

Accessing MySQL Using PHP

COMP-3077 Web-Based Data Management

Dr. Saja AL-Mamoori

Many to Many Relation

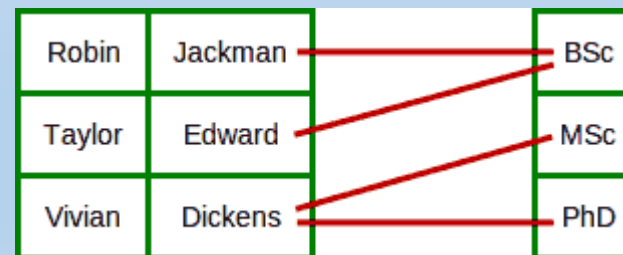
id	name
1	BSc
2	MSc
3	PhD

Education

id	first_name	last_name	job_title	salary
1	Robin	Jackman	Software Engineer	5500
2	Taylor	Edward	Software Architect	7200
3	Vivian	Dickens	Database Administrator	6000

Employee

Robin	Jackman		BSc
Taylor	Edward		MSc
Vivian	Dickens		PhD



Storing Many to Many Relation

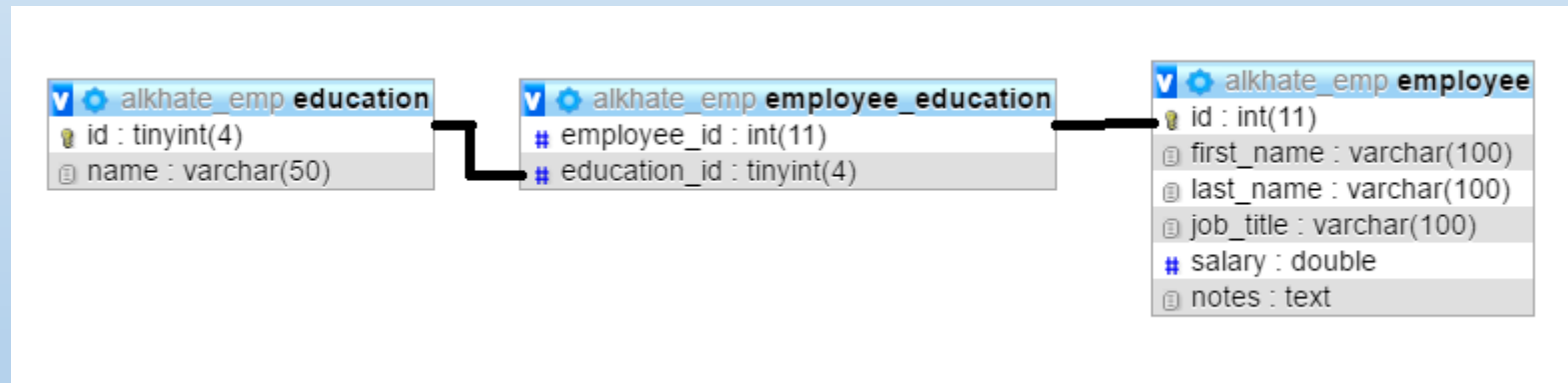
- For storing many-to-many relationships, we need an intermediate table that mainly stores the primary keys (IDs) of each relationship. In this case, we can use a table (`employee_education`) like below.

employee_id	education_id
1	1
2	1
3	2
3	3

Employee_Education

Implementing Many to Many Relation

- The example and code can be found:
- <http://www.phpknowhow.com/extra/mysql-example-tables/>



Dealing with checkboxes-send_checkbox.php

```
<form method="get" action="rec_chexbox.php">
<label for="interests">
    <input type="checkbox" name="colors[]" value="red">Red
<br> </label>
<label for="interests">
    <input type="checkbox" name="colors[]" value="blue">Blue
<br> </label>
<input type="submit" value="submit"/>
</form>
```

Dealing with checkboxes-rec_checkbox.php

<?php // of course you can do same with POST

\$_GET['colors']; // to receive the values of colors[]

foreach(\$_GET['colors'] as \$color) {

 echo \$color;

}

?>

Check / Uncheck All

```
<html> <head>
<script>
function check_them_all(){
var colors = document.forms[0];
var i;
if (colors[0].checked)
for (i = 0; i < colors.length; i++)
    colors[i].checked=true;
else
for (i = 0; i < colors.length; i++)
    colors[i].checked=false;
}
</script> </head>
```

```
<body>
<form method ="get"
action="rec_chexbox.php">
<label for="all">
    <input type="checkbox" name="all"
value="all" onclick="check_them_all()">All
<br> </label>
<label for="interests">
...
..
</form>
</body>
</html>
```

Querying a MySQL Database with PHP

- The reason for using PHP as an interface to MySQL is to format the results of querying the database.
- However, instead of using MySQL's command line to enter instructions and view output, you will create query strings that are passed to MySQL.
- SQL queries in a form visible in a web page.
- When MySQL returns its response, it will come as a data structure that PHP can recognize

The process of using MySQL with PHP

- 1) Connect to MySQL server and select the database to use.
- 2) Build a query string.
- 3) Perform the query.
- 4) Retrieve the results and output them to a web page.
- 5) Repeat steps 2 to 4 until all desired data has been retrieved.
- 6) Disconnect from MySQL.

Creating a Login File

- Common practice, (reusable login file shared by many webpages)
- *The login.php file:*

```
<?php // login.php
    $hn = 'localhost'; //hostname
    $db = 'publications'; //database
    $un = 'username'; //username
    $pw = 'password'; //password
?>
```

Connecting to MySQL

you can include it in any PHP files that will need to access the database by using the **require_once** statement

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

- This example creates a new object called \$conn by calling a new instance of the mysqli method, passing all the values retrieved from the login.php file.
- Error checking is achieved by referencing the \$conn->connect_error property.
- The -> operator indicates that the item on the right is a property or method of the object on the left. If \$conn->connect_error has a value, die the page

Connecting to MySQL

- The die function is great for when you are developing PHP code.
- Of course you will want more **user-friendly** error messages on a production server.

```
function mysql_fatal_error($msg) {  
    $msg2 = mysql_error();  
    echo <<< _END
```

We are sorry, but it was not possible to complete the requested task. The error message we got was:

```
<p>$msg: $msg2</p>
```

Please click the back button on your browser and try again. If you are still having problems, please email our administrator. Thank you.

```
_END;  
}
```

Building and executing a query

```
<? php //..... previous code
$query = "SELECT * FROM classics";
$result = $conn->query($query);
if (!$result) die($conn->error);
?>
```

- query performs a query on the database
- Unlike with MySQL's command line, no semicolon is required at the tail of the query, because the **query** function is used to issue a complete query;
- If \$result is FALSE, there was a problem and the error property of the connection object (\$conn) will contain the details

Fetching a result

- `<?php //..... previous code`
`$rows = $result->num_rows;`
`for ($j = 0 ; $j < $rows ; ++$j) {`
`$result->data_seek($j);`
`echo 'Author: ' . $result->fetch_assoc()['author'] . '
;`
`$result->data_seek($j);`
`echo 'Title: ' . $result->fetch_assoc()['title'] . '
;`
`$result->data_seek($j);`
`echo 'Category: ' . $result->fetch_assoc()['category'] . '
;`
`$result->data_seek($j);`
`echo 'Year: ' . $result->fetch_assoc()['year'] . '
;`
`$result->data_seek($j);`
`echo 'ISBN: ' . $result->fetch_assoc()['isbn'] . '

; }`
`?>`

Fetching a result

- `num_rows` function reports the number of rows returned by a query.
- Armed with the row count, we enter a for loop that extracts each cell of data from each row using the `fetch_assoc` function.
- `data_seek` is mandatory to adjust the pointer to a specific row (`$j`).
- The parameters supplied to `fetch_assoc` is the name of the column from which to extract the data.

Closing a connection

- When you have finished using a database, you should close the connection.

```
<?php // ..... previous code
```

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

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

```
?>
```

- PHP will eventually return the memory it has allocated for objects after you have finished with the script, so in small scripts, you don't usually need to worry about releasing memory yourself.
- However, you should release the memory in higher-traffic page.

Creating a Table

```
<?php require_once 'login.php';  
$conn = new mysqli($hn, $un, $pw, $db);  
if ($conn->connect_error)  
    die($conn->connect_error);  
$query = "CREATE TABLE cats (  
id SMALLINT NOT NULL AUTO_INCREMENT, family VARCHAR(32) NOT NULL,  
name VARCHAR(32) NOT NULL, age TINYINT NOT NULL, PRIMARY KEY (id)  
)";  
$result = $conn->query($query);  
if (!$result)  
die ("Database access failed: " . $conn->error); ?>
```

Adding Data

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error)
    die($conn->connect_error);
$query = "INSERT INTO cats VALUES(NULL, 'Lion', 'Leo', 4) ";
$result = $conn->query($query);
if (!$result)
    die ("Database access failed: " . $conn->error); ?>
```

Adding Data

- You may wish to add a couple more items of data by modifying \$query as follows and calling up the program in your browser again:
- \$query = "INSERT INTO cats VALUES(NULL, 'Cougar', 'Growler', 2)";
- \$query = "INSERT INTO cats VALUES(NULL, 'Cheetah', 'Charly', 3)";
- Notice the NULL value passed as the first parameter? This is because the id column is of type AUTO_INCREMENT, and MySQL will decide what value to assign according to the next available number in sequence, so we simply pass a NULL value, which will be ignored.

Updating Data

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error)
    die($conn->connect_error);
$query = "UPDATE cats SET name='Charlie' WHERE name='Charly' " ;
$result = $conn->query($query);
if (!$result)
    die ("Database access failed: " . $conn->error); ?>
```

Deleting Data

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error)
    die($conn->connect_error);
$query = "DELETE FROM cats WHERE name='Growler'";
$result = $conn->query($query);
if (!$result)
    die ("Database access failed: " . $conn->error); ?>
```

Describing a Table

```
<?php require_once 'login.php';  
$conn = new mysqli($hn, $un, $pw, $db);  
if ($conn->connect_error)  
die($conn->connect_error);  
$query = "DESCRIBE cats";  
$result = $conn->query($query);  
if (!$result)  
die ("Database access failed: " . $conn->error);  
$rows = $result->num_rows;
```

Describing a Table

```
echo "<table><tr><th>Column</th><th>Type</th><th>Null</th><th>Key</th></tr>";  
for ($j = 0 ; $j < $rows ; ++$j) {  
    $result->data_seek($j);  
    $row = $result->fetch_array(MYSQLI_NUM);  
    echo "<tr>";  
        for ($k = 0 ; $k < 4 ; ++$k)  
            echo "<td>$row[$k]</td>";  
        echo "</tr>"; }  
echo "</table>";  
?>
```

Fetch_assoc() access column (data cell) by name, \$row[\$k] is an example of accessing the column(data cell) by index

Dropping Table (not recommended)

```
<?php
require_once 'login.php';
$conn = new mysqli($hn, $un, $pw, $db);
if ($conn->connect_error)
    die($conn->connect_error);
$query = "DROP TABLE cats";
$result = $conn->query($query);
if (!$result)
    die ("Database access failed: " . $conn->error); ?>
```


A Practical Example

- A comprehensive example to add into, delete from, and query a table.
- It will use the file that calls itself as a client/server, which you've learned earlier.
- A form has a textboxes to be filled by user. The values entered by user will be inserted into classics table.
- A delete button for each record. Once the user clicks on it, that specific record will be deleted from classics table.
- isset & \$_POST functions will be used to determine whether it is first or second visit, if second , it will determine what kind of transactions (insert or delete) the user want to do.

A Practical Example – Establishing connection

```
<?php // sqltest.php
    require_once 'login.php';
    $conn = new mysqli($hn, $un, $pw, $db);
    if ($conn->connect_error)
        die($conn->connect_error);
```

Checking on 2nd visit - Deletion

```
if (isset($_POST['delete']) && isset($_POST['isbn'])) {  
    $isbn = get_post($conn, 'isbn');  
    $query = "DELETE FROM classics WHERE isbn='$isbn'";  
    $result = $conn->query($query);  
    if (!$result)  
        echo "DELETE failed: $query<br>" . $conn->error . "<br><br>";  
}
```

isset: function that checks whether value for a sent element(parameter) have been posted to the program.

Checking on 2nd visit – Insertion

```
if (isset($_POST['author']) && isset($_POST['title']) &&  
isset($_POST['category']) && isset($_POST['year']) &&  
isset($_POST['isbn'])) {
```

```
    $author = get_post($connection, 'author');
```

```
    $title = get_post($connection, 'title');
```

```
    $category = get_post($connection, 'category');
```

```
    $year = get_post($connection, 'year');
```

```
    $isbn = get_post($connection, 'isbn');
```

Checking on 2nd visit – Insertion (cont..)

```
$query = "INSERT INTO classics VALUES".("'$author', '$title', '$category', '$year',  
'$isbn'");  
$result = $conn->query($query);  
if (!$result)  
    echo "INSERT failed: $query<br>" . $conn->error . "<br><br>";  
}
```

Creating form – Both visits

```
echo <<<_END
```

```
<form action=" sqltest.php" method="post">
```

```
<pre> Author <input type="text" name="author">
```

```
Title <input type="text" name="title">
```

```
Category <input type="text" name="category">
```

```
Year <input type="text" name="year">
```

```
ISBN <input type="text" name="isbn">
```

```
<input type="submit" value="ADD RECORD"> </pre></form>
```

```
_END;
```

Showing rows + deleting rows

```
$query = "SELECT * FROM classics";  
$result = $conn->query($query);  
if (!$result)  
die ("Database access failed: " . $conn->error());  
$rows = $result->num_rows;  
for ($j = 0 ; $j < $rows ; ++$j) {  
    $result->data_seek($j);  
    $row = $result->fetch_array(MYSQLI_NUM);  
    echo <<<_END  
<pre>
```

```
Author $row[0]  
Title $row[1]  
Category $row[2]  
Year $row[3]  
    ISBN $row[4]
```

```
</pre>
```

```
<form action="sqltest.php" method="post"> <input  
type="hidden" name="delete" value="yes">
```

```
<input type="hidden" name="isbn" value="$row[4]"
```

```
<input type="submit" value="DELETE RECORD">
```

```
</form>
```

```
_END; }
```

Closing link & implementing get_post

```
$result->close();  
$conn->close();  
function get_post($conn, $var) {  
    return $conn->real_escape_string($_POST[$var]); }  
?>
```

mysql_real_escape_string

Escapes special characters in the unescaped_string, taking into account the current character set of the connection so that it is safe to place it in a mysql_query().

It is deprecated in php 5.5 and removed from php 7

Author	
Title	
Category	
Year	
ISBN	
ADD RECORD	

Author Mark Twain
Title The Adventures of Tom Sawyer
Category Fiction
Year 1876
ISBN 9781598184891

DELETE RECORD

Author Jane Austen
Title Pride and Prejudice
Category Fiction
Year 1811
ISBN 9780582506206

DELETE RECORD

Author Charles Darwin
Title The Origin of Species
Category Non-Fiction
Year 1856
ISBN 9780517123201

DELETE RECORD

Author Charles Dickens
Title The Old Curiosity Shop
Category Fiction
Year 1841
ISBN 9780099533474

DELETE RECORD

Preventing SQL Injection

- suppose you have a simple piece of code to verify a user, and it looks like this:

```
$user = $_POST['user'];
```

```
$pass = $_POST['pass'];
```

```
$query = "SELECT * FROM users WHERE user='$user' AND pass='$pass'";
```

Bad scenario, if users enters : **anything' OR 1=1 #**

→

```
$query = "SELECT * FROM users
```

```
WHERE user='anything' OR 1=1 # AND pass='$pass'";
```

in MySQL is start comment sign

***mysql_real_escape_string** can prevent special characters in string, but can not prevent logical injection(smart hacking)*

Using Placeholders

- Another way—this one virtually bulletproof—to prevent SQL injections is to use a feature called placeholders.
- The idea is to **predefine** a query using ? characters where the data will appear. Then, instead of calling a MySQL query directly, you call the predefined one, passing the data to it.
- This has the effect of ensuring that every item of data entered cannot be interpreted as SQL queries.

Using Placeholders

- **?** Is the place holder. Which goes to **prepare** the query first, then **execute** it
- “**?**” In “prepare” statements, prevents any part of the placeholder from changing the query, by default, it has the quotes for string, so anything between the quotes can not change the query.
- ```
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
if ($stmt->execute(array($_GET['name']))) {
 while ($row = $stmt->fetch()) {
 print_r($row);
 }
}
?>
```
- The query in prepare statement can be used multiple times to enter different values each time

## References

- Learning PHP, MySQL & JavaScript, 4th Edition. With jQuery, CSS & HTML5, by Robin Nixon, O'Reilly Media, 2014, 978-1-4919-1866-1, chapter 10