Unit 6: PHP Advance

PHP and MySQL

PHP –Database Connectivity

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

PHP –Database Connectivity

- **Both MySQLi and PDO have their advantages:**
  - PDO will work on 12 different database systems, where as MySQLi will only work with MySQL databases.
  - Both are object-oriented, but MySQLi also offers a procedural API.
  - Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

PHP –Database Connectivity

- **Both MySQLi and PDO have their advantages:**
  - PDO_DBLIB (FreeTDS / Microsoft SQL Server / Sybase)
  - PDO_FIREBIRD (Firebird/Interbase 6)
  - PDO_IBM (IBM DB2)
  - PDO_INFORMIX (IBM Informix Dynamic Server)
  - PDO_MYSQL (MySQL 3.x/4.x/5.x)
  - PDO_OCI (Oracle Call Interface)
  - PDO_ODBC (ODBC v3 (IBM DB2, unixODBC and win32 ODBC))
  - PDO_PGSQL (PostgreSQL)
  - PDO_SQLITE (SQLite 3 and SQLite 2)

PHP –Database Connectivity

- **Creating a Database:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Write a query for creating a database.
  - Step 4: Execute the query using **mysqli_query()** function.
  - Step 5: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Creating a Database Example:**

```
<?php
$con = mysqli_connect("localhost","root","");                    //Step 1
if(!$con)                                                        //Step 2
{
    die("Could not connect" . mysqli_connect_error());
}
$qry =  "CREATE DATABASE mydb";                                  //Step 3
$eqry = mysqli_query($con, $qry);                                //Step 4
if($eqry)
{
    echo "Database created";
}
else
{ echo "Error creating Database";}
mysqli_close($con);                                              //Step 5
?>
```

PHP –Database Connectivity

- **Creating a Table:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Select a database using **mysqli_select_db()** function
  - Step 4: Write a query for creating a database.
  - Step 5: Execute the query using **mysqli_query()** function.
  - Step 6: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Creating a Table Example 1:**

```
<?php
$con = mysqli_connect("localhost","root","");                    //Step 1
if(!$con)                                                        //Step 2
{    die("Could not connect" . mysqli_connect_error());    }
mysqli_select_db("mydb");                                        //Step 3
$qry = "CREATE TABLE abc(ID int NOT NULL AUTO_INCREMENT PRIMARY
    KEY(ID), name varchar(10), age int)";                        //Step 4
$eqry = mysqli_query($con, $qry);                                //Step 5
if($eqry)
{    echo "Table created";    }
else
{ echo "Error creating Table";}
mysqli_close($con);                                              //Step 6
?>
```

PHP –Database Connectivity

- **Creating a Table Example 2:**

```
<?php
$con = mysqli_connect("localhost","root","","mydb");                    //Step 1 & 3
if(!$con)                                                               //Step 2
{    die("Could not connect" . mysqli_connect_error());    }
$qry = "CREATE TABLE abc(ID int NOT NULL AUTO_INCREMENT PRIMARY
     KEY(ID), name varchar(10), age int)";                              //Step 4
$eqry = mysqli_query($con, $qry);                                       //Step 5
if($eqry)
{    echo "Table created";    }
else
{ echo "Error creating Table";}
mysqli_close($con);                                                     //Step 6
?>
```

PHP –Database Connectivity

- **Include dbconfig file in PHP page: <u>dbconfig.php</u>**

```
<?php
$dbhost = "localhost";
$dbuser = "root";
$dbpass = "";
$dbname = “myDB";
$con = mysqli_connect($dbhost,$dbuser,$dbpass,$dbname) or die('cannot
    connect to the server');
?>
```

PHP –Database Connectivity

- **Insert the data into Table:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Write a query for Insert the data into table.
  - Step 5: Execute the query using **mysqli_query()** function.
  - Step 6: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Insert the data into Table Example:**

```
<?php
include("dbconfig.php");
$qry = "INSERT into abc(ID, name, age) VALUES (1,'X',21)";    //Step 3
$eqry = mysqli_query($con, $qry);                              //Step 4
mysqli_close($con);                                            //Step 5
?>
```

PHP –Database Connectivity

- **Insert the data into Table coming from HTML page:**

```
<?php
include("dbconfig.php");
$id=$_POST['idno'];
$n=$_POST['name'];
$a=$_POST['age'];
$qry = "INSERT into abc(ID, name, age) VALUES ('$id', '$n', '$a')";  //Step 3
$eqry = mysqli_query($con, $qry);                                     //Step 4
mysqli_close($con);                                                   //Step 5
?>
```

PHP –Database Connectivity

- **Select and Display the data from table:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Write a query for Select a data from table..
  - Step 4: Execute the query using **mysqli_query()** function.
  - Step 5: Fetch the data from results of **mysqli_fetch_array()** function.
  - Step 6: Close the Connection using **mysqli_close()** function.

PHP –Database Connectivity

- **Select and Display the data from table :**

```
<?php
include("dbconfig.php");
$qry = "SELECT * FROM abc";                                  //Step 3
$result = mysqli_query($con, $qry);                          //Step 4
echo "<table border=1> <tr><th> ID </th><th> Name
    </th><th>Age</th></tr>";
while($row = mysqli_fetch_array($result))                    //Step 5
{
    echo "<tr><td>" . $row['ID']. "</td>";
    echo "<td>" . $row['name']. "</td>";
    echo "<td>" . $row['age']. "</td></tr>";
}
echo "</table>";
mysqli_close($con);                                          //Step 6
?>
```

PHP –Database Connectivity

- **Populate the drop down menu with table values:**

```
<?php
include("dbconfig.php");
$qry = "SELECT id, sname FROM states";                    //Step 3
$result = mysqli_query($con, $qry);                       //Step 4
echo "<select name= states>";
while($row=mysqli_fetch_array($result)){                  //Step 5
echo "<option value="; echo $row['id']; echo ">"  echo $row['sname']; echo
    "</option>"; }
echo "</select>";
mysqli_close($con);                                       //Step 6
?>
```

PHP –Database Connectivity

- **Update the data:**

```
<?php
include("dbconfig.php");
$qry = "Update states SET sname='$_POST['sname']' WHERE ID='1'";
$result = mysqli_query($con, $qry);
if($result)
{ echo "Updated";}
else
{ echo "Error in Updating"; }
mysqli_close($con);
?>
```

PHP –Database Connectivity

- **Delete the data:**

```
<?php
include("dbconfig.php");
$qry = "delete from abc WHERE sname='$_POST['sn']'";
$result = mysqli_query($con, $qry);
if($result)
{ echo "Deleted";}
else
{ echo "Error in Deleting"; }
mysqli_close($con);
?>
```

PHP –Database Connectivity

- **Display all the table in database "mydb":**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Fetch the list of tables from database using **mysql_list_tables**() function.
  - Step 4: Fetch the number of rows of tables using **mysql_num_rows()** function.
  - Step 5: Take the table name using **mysql_tablename()** function.
  - Step 6: Print all the tables.

PHP –Database Connectivity

- **Display all the table in database “mydb”:**

```
<?php
include(“dbconfig.php”);
$result = mysql_list_tables(“mydb”);                    //Step 3
$rcount = mysql_num_rows($result);                      //Step 4
$tablist = “”;
for($i=0; $i<$rcount; $i++)
{
    $tabname = mysql_tablename($result, $i);            //Step 5
    $tablist .= $tabname.”<br/>”;
}
echo $tablist;                                          //Step 6
?>
```

PHP –Database Connectivity

- **Display all the databases:**
  - Step 1: Connect the Server using **mysqli_connect()** function.
  - Step 2: Check whether connection is established or not.
  - Step 3: Fetch the list of database using **mysql_list_dbs()** function
  - Step 4: Fetch the number of rows of databases using **mysql_num_rows()** function.
  - Step 5: Take the database name using **mysql_db_name()** function.
  - Step 6: Print all the tables.

PHP –Database Connectivity

- **Display all the databases:**

```
<?php
include("dbconfig.php");
$result = mysql_list_dbs($con);                    //Step 3
$rcount = mysql_num_rows($result);                 //Step 4
$dblist = "";
for($i=0; $i<$rcount; $i++)
{
    $dbname = mysql_db_name($result, $i);          //Step 5
    $dblist .= $dbname."<br/>";
}
echo $dblist;                                      //Step 6
?>
```

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 1:** Configure The "**php.ini**" File
  - **Step 2:** In your "php.ini" file, search for the **file_uploads** directive, and set it to On:

file_uploads = On

upload_max_filesize = 50M

  - **Step 3:** Creating HTML File:

```
<form action="upload.php" method="post" enctype="multipart/form-data">
    Select image to upload:
    <input type="file" name="ufile" id="ufile">
    <input type="submit" value="Upload Image" name="btn_submit">
</form>
```

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 4: Creating an upload script.**
    - There is one global PHP variable called **$_FILES**.
    - This variable is an associate double dimension array and keeps all the information related to uploaded file.
    - So if the value assigned to the input's name attribute in uploading form was **ufile**, then PHP would create following **five variables**.
    - $_FILES[‘**ufile**’][‘name’], $_FILES[‘**ufile**’][‘temp_name’], $_FILES[‘**ufile**’][‘type’], $_FILES[‘**ufile**’][‘size’], $_FILES[‘**ufile**’][‘error’],

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 4: Creating an upload script.**
    - $_FILES[‘**ufile**’][‘name’]: This array value specifies the original name of the file, including the file extension. It doesn't include the file path.
    - $_FILES[‘**ufile**’][‘temp_name’]: This array value specifies the temporary name including full path that is assigned to the file once it has been uploaded to the server.
    - $_FILES[‘**ufile**’][‘type’]: The mime type of the file, an example would be *"image/gif".*
    - $_FILES[‘**ufile**’][‘size’]: The size, in bytes, of the uploaded file.
    - $_FILES[‘**ufile**’][‘error’]: The error code associated with the file upload.

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 4: Creating an upload script.**

```
<?php
    include("dbconfig.php");
        if(isset($_POST['btn_submit']))
        {
            $errors= array();
            $file = $_FILES['file']['name'];
            $file_loc = $_FILES['file']['tmp_name'];
            $file_size = $_FILES['file']['size'];
            $file_type = $_FILES['file']['type'];
            $folder="uploads/";
```

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 4: Creating an upload script.**

```
//check for file extension starts


$file_ext=strtolower(end(explode('.',$_FILES['image']['name'])));
$extslist = array("jpeg","jpg","png");
if(in_array($file_ext,$extslist) == false)                    //to check in the array
{
$errors[]="extension not allowed, please choose a JPEG or PNG file.";
}

//check for file extension ends
```

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 4: Creating an upload script.**

```
//check for file size starts


if($file_size > 2097152)
 {
 $errors[]='File size must be exactly 2 MB';
 }
$final_file=strtolower(str_replace(' ','-',$file));

//check for file size ends
```

PHP –Database Connectivity

- **File Upload on Server:**
  - **Step 4: Creating an upload script.**

```
if(empty($errors)==true)
{
    move_uploaded_file($file_loc,$folder.$final_file);
     $sql="INSERT INTO upload(ufile) VALUES('$final_file')";
    mysqli_query($con,$sql);
    echo "Success";
}
else
{
    print_r($errors);
} } ?>
```

PHP –Implementation of CRUD operations

What is CRUD?

- CRUD refers to the four basic types of Database operations: **C**reate, **R**ead, **U**pdate, **D**elete. Most applications and projects perform some kind of CRUD functionality.

- Once we learn about these CRUD operations, we can use them for many projects.
- For an example, if we learn how to create student table with multiple columns, you can use similar approach to create employee table or customers table.

PHP –Implementation of CRUD operations

About Simple CRUD Script:

- We have created a Simple CRUD (Create/Read/Update/Delete) Application in PHP that is easy to understand.

- This CRUD Script has the minimum number of functions required, so that anybody can easily understand the code.

- This script does not ensure security and data validation. So, if you want to use this script for your live project, please make sure to validation user data..

PHP –Implementation of CRUD operations

How to run CRUD Script:

- You just need to run the given code files and database at your server. This script should run properly after database setup on most local server. But it may require few configuration changes on remote/live server.
- Copy complete crud folder into your local server (in htdocs or www)
- Create a new database "crud_db” (You can use PHPMyAdmin)
- Import database.sql file from crud folder into database "crud_db“
- Now run files from crud folder using your server/folder path (For example: http://localhost/crud/)
- It should works properly. You can Add some user data. Then you can test Edit/Update/Delete-User functions as well

PHP –Implementation of CRUD operations

- **config.php** file store information about the database host, username and password. Most of the local server works with given detail. We can change as per our host and database details

```
<?php
$databaseHost = 'localhost';
$databaseName = 'crud_db';
$databaseUsername = 'root';
$databasePassword = ‘’;

$mysqli = mysqli_connect($databaseHost, $databaseUsername, $databasePassword, $databaseName);
?>
```

PHP –Implementation of CRUD operations

- **database.sql** is a database file. You can to create a database, after downloading the crud script. You can either run commands from above mysql file or simply import this 'database.sql' file into database (example using PHPMyAdmin).

```
//create database crud_db;
//use crud_db;
CREATE TABLE `users` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(100),
  `email` varchar(100),
  `mobile` varchar(15),
  PRIMARY KEY  (`id`)
);
```

PHP –Implementation of CRUD operations

- **index.php** file is the main file which include configuration file for database connection. Then it display all users list using MySQL Select Query. However, you need to add some users first using 'Add New User' link

```
<?php
    include_once("config.php");
    $result = mysqli_query($mysqli, "SELECT * FROM users ORDER BY id DESC");
?>
<html><head>   <title>Homepage</title> </head>
<body>
<a href="add.php">Add New User</a><br/><br/>
```

PHP –Implementation of CRUD operations

```
<table width='80%' border=1>
 <tr><th>Name</th> <th>Mobile</th> <th>Email</th> <th>Update</th>    </tr>
  <?php
  while($user_data = mysqli_fetch_array($result)) {
     echo "<tr>";
     echo "<td>".$user_data['name']."</td>";
     echo "<td>".$user_data['mobile']."</td>";
     echo "<td>".$user_data['email']."</td>";
     echo "<td><a href='edit.php?id=$user_data[id]'>Edit</a> | <a
href='delete.php?id=$user_data[id]'>Delete</a></td></tr>";
  }
  ?>   </table></body></html>
```

PHP –Implementation of CRUD operations

- **add.php** file is responsible to add new users. HTML Form is used to capture user data. After User data is submitted, MySQL INSERT Query is used to insert user data into database. You can “View User” after user added

```
<html><head>      <title>Add Users</title></head>
<body>
        <a href="index.php">Go to Home</a>              <br/><br/>
        <form action="add.php" method="post" name="form1">
                <table width="25%" border="0">
                        <tr>
                                <td>Name</td>
                                <td><input type="text"name="name"></td>
                        </tr>
```

PHP –Implementation of CRUD operations

```
                <tr>    <td>Email</td>
                        <td><input type="text"name="email"></td>
                </tr>
                <tr>    <td>Mobile</td>
                        <td><input type="text“ name="mobile“ >
                        </td>
                </tr>
                <tr>    <td><input  type="submit"  name="Submit"
                        value="Add"></td>
                </tr>
        </table>
</form>
```

PHP –Implementation of CRUD operations

```
<?php    // Check If form submitted, insert form data into users table.
if(isset($_POST['Submit'])) {
        $name = $_POST['name'];
        $email = $_POST['email'];
        $mobile = $_POST['mobile'];
        include_once("config.php");
        // Insert user data into table
        $result = mysqli_query($mysqli, "INSERT INTO users(name,email,mobile) VALUES('$name','$email','$mobile')");
        // Show message when user added
        echo "User added successfully. <a href='index.php'>View Users</a>";
        }        ?> </body></html>
```

PHP –Implementation of CRUD operations

- **edit.php** is used to edit/update user data once user click on'Edit' link. It first fetch user's current data in form. You can change user data and update it. It will redirect to homepage, after successful update.

```
<?php include_once("config.php");
if(isset($_POST['update'])){
        $id = $_POST['id'];
        $name=$_POST['name'];
        $mobile=$_POST['mobile'];
        $email=$_POST['email']; // update user data
        $result = mysqli_query($mysqli, "UPDATE users SET name='$name',email='$email',mobile='$mobile' WHERE id=$id"); // Redirect to homepage to display updated user in list
        header("Location: index.php");}?>
```

PHP –Implementation of CRUD operations

```
<?php
// Display selected user data based on id  // Getting id from url
$id = $_GET['id'];
// Fetech user data based on id
$result = mysqli_query($mysqli, "SELECT * FROM users WHERE id=$id");
while($user_data = mysqli_fetch_array($result))
{
        $name = $user_data['name'];
        $email = $user_data['email'];
        $mobile = $user_data['mobile'];
}
?>
```

PHP –Implementation of CRUD operations

```
<html> <head>      <title>Edit User Data</title> </head>
<body>
            <a href="index.php">Home</a>        <br/><br/>
            <form name="update_user" method="post" action="edit.php">
                        <table border="0">
                                    <tr>        <td>Name</td>
                                                <td><input      type="text"       name="name"
value=<?php echo $name;?>></td>                 </tr>
                                    <tr>        <td>Email</td>
                                                <td><input      type="text"       name="email"
value=<?php echo $email;?>></td>
                                    </tr>
```

PHP –Implementation of CRUD operations

```
<tr>
<td>Mobile</td>
<td><input type="text" name="mobile" value=<?php echo $mobile;?>></td>
</tr>
<tr>
<td><input type="hidden" name="id" value=<?php echo $_GET['id'];?>></td>
<td><input type="submit" name="update" value="Update"></td>
</tr>
</table>
</form>
</body> </html>
```

PHP –Implementation of CRUD operations

- **delete.php** file simply called when we click on 'Delete' link for any user. It will delete selected user. Delete/edit/update operations user particular user_id to identify users.

```
<?php
    include_once("config.php");
    // Get id from URL to delete that user
    $id = $_GET['id'];
    // Delete user row from table based on given id
    $result = mysqli_query($mysqli, "DELETE FROM users WHERE id=$id");
    // After delete redirect to Home, so that latest user list will be displayed.
    header("Location:index.php");
?>
```

PHP – Prepared Statement

- Database management systems that work with the database language SQL are widely popular but have always been **vulnerable** to manipulations during data input.
- User input that hasn't been masked enough or contains metacharacters such as quotation marks or semicolons makes an easy catch for predators.
- One possible **solution** to this problem is to use prepared statements, which are pre-prepared instructions for the database that aren't given values until they're run.
- What makes this method so special, and when can it be used? In what follows, we use the example of MySQL to show how prepared statements work and how they can be used for database management.

PHP – Prepared Statement

What are prepared statements?

- **Prepared** statements are ready-to-use **templates** for queries in SQL database systems, which **don't** contain **values** for the individual parameters.
- **Instead**, these statement templates work with **variables** or **placeholders** that are only replaced with the actual values inside the system – unlike with manual input, in which values are already assigned at execution.
- All **major SQL database management** systems like MySQL, MariaDB, Oracle, Microsoft SQL Server, and PostgreSQL **support** prepared statements.
- Most of these applications use a **NoSQL binary protocol**. However, some systems such as MySQL use typical SQL syntax for implementation.
- If you use **PHP** for **database access**, you have the choice between using the object-oriented interface PHP Data Objects (**PDO**) or the PHP extension **MySQLi** for **implementing** prepared statements.

PHP – Prepared Statement

Why does it make sense to use prepared statements in MySQL and co.?

- The main reason for working with prepared statements in database management systems like MySQL is **security**.
- The biggest problem with standard access to SQL databases is probably that they can be easily **manipulated**.
- What you're dealing with in this case is an **SQL Injection**, in which code is inserted or adapted in order to gain access to sensible data or gain complete control of the database.
- Prepared statements in PHP or other languages don't have this vulnerability, since they're only assigned concrete values within the system.

PHP – Prepared Statement

How SQL Injection Works

- In a normal MySQL call, you would do something like:

```
$name = $_POST['name'];
$mysqli->query("SELECT * FROM myTable WHERE name='$name'");
```

- The problem with this, is that if it is based on user input, like in the example, then a malicious user could do ' OR '1'='1. Now this statement will always evaluate to true, since 1=1. In this case, the malicious user now has access to your entire table. Just imagine what could happen if it were a DELETE query instead.
- Take a look at what is actually happening to the statement.

```
SELECT * FROM myTable WHERE name='' OR '1'='1'
```

PHP – Prepared Statement

How SQL Injection Works

- A hacker could do a lot of damage to your site if your queries are set up like this. An easy fix to this would be to do:
- $name = $mysqli->real_escape_string($_POST['name']);
- $mysqli->query("SELECT * FROM myTable WHERE name='$name'");
- Notice how similar to the first example, I still added quotes to the column value. Without quotes, strings are still equally susceptible to SQL injection. If you'll be using a LIKE clause, then you should also do addcslashes($escaped, '%_'), since mysqli::real_escape_string won't do this as stated here.

PHP – Prepared Statement

How SQL Injection Works

- This covers strings, as the function name implies, but what about numbers? You could do (int)$mysqli->real_escape_string($_POST['name']), which would certainly work, but that's redundant.
- If you're casting the variable to an int, you don't need to escape anything. You are already telling it to essentially make sure that the value will be an integer. Doing (int)$_POST['name'] would suffice. Since it is an integer you also obviously do not need to add quotes to the sql column name.

PHP – Prepared Statement

Why does it make sense to use prepared statements in MySQL and co.?

- But protection against SQL injections isn't the only argument for using prepared statements: Once they've been analyzed and compiled, prepared statements can be used over and over again by the database system (with the appropriately modified values).
- In other words, they use **fewer resources** and are **faster** than manual database queries when it comes to SQL tasks that have to be repeatedly executed.

PHP – Prepared Statement

How exactly do prepared statements work?

- Leaving out the syntax of the underlying scripting language and idiosyncrasies of individual database management systems, integrating and using a prepared statement generally happens in the following stages:

Stage 1: **Preparing the prepared statements**

- The first step is to create a statement template – in PHP, you can do this with the function **prepare()**. Instead of concrete values for the relevant parameters, the above-mentioned placeholders (also called bind variables) are inserted. They're typically marked with a "?", as in the following example.

```
INSERT INTO Products (Name, Price) VALUES (?, ?);
```

- Complete prepared statements are then forwarded to the database management system.

PHP – Prepared Statement

How exactly do prepared statements work?

Stage 2: **Processing the statement template with the DBMS**

- The statement template will then be parsed by the database management system so that it can be compiled, i.e., converted into an executable statement. The prepared statement is also optimized as a part of this process.

Stage 3: **Execution of the prepared statement**

- The processed template can later be executed in the database system as often as desired. The only requirement for this is appropriate input from the connected application or data source, which has to provide the values for the placeholder fields. With reference to the code example from Stage 1, this could be the values “Book” (Name) and “10” (Price) or “Computer” and “1000”.

PHP – Prepared Statement

PREPARE, EXECUTE, and DEALLOCATE PREPARE: The three basic SQL commands for using prepared statements

- There are three SQL commands that play a crucial role in prepared statements in MySQL databases:

- `The command "**PREPARE**" is necessary for preparing a prepared statement for use and for assigning it a unique name under which it can be controlled later in the process.

  PREPARE stmt_name FROM preparable_stmt

PHP – Prepared Statement

PREPARE, EXECUTE, and DEALLOCATE PREPARE: The three basic SQL commands for using prepared statements

- For the execution of prepared statements in SQL, you'll need the command "**EXECUTE**". You can refer to the relevant prepared statement by entering the name that was generated with "PREPARE". A statement can be executed as often as you'd like – you can use it to define various variables or transfer new values for the variables you set.

```
EXECUTE stmt_name
        [USING @var_name [, @var_name] ...]
```

PHP – Prepared Statement

PREPARE, EXECUTE, and DEALLOCATE PREPARE: The three basic SQL commands for using prepared statements

- In order to deallocate a PHP prepared statement, use the command "**DEALLOCATE PREPARE**". Alternatively, statements can be automatically deallocated at the end of a session.
- Deallocation is important because otherwise you'll quickly reach the limit defined by the system variable max_prepared_stmt_count. Then you won't be able to create any new prepared statements.

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

PHP – Prepared Statement

Example – (preparedstatement)

- This PHP script establishes the connection to the MySQL database with ($conn), at which point the individual server data needs to be entered.
- The crucial prepared statement part begins with the line "INSERT INTO MyCustomers (FirstName, LastName, Email) VALUES (?, ?, ?)".
- The customer database “MyCustomers” will receive input (INSERT INTO) in the columns “FirstName”, “LastName” and “Email”.
- Placeholders are used for VALUES, which are marked using question marks.

PHP – Prepared Statement

Example

- Next, the parameters need to be bound (bind_parameters). In addition, the database also needs information about what type of data is being dealt with. The argument "sss" used here indicates that all three parameters will be strings.
- Some possible alternative data types would be:

  i: INTEGER (whole number)

  d: DOUBLE (also called a float, a number with a decimal point or a number in exponential form)

  b: BLOB (collection of binary data)

PHP – stored procedure execution

Getting started with stored procedures

- The following SELECT statement returns all rows in the table customers from the sample database:

```
SELECT
    customerName,
    city,
    state,
    postalCode,
    country
FROM
    customers
ORDER BY customerName;
```

PHP – stored procedure execution

Getting started with stored procedures

- This picture shows the partial output of the query:

customerName	city	state	postalCode	country
Alpha Cognac	Toulouse	NULL	31000	France
American Souvenirs Inc	New Haven	CT	97823	USA
Amica Models & Co.	Torino	NULL	10100	Italy
ANG Resellers	Madrid	NULL	28001	Spain
Anna's Decorations, Ltd	North Sydney	NSW	2060	Australia
Anton Designs, Ltd.	Madrid	NULL	28023	Spain
Asian Shopping Network, Co	Singapore	NULL	038988	Singapore
Asian Treasures, Inc.	Cork	Co. Cork	NULL	Ireland
Atelier graphique	Nantes	NULL	44000	France
Australian Collectables, Ltd	Glen Waverly	Victoria	3150	Australia
Australian Collectors, Co.	Melbourne	Victoria	3004	Australia

PHP – stored procedure execution

Getting started with stored procedures

- When you use MySQL Workbench or mysql shell to issue the query to MySQL Server, MySQL processes the query and returns the result set.

- If you want to save this query on the database server for execution later, one way to do it is to use a stored procedure.

PHP – stored procedure execution

Getting started with stored procedures

- The following CREATE PROCEDURE statement creates a new stored procedure that wraps the query above:

```
DELIMITER $$
CREATE PROCEDURE GetCustomers()
BEGIN
        SELECT
                customerName,
                city,
                state,
                postalCode,
                country
        FROM
                customers
        ORDER BY customerName;
END$$
DELIMITER ;
```

PHP – stored procedure execution

Getting started with stored procedures

- By definition, a stored procedure is a segment of declarative SQL statements stored inside the MySQL Server. In this example, we have just created a stored procedure with the name GetCustomers().
- Once we save the stored procedure, you can invoke it by using the CALL statement:

```
CALL GetCustomers();
```

- And the statement returns the same result as the query.
- The first time you invoke a stored procedure, MySQL looks up for the name in the database catalog, compiles the stored procedure's code, place it in a memory area known as a cache, and execute the stored procedure.
- If you invoke the same stored procedure in the same session again, MySQL just executes the stored procedure from the cache without having to recompile it.

PHP – stored procedure execution

- A stored procedure can have parameters so you can pass values to it and get the result back. For example, you can have a stored procedure that returns customers by country and city. In this case, the country and city are parameters of the stored procedure.
- A stored procedure may contain control flow statements such as IF, CASE, and LOOP that allow you to implement the code in the procedural way.
- A stored procedure can call other stored procedures or stored functions, which allows you to modulize your code.

PHP – stored procedure execution

MySQL stored procedures advantages

Reduce network traffic

- Stored procedures help reduce the network traffic between applications and MySQL Server. Because instead of sending multiple lengthy SQL statements, applications have to send only the name and parameters of stored procedures.

Centralize business logic in the database

- You can use the stored procedures to implement business logic that is reusable by multiple applications. The stored procedures help reduce the efforts of duplicating the same logic in many applications and make your database more consistent.

Make database more secure

- The database administrator can grant appropriate privileges to applications that only access specific stored procedures without giving any privileges on the underlying tables.

PHP – stored procedure execution

MySQL stored procedures disadvantages

Resource usages

- If you use many stored procedures, the memory usage of every connection will increase substantially.
- Besides, overusing a large number of logical operations in the stored procedures will increase the CPU usage because the MySQL is not well-designed for logical operations.

Troubleshooting

- It's difficult to debug stored procedures. Unfortunately, MySQL does not provide any facilities to debug stored procedures like other enterprise database products such as Oracle and SQL Server.

Maintenances

- Developing and maintaining stored procedures often requires a specialized skill set that not all application developers possess. This may lead to problems in both application development and maintenance.

PHP – stored procedure execution

MySQL CREATE PROCEDURE statement

- This query returns all products in the products table from the sample database.

```
SELECT * FROM products;
```

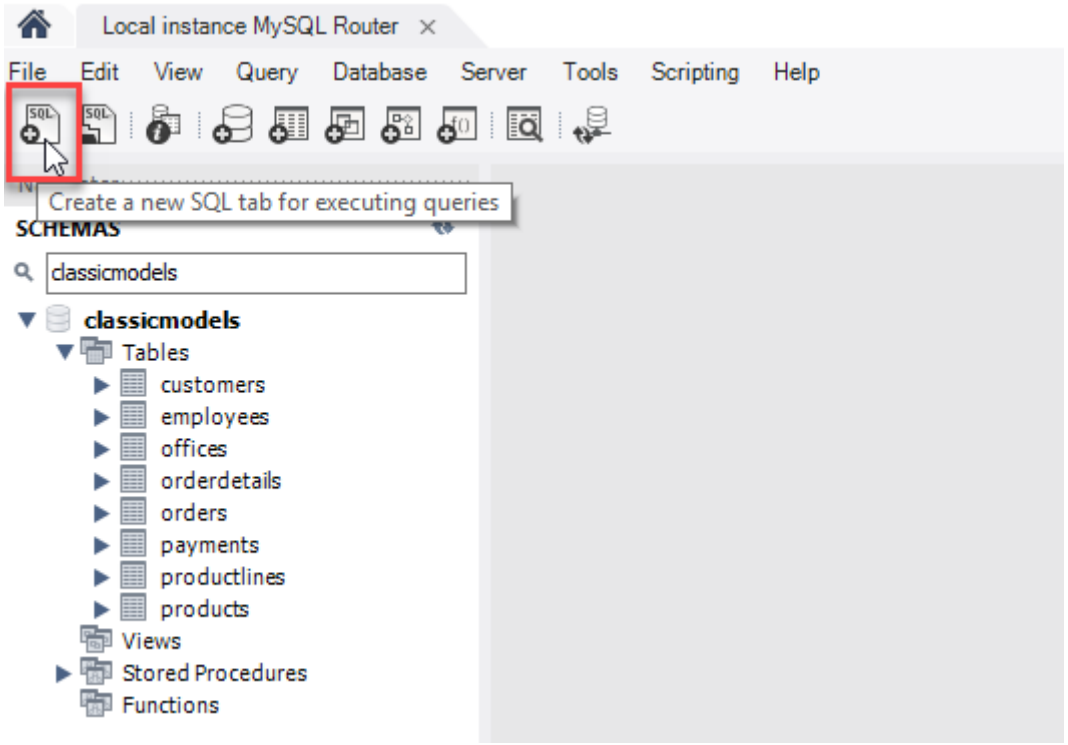
- The following statement creates a new stored procedure that wraps the query:

```
DELIMITER //

CREATE PROCEDURE GetAllProducts()
BEGIN
    SELECT *  FROM products;
END //

DELIMITER ;
```

PHP – stored procedure execution

MySQL CREATE PROCEDURE statement

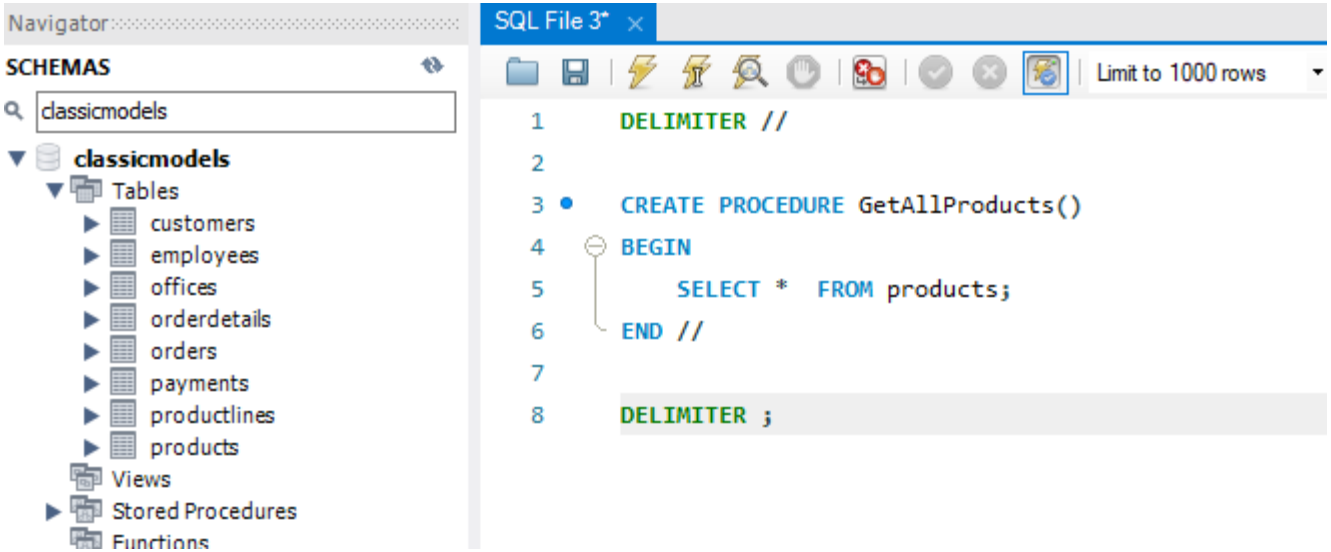
- To execute these statements:
- First, launch MySQL Workbench.
- Second, create a new SQL tab for executing queries:



PHP – stored procedure execution

MySQL CREATE PROCEDURE statement

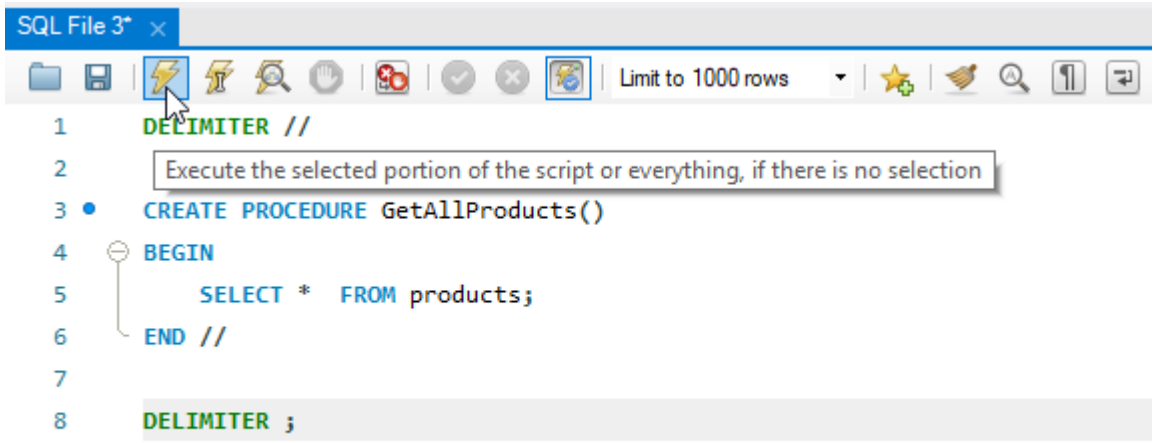
- Third, enter the statements in the SQL tab:



PHP – stored procedure execution

MySQL CREATE PROCEDURE statement

- Fouth, execute the statements. Note that you can select all statements in the SQL tab (or nothing) and click the Execute button. If everything is fine, MySQL will create the stored procedure and save it in the server.



PHP – stored procedure execution

MySQL CREATE PROCEDURE statement

- Fifth, check the stored procedure by opening the Stored Procedures node. If you don't see the stored procedure, you can click the Refresh button next to the SCHEMAS title: