**NAME:** Pulin Prabhu

**BATCH:** SE COMPS, C

**ROLL NO.:** 45

***EXPERIMENT 7***

**AIM:** Implementing Server Side Scripting (Creating and accessing Databases, and Session control) using PHP.

**THEORY:**

PHP is an open source scripting language used to create dynamic web page, modify the database,collect data,send and receive cookies.PHP files can contain text, HTML, CSS, JavaScript, and PHP code. A PHP script can be placed anywhere in the document.

**Syntax:**

A PHP script starts with **<?php** and ends with **?>**

***Designing and Creating Databases***

We can design and create a database, simply by entering “localhost/phpmyadmin” on the browser, logging in to phpmyadmin, and create your own database, from the given options. Phpmyadmin gives a set of options to design and create a database, rather than using the traditional DDL commands.

***Accessing the Database***

PHP 5 and later can work with a MySQL database using **MySQLi extension (the "i" stands for improved)**, **PDO (PHP Data Objects)**

In order to access the database, created through phpmyadmin, we need to first establish a connection to the MySQL database on the localhost server. To do so, we need to follow the following syntax:

<?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);

}

echo "Connected successfully";

?>

We establish a new connection using the syntax:

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

Then, we check whether the connection has been established. To check for an error, we check the function $conn->connect\_error, error occurs if it return true and otherwise.

Then, we enter the query to store or retrieve data from the database. The following syntax is followed:

$sql = “//Your query in the general MySQL”;

**NOTE-** While using the php variables with the query, we must know that these variables, need to be within inverted commas, otherwise, they are entered in the query without the inverted commas which is a syntax error by the terms of MySQL.

We, then send this query to our connected database and get the required output in a **$result** php variable, using the following syntax:

$result = $conn->query($sql);

Lastly, we close the current connection with the database, using this syntax:

$conn->close();

***Sessions***

When you work with an application, you open it, do some changes, and then you close it. This is much like a **Session**. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem, the web server does not know who you are or what you do, because the **HTTP address doesn't maintain state**.

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.

The session support allows you to store data between requests in the **$\_SESSION** superglobal array. When a visitor accesses your site, PHP will check automatically (if session.auto\_start is set to 1) or on your request (explicitly through session\_start()) whether a specific session id has been sent with the request. If this is the case, the **prior saved environment is recreated**.

1. **Start a PHP Session**

session\_start() creates a session or resumes the current one based on a session identifier passed via a GET or POST request, or passed via a cookie.

When session\_start() is called or when a session auto starts, PHP will call the open and read session save handlers. These will either be a built-in save handler provided by default or by PHP extensions. The read callback will retrieve any existing session data (stored in a special serialized format) and will be unserialized and used to automatically populate the $\_SESSION superglobal when the read callback returns the saved session data back to PHP session handling.

A session is started with the session\_start() function in one page. From the second page, we will access the session information we set on the first page. The session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session\_start()).

1. **Unset a PHP Session**

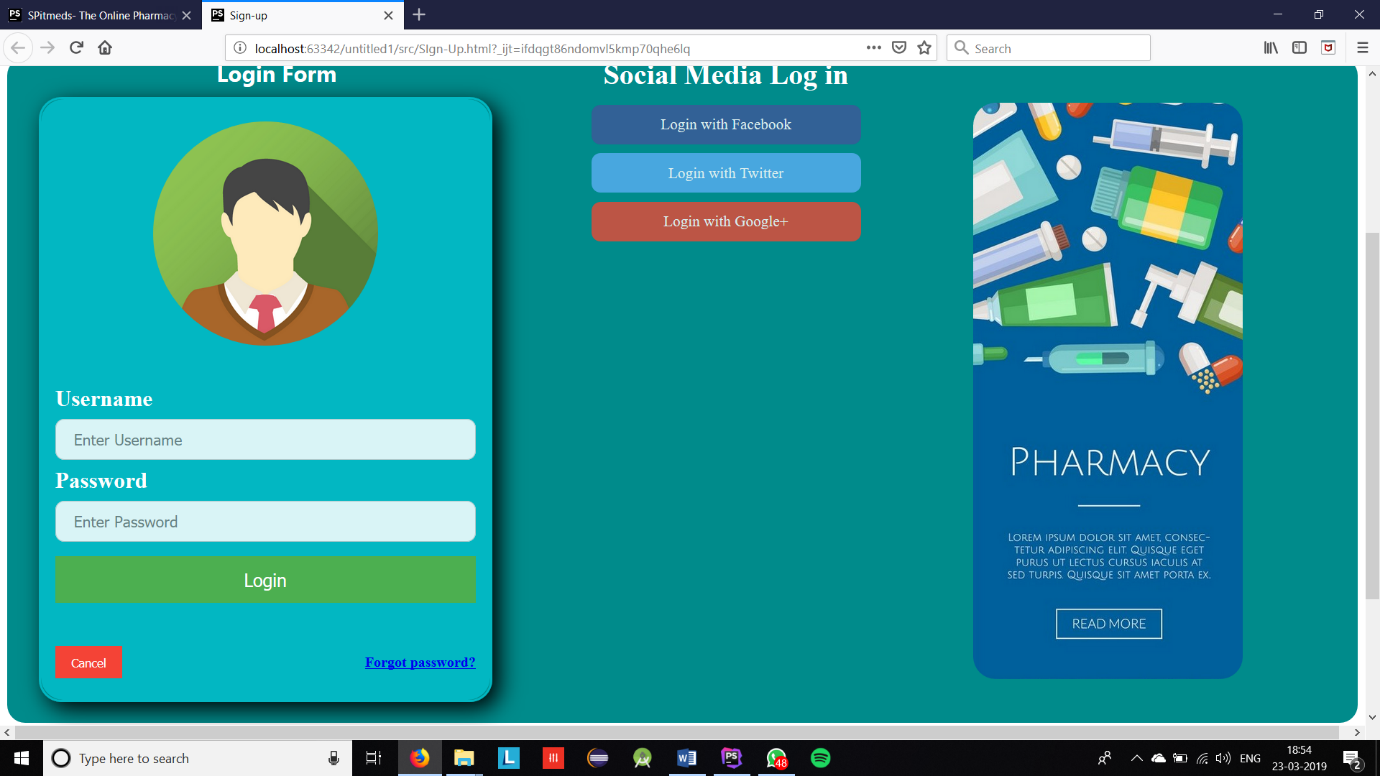
The **session\_unset()** function frees all session variables currently registered.

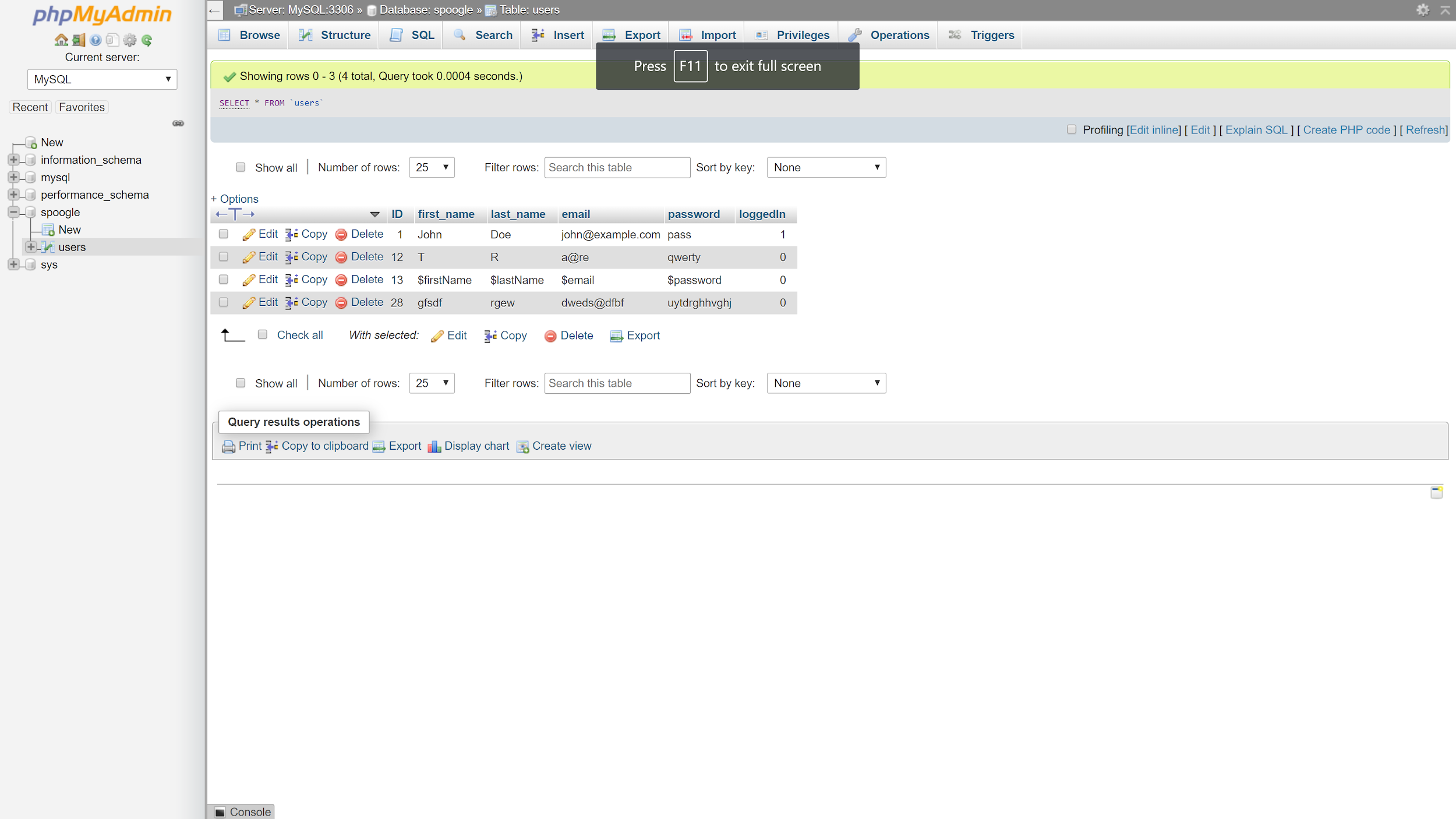
1. **Destroy a PHP Session**

**session\_destroy()** destroys all of the data associated with the current session. It does not unset any of the global variables associated with the session, or unset the session cookie. To use the session variables again, **session\_start()** has to be called.

In order to kill the session altogether, the session ID must also be unset. If a cookie is used to propagate the session ID, then the session cookie must be deleted.

**SCREENSHOTS:**



****

**CONCLUSION:** From this experiment, we have now designed our own MySQL database through phpmyadmin, and later on used PHP to send queries, store and retrieve data to and from the database. Also, used session control, to keep the session active, for a certain time, and destroy the session afterwards.