A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

PHPMyAdmin only needs to be used when you need to create or modify a database or tables within a database. Once the database and tables are created for your project, you will not need to work with PHPMyAdmin that much. It is still important to understand how to use and get used to its quirks which are largely due to it being a web-based application.

PHPMyAdmin is a common tool found on most third-party web hosting services, so if you ever create/admin/maintain an online web application, you will most likely have to use it

A screenshot of a computer

AI-generated content may be incorrect.

Comments –

<?php

//Connection information for finding and connecting to the MySQL server

//Since this is a development environment in XAMPP, we are using 'root' user with no password

$dsn = 'mysql:host=localhost;dbname=comment\_app';

$dbuser = 'root';

$dbpass = '';

//use the variables above to create a new PDO object: $db

//This variable now contains the information needed to interact

//with the MySQL database

$db = new PDO($dsn, $dbuser, $dbpass);

function addComment($name, $email, $comment)

{

//This function inserts a new record in the table

global $db; //Make $db accessible inside the function block

//The SQL - this is to add a new record in the table

//Note the 'placeholders' :col\_one, :col\_two, :col\_three

$query = 'INSERT INTO comments(name, email, comment) VALUES (:name, :email, :comment)';

//Call the prepare method from the $db object

//to setup a prepared (secure) interaction with the database

$statement = $db->prepare($query);

//'bind' each variable to the placeholders specified in the SQL query

$statement->bindValue(':name', $name);

$statement->bindValue(':email', $email);

$statement->bindValue(':comment', $comment);

//Execute the SQL command

$statement->execute();

//Our interaction with the DB is done, close the connection to the server

$statement->closeCursor();

//There is nothing to return from this function

}

function getComments()

{

//This function finds all records in the specified table

//and returns them as a result set (2-dimensional associative array)

global $db;

//Setup the SQL statement - no placeholders needed

$query = 'SELECT \* FROM comments';

$statement = $db->prepare($query);

$statement->execute();

//We use the fetchAll() function because we expect

//that there could be multiple results

//After this statement, $records contains all the data for the found records

// stored as a 2-dimensional associative array

$comments = $statement->fetchAll();

$statement->closeCursor();

//return the $records found

return $comments;

}

function getComment($id)

{

//This function finds a single record based on the id (primary key)

global $db; //Make the $db visible within the function

//Since we are using a variable $id as part of the SQL statement

//we need to use a placeholder ':id'

$query = 'SELECT \* FROM comments WHERE id = :id';

$statement = $db->prepare($query);

//'bind' the variable to the placeholder

$statement->bindValue(':id', $id);

$statement->execute();

//This query will find at most 1 matching record,

//so we use the fetch() function here instead of fetchAll()

$comment = $statement->fetch();

$statement->closeCursor();

//return the record- $comment is a 1-dimensional associative array

return $comment;

}

function updateComment($id, $name, $email, $comment)

{

global $db; //Make $db accessible inside the function block

//The SQL - this is to update an existing entry in the table

//Note the 'placeholders' :col\_one, :col\_two, :col\_three, :id

$query = 'UPDATE comments SET name= :name, email= :email, comment= :comment WHERE id = :id';

//Call the prepare method from the $db object

//to setup a prepared (secure) interaction with the database

$statement = $db->prepare($query);

//'bind' each variable to the placeholders specified in the SQL query

$statement->bindValue(':name', $name);

$statement->bindValue(':email', $email);

$statement->bindValue(':comment', $comment);

$statement->bindValue(':id', $id);

//Execute the SQL command

$statement->execute();

//Our interaction with the DB is done, close the connection to the server

$statement->closeCursor();

//There is nothing to return from this function

}

function deleteComment($id)

{

//This function deletes an single record from the table based on the id

global $db;

//setup the query - id comes from the client, so use a placeholder, :id

$query = 'DELETE FROM comments WHERE id = :id';

$statement = $db->prepare($query);

$statement->bindValue(':id', $id);

$statement->execute();

$statement->closeCursor();

//There is nothing to return from this function

}

$action = filter\_input(INPUT\_POST, 'action');

if($action == NULL)

$action = filter\_input(INPUT\_GET, 'action');

if($action == "Submit Comment")

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

addComment($name, $email, $comment)

?>

<!DOCTYPE html>

<html>

<head>

<title>Simple Form Application - PHP</title>

<link rel="stylesheet" href="main.css">

</head>

<body>

<p><a href="comments.php">Back to HTML Form</a></p>

<main>

<h1>User Submitted Comments</h1>

<p>Name: <?php echo $name; ?> </p>

<p>Email: <?php echo $email; ?> </p>

<p>Comments: <?php echo $comment; ?></p>

</main>

</body>

</html>

<?php

}

elseif($action == 'edit')

{

//User is trying to edit and update a specific comment.

//This request is sent via the 'GET' method when the user clicks the 'edit' link

//first capture the 'id' value of the record to be updated

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'getComment' function to get the data for that record

$commentRecord = getComment($id);

//Assign the values from the database results to individual variables

$name = $commentRecord['name'];

$email = $commentRecord['email'];

$comment = $commentRecord['comment'];

//Finally show the 'edit' form populated with the data from the record

//Note that we use a 'hidden' input element to include the record id in the form

?>

<!DOCTYPE html>

<html>

<head>

<title>Simple Form Application - PHP</title>

<link rel="stylesheet" href="main.css">

</head>

<body>

<main>

<h1>Update Comment Form</h1>

<form action="comments.php" method="post">

<fieldset>

<legend>Enter Your Comments</legend>

<label for="name">Name:</label>

<input type="text" name="name" id="name" value = "<?php echo $name ?>"><br>

<label for="email">Email:</label>

<input type="text" name="email" id="email" value = "<?php echo $email ?>"><br>

<label for="comments">Comments</label><br>

<textarea name="comment" id="comment">

<?php echo $comment ?>

</textarea><br>

<input type = "hidden" name="id" value = "<?php echo $id ?>">

<input type="submit" name = "action" value="Update Comment" >

</fieldset>

</main>

</body>

</html>

<?php

}

elseif($action == 'Update Comment')

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

$id = filter\_input(INPUT\_POST, 'id');

//call the updateComment function and send the captured data as parameters

updateComment($id, $name, $email, $comment);

//Redirect the user back to the default view

header("Location:comments.php");

}

elseif($action == 'delete')

{

//User is trying to delete a secific comment.

//This request is sent via the 'GET' method when the user clicks the 'delete' link

//first capture the 'id' value of the record to be deleted

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'deleteComment' function to delete that record

deleteComment($id);

//Redirect the user back to the default view

header("Location:comments.php");

}

else

{

//default view: Shows the new comment form and the list of comments

//

$comments = getComments();

if($comments != NULL){

$commentList = "<h2>Comments Found:</h2>";

foreach($comments as $row){

$name = $row['name'];

$email = $row['email'];

$comment = $row['comment'];

$id = $row['id'];

$commentList .= "<p>Name: $name email: $email <a href = 'comments.php?action=edit&id=$id'>edit</a> | <a href = 'comments.php?action=delete&id=$id'>delete</a></p><p>$comment</p><hr>";

}

}

else

$commentList = "<h2> No comments found</h2>";

?>

<!DOCTYPE html>

<html>

<head>

<title>Simple Form Application - PHP</title>

<link rel="stylesheet" href="main.css">

</head>

<body>

<main>

<h1>User Input Form</h1>

<form action="comments.php" method="post">

<fieldset>

<legend>Enter Your Comments</legend>

<label for="name">Name:</label>

<input type="text" name="name" id="name"><br>

<label for="email">Email:</label>

<input type="text" name="email" id="email"><br>

<label for="comments">Comments</label><br>

<textarea name="comment" id="comment">

</textarea><br>

<input type="submit" name = "action" value="Submit Comment" >

</fieldset>

</form>

<?php echo $commentList ?>

</main>

</body>

</html>

<?php

}

?>

Final version –

<?php

//Connection information for finding and connecting to the MySQL server

//Since this is a development environment in XAMPP, we are using 'root' user with no password

$dsn = 'mysql:host=localhost;dbname=comment\_app';

$dbuser = 'root';

$dbpass = '';

//use the variables above to create a new PDO object: $db

//This variable now contains the information needed to interact

//with the MySQL database

$db = new PDO($dsn, $dbuser, $dbpass);

function addComment($name, $email, $comment)

{

//This function inserts a new record in the table

global $db; //Make $db accessible inside the function block

//The SQL - this is to add a new record in the table

//Note the 'placeholders' :col\_one, :col\_two, :col\_three

$query = 'INSERT INTO comments(name, email, comment) VALUES (:name, :email, :comment)';

//Call the prepare method from the $db object

//to setup a prepared (secure) interaction with the database

$statement = $db->prepare($query);

//'bind' each variable to the placeholders specified in the SQL query

$statement->bindValue(':name', $name);

$statement->bindValue(':email', $email);

$statement->bindValue(':comment', $comment);

//Execute the SQL command

$statement->execute();

//Our interaction with the DB is done, close the connection to the server

$statement->closeCursor();

//There is nothing to return from this function

}

function getComments()

{

//This function finds all records in the specified table

//and returns them as a result set (2-dimensional associative array)

global $db;

//Setup the SQL statement - no placeholders needed

$query = 'SELECT \* FROM comments';

$statement = $db->prepare($query);

$statement->execute();

//We use the fetchAll() function because we expect

//that there could be multiple results

//After this statement, $records contains all the data for the found records

// stored as a 2-dimensional associative array

$comments = $statement->fetchAll();

$statement->closeCursor();

//return the $records found

return $comments;

}

function getComment($id)

{

//This function finds a single record based on the id (primary key)

global $db; //Make the $db visible within the function

//Since we are using a variable $id as part of the SQL statement

//we need to use a placeholder ':id'

$query = 'SELECT \* FROM comments WHERE id = :id';

$statement = $db->prepare($query);

//'bind' the variable to the placeholder

$statement->bindValue(':id', $id);

$statement->execute();

//This query will find at most 1 matching record,

//so we use the fetch() function here instead of fetchAll()

$comment = $statement->fetch();

$statement->closeCursor();

//return the record- $comment is a 1-dimensional associative array

return $comment;

}

function updateComment($id, $name, $email, $comment)

{

global $db; //Make $db accessible inside the function block

//The SQL - this is to update an existing entry in the table

//Note the 'placeholders' :col\_one, :col\_two, :col\_three, :id

$query = 'UPDATE comments SET name= :name, email= :email, comment= :comment WHERE id = :id';

//Call the prepare method from the $db object

//to setup a prepared (secure) interaction with the database

$statement = $db->prepare($query);

//'bind' each variable to the placeholders specified in the SQL query

$statement->bindValue(':name', $name);

$statement->bindValue(':email', $email);

$statement->bindValue(':comment', $comment);

$statement->bindValue(':id', $id);

//Execute the SQL command

$statement->execute();

//Our interaction with the DB is done, close the connection to the server

$statement->closeCursor();

//There is nothing to return from this function

}

function deleteComment($id)

{

//This function deletes an single record from the table based on the id

global $db;

//setup the query - id comes from the client, so use a placeholder, :id

$query = 'DELETE FROM comments WHERE id = :id';

$statement = $db->prepare($query);

$statement->bindValue(':id', $id);

$statement->execute();

$statement->closeCursor();

//There is nothing to return from this function

}

$action = filter\_input(INPUT\_POST, 'action');

if($action == NULL)

$action = filter\_input(INPUT\_GET, 'action');

if($action == "Submit Comment")

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

addComment($name, $email, $comment)

?>

<!DOCTYPE html>

<html>

<head>

<title>Simple Form Application - PHP</title>

<link rel="stylesheet" href="main.css">

</head>

<body>

<p><a href="comments.php">Back to HTML Form</a></p>

<main>

<h1>User Submitted Comments</h1>

<p>Name: <?php echo $name; ?> </p>

<p>Email: <?php echo $email; ?> </p>

<p>Comments: <?php echo $comment; ?></p>

</main>

</body>

</html>

<?php

}

elseif($action == 'edit')

{

//User is trying to edit and update a specific comment.

//This request is sent via the 'GET' method when the user clicks the 'edit' link

//first capture the 'id' value of the record to be updated

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'getComment' function to get the data for that record

$commentRecord = getComment($id);

//Assign the values from the database results to individual variables

$name = $commentRecord['name'];

$email = $commentRecord['email'];

$comment = $commentRecord['comment'];

//Finally show the 'edit' form populated with the data from the record

//Note that we use a 'hidden' input element to include the record id in the form

?>

<!DOCTYPE html>

<html>

<head>

<title>Simple Form Application - PHP</title>

<link rel="stylesheet" href="main.css">

</head>

<body>

<main>

<h1>Update Comment Form</h1>

<form action="comments.php" method="post">

<fieldset>

<legend>Enter Your Comments</legend>

<label for="name">Name:</label>

<input type="text" name="name" id="name" value = "<?php echo $name ?>"><br>

<label for="email">Email:</label>

<input type="text" name="email" id="email" value = "<?php echo $email ?>"><br>

<label for="comments">Comments</label><br>

<textarea name="comment" id="comment">

<?php echo $comment ?>

</textarea><br>

<input type = "hidden" name="id" value = "<?php echo $id ?>">

<input type="submit" name = "action" value="Update Comment" >

</fieldset>

</main>

</body>

</html>

<?php

}

elseif($action == 'Update Comment')

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

$id = filter\_input(INPUT\_POST, 'id');

//call the updateComment function and send the captured data as parameters

updateComment($id, $name, $email, $comment);

//Redirect the user back to the default view

header("Location:comments.php");

}

elseif($action == 'delete')

{

//User is trying to delete a secific comment.

//This request is sent via the 'GET' method when the user clicks the 'delete' link

//first capture the 'id' value of the record to be deleted

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'deleteComment' function to delete that record

deleteComment($id);

//Redirect the user back to the default view

header("Location:comments.php");

}

else

{

//default view: Shows the new comment form and the list of comments

//

$comments = getComments();

if($comments != NULL){

$commentList = "<h2>Comments Found:</h2>";

foreach($comments as $row){

$name = $row['name'];

$email = $row['email'];

$comment = $row['comment'];

$id = $row['id'];

$commentList .= "<p>Name: $name email: $email <a href = 'comments.php?action=edit&id=$id'>edit</a> | <a href = 'comments.php?action=delete&id=$id'>delete</a></p><p>$comment</p><hr>";

}

}

else

$commentList = "<h2> No comments found</h2>";

?>

<!DOCTYPE html>

<html>

<head>

<title>Simple Form Application - PHP</title>

<link rel="stylesheet" href="main.css">

</head>

<body>

<main>

<h1>User Input Form</h1>

<form action="comments.php" method="post">

<fieldset>

<legend>Enter Your Comments</legend>

<label for="name">Name:</label>

<input type="text" name="name" id="name"><br>

<label for="email">Email:</label>

<input type="text" name="email" id="email"><br>

<label for="comments">Comments</label><br>

<textarea name="comment" id="comment">

</textarea><br>

<input type="submit" name = "action" value="Submit Comment" >

</fieldset>

</form>

<?php echo $commentList ?>

</main>

</body>

</html>

<?php

}

?>

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

Comments\_MVC –

<?php

require("model/functions.php");

$action = filter\_input(INPUT\_POST, 'action');

if($action == NULL)

$action = filter\_input(INPUT\_GET, 'action');

if($action == "Submit Comment")

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

addComment($name, $email, $comment);

include("view/comment\_submitted.php");

}

elseif($action == 'edit')

{

//User is trying to edit and update a specific comment.

//This request is sent via the 'GET' method when the user clicks the 'edit' link

//first capture the 'id' value of the record to be updated

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'getComment' function to get the data for that record

$commentRecord = getComment($id);

//Assign the values from the database results to individual variables

$name = $commentRecord['name'];

$email = $commentRecord['email'];

$comment = $commentRecord['comment'];

//Finally show the 'edit' form populated with the data from the record

//Note that we use a 'hidden' input element to include the record id in the form

include("view/edit\_form.php");

}

elseif($action == 'Update Comment')

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

$id = filter\_input(INPUT\_POST, 'id');

//call the updateComment function and send the captured data as parameters

updateComment($id, $name, $email, $comment);

//Redirect the user back to the default view

header("Location:comments.php");

}

elseif($action == 'delete')

{

//User is trying to delete a secific comment.

//This request is sent via the 'GET' method when the user clicks the 'delete' link

//first capture the 'id' value of the record to be deleted

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'deleteComment' function to delete that record

deleteComment($id);

//Redirect the user back to the default view

header("Location:comments.php");

}

else

{

//default view: Shows the new comment form and the list of comments

//

$comments = getComments();

if($comments != NULL){

$commentList = "<h2>Comments Found:</h2>";

foreach($comments as $row){

$name = $row['name'];

$email = $row['email'];

$comment = $row['comment'];

$id = $row['id'];

$commentList .= "<p>Name: $name email: $email <a href = 'comments.php?action=edit&id=$id'>edit</a> | <a href = 'comments.php?action=delete&id=$id'>delete</a></p><p>$comment</p><hr>";

}

}

else

$commentList = "<h2> No comments found</h2>";

include("view/default\_view.php");

}

?>

Contains all the files to convert the comments.php application into an MVC application.

<?php

require("model/functions.php");

$action = filter\_input(INPUT\_POST, 'action');

if($action == NULL)

$action = filter\_input(INPUT\_GET, 'action');

if($action == "Submit Comment")

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

addComment($name, $email, $comment);

include("view/comment\_submitted.php");

}

elseif($action == 'edit')

{

//User is trying to edit and update a specific comment.

//This request is sent via the 'GET' method when the user clicks the 'edit' link

//first capture the 'id' value of the record to be updated

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'getComment' function to get the data for that record

$commentRecord = getComment($id);

//Assign the values from the database results to individual variables

$name = $commentRecord['name'];

$email = $commentRecord['email'];

$comment = $commentRecord['comment'];

//Finally show the 'edit' form populated with the data from the record

//Note that we use a 'hidden' input element to include the record id in the form

include("view/edit\_form.php");

}

elseif($action == 'Update Comment')

{

// Capture the data from the form

$name = filter\_input(INPUT\_POST, 'name');

$email = filter\_input(INPUT\_POST, 'email');

$comment= filter\_input(INPUT\_POST, 'comment');

$id = filter\_input(INPUT\_POST, 'id');

//call the updateComment function and send the captured data as parameters

updateComment($id, $name, $email, $comment);

//Redirect the user back to the default view

header("Location:comments.php");

}

elseif($action == 'delete')

{

//User is trying to delete a secific comment.

//This request is sent via the 'GET' method when the user clicks the 'delete' link

//first capture the 'id' value of the record to be deleted

$id = filter\_input(INPUT\_GET, 'id');

//Next call the 'deleteComment' function to delete that record

deleteComment($id);

//Redirect the user back to the default view

header("Location:comments.php");

}

else

{

//default view: Shows the new comment form and the list of comments

//

$comments = getComments();

if($comments != NULL){

$commentList = "<h2>Comments Found:</h2>";

foreach($comments as $row){

$name = $row['name'];

$email = $row['email'];

$comment = $row['comment'];

$id = $row['id'];

$commentList .= "<p>Name: $name email: $email <a href = 'comments.php?action=edit&id=$id'>edit</a> | <a href = 'comments.php?action=delete&id=$id'>delete</a></p><p>$comment</p><hr>";

}

}

else

$commentList = "<h2> No comments found</h2>";

include("view/default\_view.php");

}

?>

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Code example 3-1

SELECT vendorName, invoiceNumber, invoiceDate, invoiceTotal

FROM vendors INNER JOIN invoices

   ON vendors.vendorID = invoices.vendorID

WHERE invoiceTotal >= 500

ORDER BY vendorName DESC

When you code a DELETE statement, you usually need to include

1. an inner join
2. **Correct:**

a WHERE clause

Correct answer

1. a SORT BY clause
2. an outer join

**Question 2**

**2**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Each row in a table should be able to be uniquely identified by a:

1. cell
2. record
3. field name
4. **Correct:**

primary key

Correct answer

**Question 3**

**3**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

To execute a prepared SQL statement, you can use the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and execute() methods of the PDOStatement object to set parameter values and execute the statement.

1. fetch()
2. **Correct:**

bindValue()

Correct answer

1. exec()
2. elements()

**Question 4**

**4**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Code example 3-1

SELECT vendorName, invoiceNumber, invoiceDate, invoiceTotal

FROM vendors INNER JOIN invoices

   ON vendors.vendorID = invoices.vendorID

WHERE invoiceTotal >= 500

ORDER BY vendorName DESC

(Refer to code example 3-1.) How many columns will the result set have?

1. 7
2. 5
3. **Correct:**

4

Correct answer

1. 6

**Question 5**

**5**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

An advantage of using MySQL is:

1. **Correct:**

all of these

Correct answer

1. ease of use
2. cost
3. speed

**Question 6**

**6**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Which clause in a SQL statement specifies the columns to return?

1. FROM
2. **Correct:**

SELECT

Correct answer

1. ORDER BY
2. WHERE

**Question 7**

**7**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Which clause in a SQL statement specifies the records to return based on criteria?

1. SELECT
2. ORDER BY
3. **Correct:**

WHERE

Correct answer

1. FROM

**Question 8**

**8**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

The most common type of relationship between tables in a relational database is

1. one-to-one
2. many-to-many
3. **Correct:**

one-to-many

Correct answer

**Question 9**

**9**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Which type of SQL statement returns a result set?

1. INSERT
2. UPDATE
3. CREATE
4. **Correct:**

SELECT

Correct answer

**Question 10**

**10**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Code example 3-1

SELECT vendorName, invoiceNumber, invoiceDate, invoiceTotal

FROM vendors INNER JOIN invoices

   ON vendors.vendorID = invoices.vendorID

WHERE invoiceTotal >= 500

ORDER BY vendorName DESC

When you code an INSERT statement, you don’t have to include the data for a column that

1. **Correct:**

has a default value

Correct answer

1. is a foreign key
2. does not allow null values
3. is a primary key

**Question 11**

**11**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Which of the following can be used to get the data from an array that contains all rows of a result set?

1. **Correct:**

foreach statement

Correct answer

1. roll over
2. fetch() method
3. forall loop

**Question 12**

**12**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

Which PHP class can be used to connect to a MySQL database?

1. **Correct:**

PDO

Correct answer

1. argument
2. DataConnect
3. db

**Question 13**

**13**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

The result set retrieved by the following SELECT statement contains rows that have

SELECT balance, number

FROM accounts

WHERE balance < 0

1. two of the rows from the account table
2. **Correct:**

two of the columns from the accounts table where balance is less than 0

Correct answer

1. all of the columns from the accounts table where balance is less than 0
2. all of the columns from the accounts table

**Question 14**

**14**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

An index for a PHP array:

1. does not exist
2. must be a string
3. must be a number
4. **Correct:**

can be a number or a string

Correct answer

**Question 15**

**15**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

To get an array for all rows in a result set, you can call the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ method of the PDOStatement object for the result set.

1. get()
2. fetch()
3. getAll()
4. **Correct:**

fetchAll()

Correct answer

**Question 16**

**16**

**Multiple Choice**

**CORRECT**

**1/1**

**Grade: 1 out of 1 point possible**

To return an array for the first row of a result set that’s returned by a SELECT statement, you use the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ method of the PDOStatement object for the result set.

1. get()
2. fetchFirst()
3. **Correct:**

fetch()

Correct answer

1. getFirst()