex Porter";
setcookie($name, $value, time() + (86400 * 30), "/"); //3600-one hour

if(!isset($_COOKIE[$cookie_name]))

{
    echo "Cookie named '" . $name . "' is not set!";
}

else

 {
    echo "Cookie '" . $name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$value];
}

if(count($_COOKIE) > 0) {
   echo "Cookies are enabled.";
} else {
   echo "Cookies are disabled.";
}

// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
```

Session in PHP

- session refers to a frame of communication between two medium.
- A PHP session is used to store data on a server rather than the computer of the user.
- Session identifiers or SID is a unique number which is used to identify every user in a session based environment. The SID is used to link the user with his information on the server like posts, emails etc.
- The session IDs are randomly generated by the PHP engine .
- The session data is stored on the server therefore it doesn't have to be sent with every browser request.
- The session_start() function needs to be called at the beginning of the page, before any output is generated by the script in the browser.

- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- So; Session variables hold information about one single user, and are available to all pages in one application.
- Sessions are like global variables stored on the server
- Each session is given a unique identification id that is used to track the variables for a user.
- Both cookies and sessions must be started before any HTML tags have been sent to the browser.

Uses of Sessions

- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL
- You are developing an application such as a shopping cart that has to temporary store information with a capacity larger than 4KB.

Starting a PHP Session:

- The first step is to start up a session.
- After a session is started, session variables can be created to store information. The PHP **session_start()** function is used to begin a new session.It als creates a new session ID for the user.
- Below is the PHP code to start a new session:

```
<?php
  session_start();
  ?>
```

Storing Session Data:

- Session data in key-value pairs using the **$_SESSION[]** superglobal array.The stored data can be accessed during lifetime of a session.
- Below is the PHP code to store a session with two session variables Rollnumber and Name:

```
<?php
session_start();
$_SESSION["Rollnumber"] = "11";
$_SESSION["Name"] = "Ajay";
```

```
?>
```

Accessing Session Data:

- Data stored in sessions can be easily accessed by firstly calling **session_start()** and then by passing the corresponding key to the **$_SESSION** associative array.
- The PHP code to access a session data with two session variables Rollnumber and Name is shown below:

```
<?php
session_start();
echo 'The Name of the student is :' . $_SESSION["Name"] . '<br>';
echo 'The Roll number of the student is :' . $_SESSION["Rollnumber"] . '<br>';
?>
```

- **Output:**

```
The Name of the student is :Ajay
The Roll number of the student is :11
```

Destroying Certain Session Data:

- To delete only a certain session data,the unset feature can be used with the corresponding session variable in the **$_SESSION** associative array.
- The PHP code to unset only the “Rollnumber” session variable from the associative session array:

```
<?php
session_start();
  if(isset($_SESSION["Name"])){
    unset($_SESSION["Rollnumber"]);
}
?>
```

Destroying Complete Session:

- The **session_destroy()** function is used to completely destroy a session. The session_destroy() function does not require any argument.

```
<?php
session_start();
session_destroy();
 ?>
```

Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";

// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);

session_unset();

// destroy the session
session_destroy();


?>

</body>
</html>
```

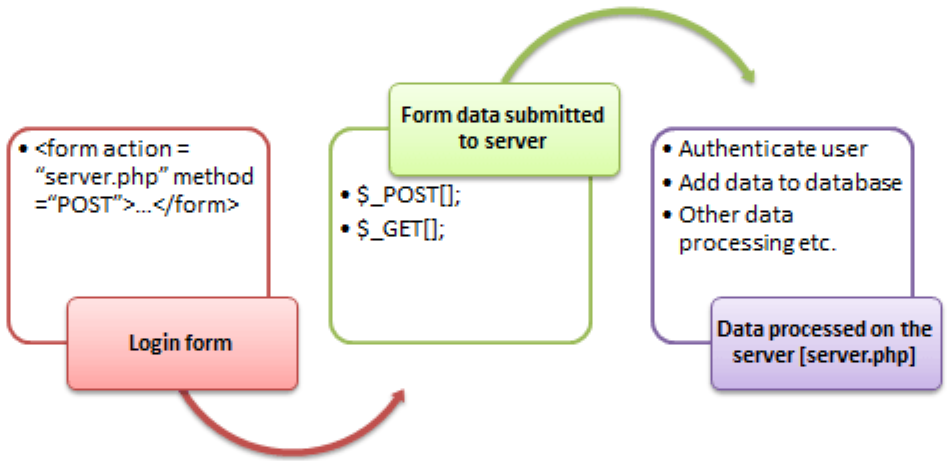
PHP and web forms

What is Form?

When you login into a website or into your mail box, you are interacting with a form.

Forms are used to get input from the user and submit it to the web server for processing.

The diagram below illustrates the form handling process.



• <form action = "server.php" method ="POST">...</form>
Login form
Form data submitted to server
• $_POST[];
• $_GET[];
• Authenticate user
• Add data to database
• Other data processing etc.
Data processed on the server [server.php]

A form is an HTML tag that contains graphical user interface items such as input box, check boxes radio buttons etc.

The form is defined using the <form>...</form> tags and GUI items are defined using form elements such as input.

When and why we are using forms?

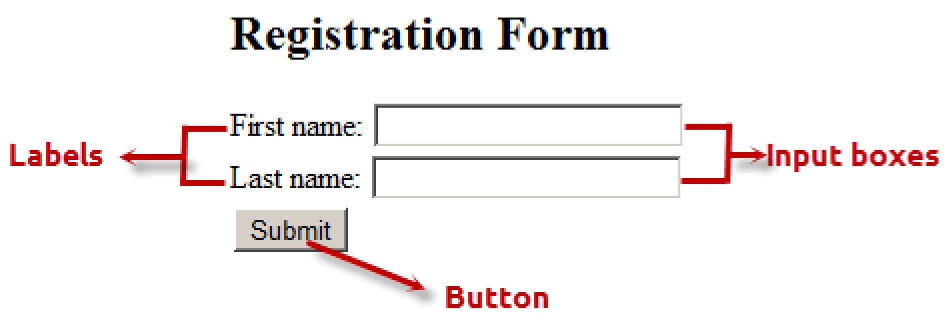
- Forms come in handy when developing flexible and dynamic applications that accept user input.
- Forms can be used to edit already existing data from the database

Create a form

We will use HTML tags to create a form. Below is the minimal list of things you need to create a form.

- Opening and closing form tags <form>…</form>
- Form submission type POST or GET
- Submission URL that will process the submitted data
- Input fields such as input boxes, text areas, buttons,checkboxes etc.

Viewing the above code in a web browser displays the following form.



The code below creates a simple registration form

Registration.html

```
<html>

<body>

   <h2>Registration Form</h2>

   <form action="registration.php" method="POST">

         First name:  <input type="text" name="firstname"> <br>

          Last name:  <input type="text" name="lastname">

         Age:  <input type="text" name="age">


      <input type="submit" value="Submit">

   </form>

</body>

</html>
```

HERE,

- <form…>…</form> are the opening and closing form tags
- action="registration_form.php" method="POST"> specifies the destination URL and the submission type.
- First/Last name: are labels for the input boxes
- <input type=”text”…> are input box tags

- <br> is the new line tag
- <input type="hidden" name="form_submitted" value="1"/> is a hidden value that is used to check whether the form has been submitted or not
- <input type="submit" value="Submit"> is the button that when clicked submits the form to the server for processing

Submitting the form data to the server

The action attribute of the form specifies the submission URL that processes the data. The method attribute specifies the submission type.

PHP POST method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP POST method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method is ideal when you do not want to display the form post values in the URL.
- A good example of using post method is when submitting login details to the server.

It has the following syntax.

```
<?php
 $_POST['variable_name'];
?>
```

HERE,

- "$_POST[…]" is the PHP array
- "'variable_name'" is the URL variable name.

PHP GET method

- This is the built in PHP super global array variable that is used to get values submitted via HTTP GET method.
- The array variable can be accessed from any script in the program; it has a global scope.
- This method displays the form values in the URL.
- It's ideal for search engine forms as it allows the users to book mark the results.

It has the following syntax.

```
<?php

$_GET['variable_name'];

?>
```

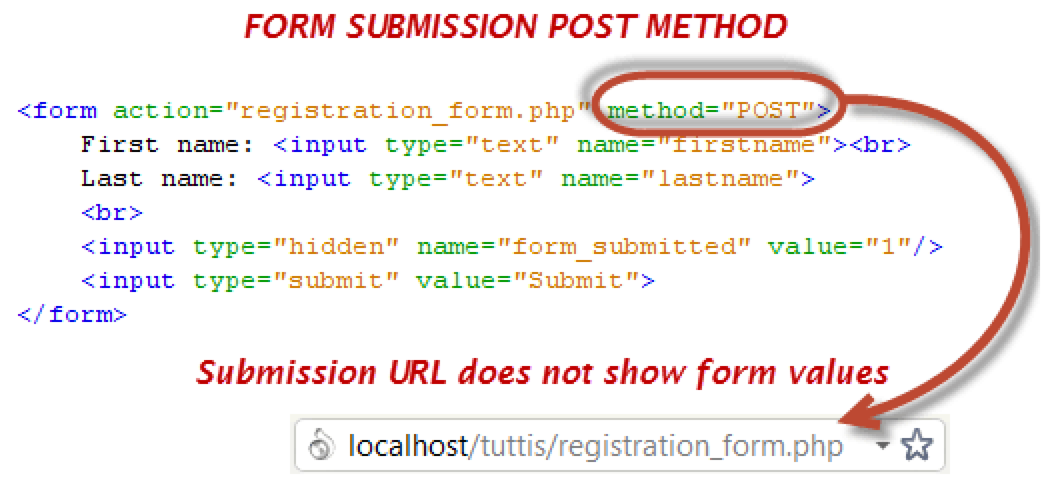
HERE,

- "$_GET[…]" is the PHP array
- "'variable_name'" is the URL variable name.

GET vs POST Methods

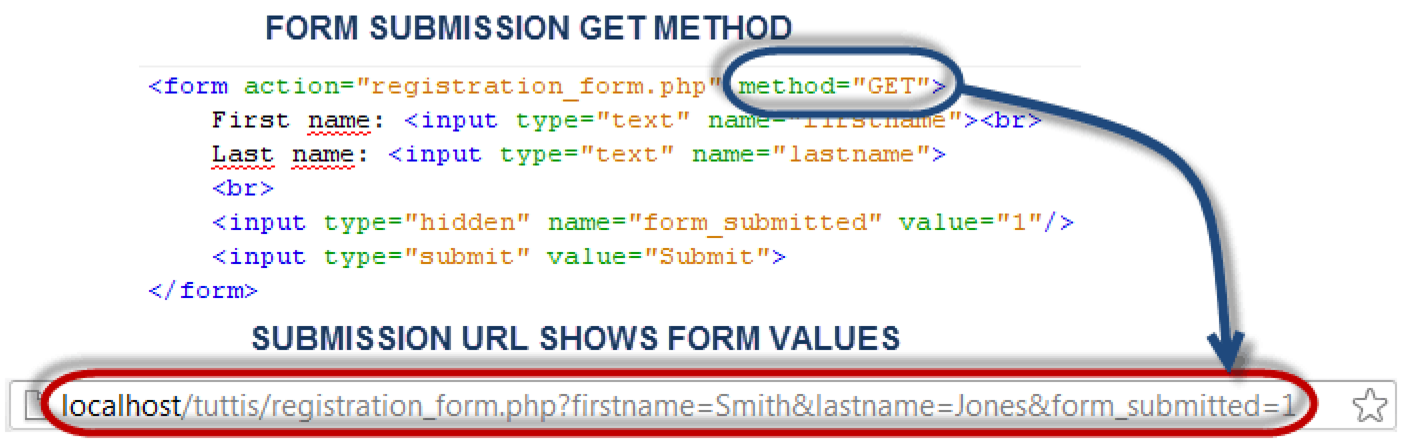
POST	**GET**
Values not visible in the URL	Values visible in the URL
Has not limitation of the length of the values since they are submitted via the body of HTTP	Has limitation on the length of the values usually 255 characters. This is because the values are displayed in the URL. Note the upper limit of the characters is dependent on the browser.
Has lower performance compared to Php_GET method due to time spent encapsulation the Php_POST values in the HTTP body	Has high performance compared to POST method dues to the simple nature of appending the values in the URL.
Supports many different data types such as string, numeric, binary etc.	Supports only string data types because the values are displayed in the URL
Results cannot be book marked	Results can be book marked due to the visibility of the values in the URL

The below diagram shows the difference between get and post


FORM SUBMISSION POST METHOD
Submission URL does not show form values

```
<form action="registration_form.php" method="POST">
    First name: <input type="text" name="firstname"><br>
    Last name: <input type="text" name="lastname">
    <br>
    <input type="hidden" name="form_submitted" value="1"/>
    <input type="submit" value="Submit">
</form>
```

localhost/tuttis/registration_form.php



```
<form action="registration_form.php" method="GET">
    First name: <input type="text" name="firstname"><br>
    Last name: <input type="text" name="lastname">
    <br>
    <input type="hidden" name="form_submitted" value="1"/>
    <input type="submit" value="Submit">
</form>
```

localhost/tuttis/registration_form.php?firstname=Smith&lastname=Jones&form_submitted=1

Processing the registration form data

The registration form submits data to itself as specified in the action attribute of the form.

When a form has been submitted, the values are populated in the $_POST super global array.

We will use the PHP isset function to check if the form values have been filled in the $_POST array and process the data.

We will modify the registration form to include the PHP code that processes the data. Below is the modified code

<u>Registration.php</u>

```
<?php

$fname=$_POST['firstname'];
```

```
$lname=$_POST['lastname'];

$a=$_POST['age'];

echo "First name is  : " .$fname.  "<br>";

echo "Last name is  : " .$lname.  "<br>";

echo "Age is  : " .$a.  "<br>";

?>
```

PHP MySQL Database

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Opening Database Connection

PHP provides **mysql_connect** function to open a database connection. This function takes five parameters and returns a MySQL link identifier on success, or FALSE on failure.

Syntax

```
connection mysql_connect(server,user,passwd,new_link,client_flag);
```

Sr.No	Parameter & Description
1	**server** Optional − The host name running database server. If not specified then default value is **localhost:3306**.
2	**user** Optional − The username accessing the database. If not specified then default is the name of the user that owns the server process.
3	**passwd** Optional − The password of the user accessing the database. If not

	specified then default is an empty password.
4	**new_link** Optional − If a second call is made to mysql_connect() with the same arguments, no new connection will be established; instead, the identifier of the already opened connection will be returned.
5	**client_flags** Optional − A combination of the following constants − • **MYSQL_CLIENT_SSL** − Use SSL encryption • **MYSQL_CLIENT_COMPRESS** − Use compression protocol • **MYSQL_CLIENT_IGNORE_SPACE** − Allow space after function names • **MYSQL_CLIENT_INTERACTIVE** − Allow interactive timeout seconds of inactivity before closing the connection

Database Queries

- A query is a question or a request.
- We can query a database for specific information and have a recordset returned.
- Look at the following query (using standard SQL):
- SELECT LastName FROM Employees
- The query above selects all the data in the "LastName" column from the "Employees" table.
- PHP Connect to MySQL
- PHP 5 and later can work with a MySQL database using:
- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**
- Prepared Statements protect from SQL injection, and are very important for web application security.
- Open a Connection to MySQL
- Before we can access data in the MySQL database, we need to be able to connect to the server:
- Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
```

```
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

Example (MySQLi Object-Oriented)

```
$conn->close();
```

PHP Create a MySQL Database

A database consists of one or more tables.

- The CREATE DATABASE statement is used to create a database in MySQL.
- The following examples create a database named "myDB":
- Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

PHP Create MySQL Tables

- The CREATE TABLE statement is used to create a table in MySQL.

- We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":
```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```
- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT
- Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
   echo "Table MyGuests created successfully";
} else {
   echo "Error creating table: " . $conn->error;
}

$conn->close();
?>
```

PHP Insert Data Into MySQL

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted
- The INSERT INTO statement is used to add new records to a MySQL table:

  ```
  INSERT INTO table_name (column1, column2, column3,...)
  VALUES (value1, value2, value3,...)
  ```

- The 'insert.php' file connects to the MySQL database server, retrieves forms fields using the PHP $_REQUEST variables and finally execute the insert query to add the records.
- **Example1:**

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
   echo "New record created successfully";
} else {
   echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

- **Example2**

```
<?php
$link = mysqli_connect("localhost", "root", "", "demo"); // Check connection if($link === false)
{
 die("ERROR: Could not connect. " . mysqli_connect_error());
} // Escape user inputs for security $first_name = mysqli_real_escape_string($link,
$_REQUEST['first_name']); $last_name = mysqli_real_escape_string($link,
$_REQUEST['last_name']); $email = mysqli_real_escape_string($link, $_REQUEST['email']);
$sql = "INSERT INTO persons (first_name, last_name, email) VALUES ('$first_name',
'$last_name', '$email')";
if(mysqli_query($link, $sql))
{ echo "Records added successfully."; }
Else
{ echo "ERROR: Could not able to execute $sql. " . mysqli_error($link);
} // Close connection mysqli_close($link); ?>
```

PHP Select Data From MySQL

- Select Data From a MySQL Database
- The SELECT statement is used to select data from one or more tables:
  SELECT column_name(s) FROM table_name
- or we can use the * character to select ALL columns from a table:
  SELECT * FROM table_name;

Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
   // output data of each row
```

```
    while($row = $result->fetch_assoc()) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
$conn->close();
?>
```

PHP Delete Data From MySQL

- The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

- The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!
- Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

PHP Update Data in MySQL

- The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

- The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!
- Example

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
   echo "Record updated successfully";
} else {
   echo "Error updating record: " . $conn->error;
}

$conn->close();
?>
```

PHP Prepared Statements

- Prepared statements are very useful against SQL injections.
- A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

The prepared statement execution consists of two stages: prepare and execute.

- **Prepare** — At the prepare stage a SQL statement template is created and sent to the database server. The server parses the statement template, performs a syntax check and query optimization, and stores it for later use.
- **Execute** — During execute the parameter values are sent to the server. The server creates a statement from the statement template and these values to execute it.

Prepared statements is very useful, particularly in situations when you execute a particular statement multiple times with different values, for example, a series of INSERT statements.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Prepared statements have three main advantages:

- A prepared statement can execute the same statement repeatedly with high efficiency
- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?,
?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();
```

```
$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

- "INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"
- In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.
- Then, have a look at the bind_param() function:
- $stmt->bind_param("sss", $firstname, $lastname, $email);
- This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.
- The argument may be one of four types:

  b — binary (such as image, PDF file, etc.)
  d — double (floating point number)
  i — integer (whole number)
  s — string (text)

Database Transaction

- MySQL transaction in PHP to ensure data integrity of the database.
- A transaction is a set of inter-dependent SQL statements that needs to execute in all-or-nothing mode.
- A transaction is successful if all SQL statements executed successfully.
- A failure of any statement will trigger the system to rollback to the original state to avoid data inconsistency.
- A classic example of the transaction is a money transfer transaction from one bank account to another.
- It requires three steps:

1. Check the balance of the transferred account to see if the amount is sufficient for the transfer.

2. If the amount is sufficient, deduct the amount from the balance of the transferred account.
3. Add the transfer amount to the balance of the receiving account.

- If an error occurs in the second step, the third step should not continue. In addition, if an error occurs in the third step, the second step must be reversed. The amounts of both bank accounts are intact in case of failure or adjusted correctly if the transaction is completed successfully.

MySQL transaction in PHP

- When you use PDO to create a connection to the database that supports the transaction, the auto-commit mode is set. It means that every query you issue is wrapped inside an implicit transaction.
- Notice that not all storage engines in MySQL support transaction e.g., MyISAM does not support the transaction, however, InnoDB does.
- To handle MySQL transaction in PHP, you use the following steps:
- Start the transaction by calling the beginTransaction() method of the PDO object.
- Place the SQL statements and the commit() method call in a try block.
- Rollback the transaction in the catch block by calling the rollBack() method of the PDO object.
- PHP MySQL transaction example
- We will create a table named accounts to demonstrate the money transfer between two bank accounts.
- First, execute the following statement to create the accounts table:

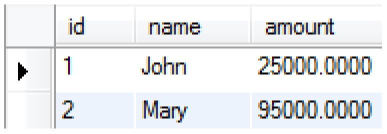
```
CREATE TABLE accounts (
    id     INT AUTO_INCREMENT PRIMARY KEY,
    name   VARCHAR (50)    NOT NULL,
    amount DECIMAL (19, 4) NOT NULL
);
```

- Second, insert two rows into the accounts table:

```
INSERT INTO accounts(name,amount)
VALUES('John',25000),
      ('Mary',95000);
```

- Third, query the accounts table:

```
SELECT *
  FROM accounts;
```



id	name	amount
1	John	25000.0000
2	Mary	95000.0000

Example

```
<?php
6./* Change database details according to your database */
$dbConnection = mysqli_connect('localhost', 'robin', 'robin123', 'company_db');
mysqli_autocommit($dbConnection, false);
$flag = true;
$query1 = "INSERT INTO `employee` (`id`, `first_name`, `last_name`, `job_title`, `salary`)
VALUES (9, 'Grace', 'Williams', 'Softwaree Engineer', $salary)";
$query2 = "INSERT INTO `telephone` (`id`, `employee_id`, `type`, `no`) VALUES (13, 9,
'mobile', '270-598712')";
$result = mysqli_query($dbConnection, $query1);
if (!$result) {
$flag = false;
echo "Error details: " . mysqli_error($dbConnection) . ".";
}
$result = mysqli_query($dbConnection, $query2);
if (!$result) {
$flag = false;
echo "Error details: " . mysqli_error($dbConnection) . ".";
}
if ($flag) {
mysqli_commit($dbConnection);
echo "All queries were executed successfully";
} else {
mysqli_rollback($dbConnection);
echo "All queries were rolled back";
}
mysqli_close($dbConnection);
 ?>
```

- When you execute mysqli_query() function, result is immediately committed to the database. Using **mysqli_autocommit()** function, you can turn off this behavior so that result won't be committed to the database permanently till you command.
- After that we simply execute the necessary statements and set the **$flag** to false if any statement fails. If there are many statements to run, consider storing the statements in an array and using a for or foreach loop.
- At the end, if flag is true (no error has occurred), we commit the results to the database permanently using **mysqli_commit()**. Else we roll back the results using **mysqli_rollback()** function.