multiple_queries.php

```
<?php
 require_once 'login.php';
 $conn = new mysqli($hn, $un, $pw, $db);
 if ($conn->connect_error) die("Unable to connect to database");

 $query = "CREATE TABLE employee ( id SMALLINT AUTO_INCREMENT,
             name VARCHAR(20), salary FLOAT,PRIMARY KEY (id))";

 // SMALLINT has range from -32768 to +32767
 $result = $conn->query($query);
 if ($result)
        echo "<br>Creation Successful";
 else
        echo "<br>Creation Failed";

 $query = "ALTER TABLE employee AUTO_INCREMENT = 101";
 $result = $conn->query($query);
 if ($result)
        echo "<br>Alteration Successful";
 else
        echo "<br>Alteration Failed";

 $query = "INSERT INTO employee VALUES (NULL, 'Ganesh Gaitonde', 14000.75),
             (NULL, 'Kuku Kumari', 28000.24), (NULL, 'Sartaj Singh', 23000.14)";
 $result = $conn->query($query);
 if ($result)
        echo "<br>Insertion Successful";
 else
        echo "<br>Insertion Failed";

 $query = "UPDATE employee SET salary='18000.24' WHERE name='kuku kumari' ";
 $result = $conn->query($query);
 if(mysqli_affected_rows($conn) > 0)
       echo "<br>Updation Successful";
 else
       echo "<br>Updation Failed";

 $query = "DELETE FROM employee WHERE id=101";
 $result = $conn->query($query);
 if(mysqli_affected_rows($conn) > 0)
       echo "<br>Deletion Successful";
 else
       echo "<br>Deletion Failed";

 $query  = "SELECT * FROM employee";
 $result = $conn->query($query);
 $rows = $result->num_rows;
 for ($j = 0 ; $j < $rows ; ++$j)
 {
  $result->data_seek($j);
       $row = $result->fetch_array(MYSQLI_NUM);
  echo "<br> ID: $row[0]   NAME: $row[1]    SALARY: $row[2] ";
 }
```

```
  $query = "DROP TABLE employee";
  $result = $conn->query($query);
  if ($result)
        die ("<br>Table Drop Successful");
  else
        die ("<br>Table Drop Failed");

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

HTML5 Forms

With HTML5, developers can draw on a number of useful enhancements to form handling to make using forms easier than ever. Although some of these features are not yet implanted across all major browsers, the following new features, however, will work on all browsers.

The autocomplete Attribute

You can apply the autocomplete attribute to either the <form> element. With autocomplete enabled, previous user inputs are recalled and automatically entered into fields as suggestions. You can also disable this feature by turning auto- complete off. Here's how to turn autocomplete on for an entire form but disable it for specific fields

```
<form action='myform.php' method='post' autocomplete='on'> <input type='text' name='username'>
        <input type='password' name='password' autocomplete='off'>
</form>
```

The autofocus Attribute

The autofocus attribute gives immediate focus to an element when a page loads. It can be applied to any <input>, <textarea>, or <button> element, like this:

```
<input type='text' name='query' autofocus='autofocus'>
```

The placeholder Attribute

The placeholder attribute lets you place into any blank input field a helpful hint to explain to users what they should enter. You use it like this:

```
<input type='text' name='name' size='50' placeholder='First & Last name'>
```

The input field will display the placeholder text as a prompt until the user starts typing, at which point the placeholder will disappear.

The required Attribute

The required attribute is used to ensure that a field has been completed before a form is submitted. You use it like this:

```
<input type='text' name='creditcard' required='required'>
```

When the browser detects attempted form submission where there's an uncompleted required input, a message is displayed, prompting the user to complete the field.

HTML5 input types

HTML5 adds a number of extra input types, which amongst other things, enable us to limit the data that users input without the need for extraneous JavaScript code. The most comforting thing about these new input types is that by default, where browsers don't support the feature, they degrade to a standard text input box.

email

type="email" – supporting browsers will expect a user input that matches the syntax of an e-mail address.

number
type="number" – supporting browsers expect a number to be entered in a number type input field. They also supply spinner controls by default, allowing users to easily click up or down to alter the value.

url
type="url" – as you might expect, the URL input type is for URL values.

tel
type="tel" is another contact information specific input type. tel is used to signify to the browser that the form expects a telephone number entered within that field.

form_submit.html

```
<!DOCTYPE html>
<html>
  <head>
              <title>Form Handling</title>
    <style type="text/css">
          * {
              font-size: 16px;
              font-family: verdana;
        padding: 5px;
          }
         </style>
  </head>

  <body>
       <form action="form_submit2.php" method="post" autocomplete="on">
      <label>Name:      <br><input      type="text"      name="name"      size="30"
placeholder="First    Name    followed    by    Last    name"    required    autofocus
/></label><br><br>
      <label>Mobile:  <br><input  type="tel"  name="tel"  size="30"  placeholder="10
Digits" autocomplete="off" required /></label><br><br>
                <label>Email:  <br><input  type="email"  name="email"  size="30"
placeholder="abc@gmail.com" required /></label><br><br>
      <label>Website:      <br><input      type="url"      name="url"      size="30"
placeholder="www.yourwebsite.com"/></label><br><br>
      <input type="submit" value="Submit" />
       </form>
</body>
</html>
<!-- While executing this program give http://localhost/php/form_submit.html
Otherwise the php program wont execute. Its source code will be shown -->
```

form_submit1.php

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Unable to connect to database");

  $name  = $_POST['name'];
  $tel     = $_POST['tel'];
  $email  = $_POST['email'];
  $url     = $_POST['url'];

  $query    = "INSERT INTO contactdetails VALUES ('$name', '$tel', '$email', '$url')";
  $result   = $conn->query($query);
  if ($result)
        die("Insertion Successful");
  else
        die("Insertion Failed");

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

form_submit2.php

```
<?php
  require_once 'login.php';
  $conn = new mysqli($hn, $un, $pw, $db);
  if ($conn->connect_error) die("Unable to connect to database");

  $name  = $conn->real_escape_string($_POST['name']);
  $tel     = $conn->real_escape_string($_POST['tel']);
  $email  = $conn->real_escape_string($_POST['email']);
  $url     = $conn->real_escape_string($_POST['url']);

  $query    = "INSERT INTO contactdetails VALUES ('$name', '$tel', '$email', '$url')";
  $result   = $conn->query($query);
  if ($result)
        die("Insertion is Successful");
  else
        die("Insertion Failed");

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

JavaScript Form Validation

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

Basic Validation – First of all, the form must be checked to make sure all the mandatory fields are filled in.

Data Format Validation – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data. Data format validation is normally done using Regular Expressions.

Regular Expressions

Regular expressions are patterns used to match character combinations in strings. A regular expression pattern is composed of two slashes and some simple characters in between such as /abc/ or a combination of simple and special characters such as /ab*c/.

Using simple patterns

Simple patterns are constructed of characters for which you want to find a direct match. For example, the pattern /abc/ matches character combinations in strings only when exactly the characters 'abc' occur together and in that order. Such a match would succeed in the strings "Hi, do you know your abc's?" and "The latest airplane designs evolved from slabcraft." In both cases the match is with the substring 'abc'. There is no match in the string 'Grab crab' because while it contains the substring 'ab c', it does not contain the exact substring 'abc'.

Matching Through Metacharacters

Every regular expression must be enclosed in slashes. Within these slashes, certain characters have special meanings; they are called metacharacters. For instance, an asterisk (*) has a meaning similar to what you have seen if you use a shell or Windows command prompt (but not quite the same). An asterisk means, “The text you’re trying to match may have any number of the preceding characters—or none at all.”
For instance, let’s say you’re looking for the name Le Guin and know that someone might spell it with or without a space. Because the text is laid out strangely (for instance, someone may have inserted extra spaces to right-justify lines), you could have to search for a line such as this:

The difficulty of classifying Le Guin's works

So you need to match LeGuin, as well as Le and Guin separated by any number of spaces. The solution is to follow a space with an asterisk:

/Le *Guin/

Suppose that you know there is always at least one space. In that case, you could use the plus sign (+), because it requires at least one of the preceding characters to be present: /Le +Guin/

Fuzzy Character Matching

The dot (.) is particularly useful, because it can match anything except a newline. Suppose that you are looking for HTML tags, which start with < and end with >. A simple way to do so is shown here:
/<.*>/
The dot matches any character, and the * expands it to match zero or more characters, so this is saying, "Match anything that lies between < and >, even if there's nothing." You will match <>, ,
, and so on. But if you don't want to match the empty case, <>, you should use + instead of *, like this:
/<.+>/
The plus sign expands the dot to match one or more characters, saying, "Match anything that lies between < and > as long as there's at least one character between them." You will match and , <h1> and </h1>, and tags with attributes, such as this: <a href="www.mozilla.org">

Character Classes

Sometimes you want to match something fuzzy, but not so broad that you want to use a dot. Fuzziness is the great strength of regular expressions: they allow you to be as precise or vague as you want.
One of the key features supporting fuzzy matching is the pair of square brackets, []. It matches a single character, like a dot, but inside the brackets you put a list of things that can match. If any of those characters appears, the text matches. For instance, if you wanted to match both the American spelling gray and the British spelling grey, you could specify the following:
/gr[ae]y/
After the gr in the text you're matching, there can be either an a or an e. But there must be only one of them: whatever you put inside the brackets matches exactly one character. The group of characters inside the brackets is called a character class.

Indicating a Range

Inside the brackets, you can use a hyphen (-) to indicate a range. One very common task is matching a single digit, which you can do with a range as follows:
/[0-9]/
Digits are such a common item in regular expressions that a single character is provided to represent them: \d. You can use it in place of the bracketed regular expression to match a digit:
/\d/

Negation

One other important feature of the square brackets is negation of a character class. You can turn the whole character class on its head by placing a caret (^) after the opening bracket. Here it means, "Match any characters except the following." So let's say you want to find instances of Yahoo that lack the following exclamation point. You could do it as follows:

/Yahoo[^!]/

Refer form_validation.html

Refer form_validation.php