Programação Web Servidor

Desenvolvimento Web e Multimédia, 1º ano – 2º semestre

MySQL Improved (MySQLi)

POLITÉCNICO DE LEIRIA
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Copyright

Author(s):

- **Vítor Carreira**

Contributor(s):

- **Ana Nogueira**
- **João Real**
- **Michael Pinheiro**
- **Norberto Henriques**
- **Telmo Marques**
- **Alexandrino Gonçalves**

Revised on: 04/05/2023

Database Integration

- PHP has native connections available to many database systems: MySQL, MariaDB, PostgreSQL, SYBASE, Oracle, IBM-DB2, filePro, Mongo, Informix, etc.
- Other connectivity options:
  - SQLite;
  - Open Database Connectivity (ODBC) - allow connectivity to any database that provides an ODBC driver;
  - PHP Data Objects (PDO) - database access abstraction layer which allows consistent access and promotes secure coding practices.

MySQL

- Since its inception, MySQL and PHP have long enjoyed a close relationship.
- MySQL main features:
  - Portability - can be used on different operating systems;
  - Secure;
  - Scalable;
  - High performance;
  - Ease of configuration and use;
  - Low hardware requirements;
  - Low cost (free open-source license).

PHP MySQLi

- MySQLi Extension (MySQL Improved) is a relational database driver used in PHP:
  - Object oriented interface - the mysqli extension is encapsulated into a class;
  - Prepared statements;
  - Enhanced debugging capabilities;
  - Support for multiple statements;
  - Support for transactions.

Querying a MySQL Database

- The basic steps for querying a MySQL database are:

1. Setup a connection
2. Execute the query
3. Retrieve the results
4. Release resources

Setup a connection

- *mysqli* constructor creates a new connection to host *$host* with username *$user* and password *$pass*. The connection is setup to use database *$database*

```
$host="localhost";
$database="pws";
$user="root";
$pass="";
$conn = new mysqli($host, $user, $pass, $database);
if ($conn->connect_errno) {
   echo "Failed to connect to MySQL: ".$conn->connect_error;
   exit();
}
```

Execute the query 1/2

- `mysqli->query($query)`
  - Executes the query *$query*. Returns a pointer to the result set in case of success; *null* in case of error.

```
$query = "select * from books";
$result_set = $conn->query($query);
if ($result_set) {
  // Iterates the result
} else {
  $code = $conn->errno; // error code of the most recent operation
  $message = $conn->error; // error message of the most recent op.
  printf("<p>Query error: %d %s</p>", $code, $message);
}
```

Execute the query 2/2

- mysqli_result->num_rows

```
$query = "select * from books";
$result_set = $conn->query($query);
if ($result_set) {
   echo "Number of books: ".$result_set->num_rows;
}
```

Retrieve the results 1/3

- `mysqli_result->fetch_row()`
  - Iterates the result set as an enumerated array. Each index matches a column from the query;
  - The method returns false when no more rows are available.

```
while ($row = $result_set->fetch_row()) {
    echo $row[0]."<br>".$row[1];
}
```

Retrieve the results 2/3

- `mysqli_result->fetch_assoc()`
  - Iterates the results as an associative array. The column's names are used as keys;
  - The method returns false when no more rows are available.

```
while ($row = $result_set->fetch_assoc()) {
      echo $row['author']."<br>".$row['title'];
}
```

Retrieve the results 3/3

- `mysqli_result->fetch_object()`
  - Iterates the results as an array of objects. The column's name are used as properties of the object;
  - The method returns false when no more rows are available.

```
while ($row = $result_set->fetch_object()) {
      echo $row->author."<br>".$row->title;
}
```

Free resources

- `mysqli_result->free()`
  - Restore any memory consumed by a result set.

```
$result_set->free();
```

- `mysqli->close()`
  - Databases have a maximum number of concurrent connections. If the connection is no longer needed, it SHOULD be closed.

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

Insert/Update/Delete 1/3

- `mysqli->query($query)`
  - The parameter *$query* can be an INSERT, UPDATE or DELETE SQL statement. The method returns true in case of success; false otherwise.

```
$query = "insert into books (isbn, author, title, price) values
   ('".$isbn."','".$author."','".$title."','".$price."')";
$result = $conn->query($query);
if ($result) {
  echo "Number of inserted rows: ".$conn->affected_rows;
}
```

Insert/Update/Delete 2/3

- INSERT (another version)

```
$isbn = "'".$isbn."'";
$author = "'".$author."'";
$title = "'".$title."'";
$price = "'".$price."'";
$query="insert into books values($isbn, $author, $title,
            $price)";
$result = $conn->query($query);
if ($result) {
  echo "Number of inserted rows: ".$conn->affected_rows;
}
```

Insert/Update/Delete 3/3

- `mysqli->affected_rows`
  - Property that returns the number of rows affected by the INSERT, UPDATE or DELETE statement.

```
$query = "update books set price=10 where price=1";
$result = $conn->query($query);
if ($result) {
  echo "Number of updated rows: ".$conn->affected_rows;
}
```

```
$query = "delete books where price<1";
$result = $conn->query($query);
if ($result) {
  echo "Number of deleted rows: ".$conn->affected_rows;
}
```

Prepared statements 1/9

- Useful for speeding up execution when performing large numbers of the same query with different data.
- Provides protection against SQL injection-style attacks.
- Basic steps:
  1. Prepare the statement
  2. For each iteration (one or more)
     (i) Bind parameters
     (ii) Execute the statement
     (iii) Bind and fetch results
  3. Release the prepared statement

Prepared statements 2/9

- `mysqli->prepare($query)`
  - Returns a new prepared statement object for query *$query* and returns false in case of error.
  - The query can include one or more parameter markers in the SQL statement by embedding question mark (?) characters at the appropriate positions.

```
$query = "insert into books (isbn, author, title,
price) values (?, ?, ?, ?)";
$stmt = $conn->prepare($query);
if (!stmt) {
  $code = $conn->errno;
  $message = $conn->error;
  printf("<p>SQL Error: %d %s</p>", $code, $message);}
```

Prepared statements 3/9

- `mysqli_stmt->bind_param($format, $var_list)`
  - The first parameter is a format string, where each character represents the data type of each parameter: d (doubles), i (integers), b (blobs) and s (all other types including strings).
  - After the first parameter, comes the list of variables that should be substituted for the question marks.

```
$query = "insert into books (isbn, author, title,
price) values(?, ?, ?, ?)";
$stmt = $conn->prepare($query);
$stmt->bind_param("sssd",$isbn, $author, $title,
      $price);
```

Prepared statements 4/9

- `mysqli_stmt->execute()`
  - Executes the prepared statement. Returns true in case of success; false otherwise

```
$query = "insert into books (isbn, author, title, price)
values(?, ?, ?, ?)";
$stmt = $conn->prepare($query);
$stmt->bind_param("sssd", $isbn, $author, $title, $price);
if ($stmt->execute()) {
   // Iterate the result set (for SELECT queries)
} else {
  $code = $stmt->errno; // error code for the execute op
  $message = $stmt->error; // error message for the execute op
  printf("<p>Execution error: %d %s</p>", $code, $message);
}
```

Prepared statements 5/9

- `mysqli_stmt->num_rows`
  - Returns the number of rows after executing the prepared statement.
    - Fetching all results to the web server has a sever impact on memory consumption.

```
$query = "select isbn, title, author, price from
            books where price < ?";
$stmt = $conn->prepare($query);
$stmt->bind_param("d", $price);
if ($stmt->execute()) {
  $stmt->store_result();
  $num_rows = $stmt->num_rows;
}
```

Prepared statements 6/9

- `mysqli_stmt->bind_result($var_list)`
  - The method binds variables to the retrieved fields.

```
$query = "select isbn, title, author, price from books
                where price < ?";
$stmt = $conn->prepare($query);
$stmt->bind_param("d", $price);
if ($stmt->execute()) {
  $stmt->bind_result($isbn, $title, $author, $price);
  // Iterate each row in the result set
}
```

Prepared statements 7/9

- `mysqli_stmt->fetch()`
  - Retrieves each row from the prepared statement result and assigns the fields to the bound results.

```
$query = "select isbn, title, author, price from books
            where price < ?";
$stmt = $conn->prepare($query);
$stmt->bind_param("d", $price);
if ($stmt->execute()) {
  $stmt->bind_result($isbn, $title, $author, $price);
  while ($stmt->fetch()) {
    echo $isbn." ".$title." ".$author." ".$price;
    }
}
```

Prepared statements 8/9

- `mysqli_stmt->free_result()`
  - Restores the memory consumed by the statement.

```
$query = "select isbn, title, author, price from books
              where price < ?";
$stmt = $conn->prepare($query);
$stmt->bind_param("d", $price);
if ($stmt->execute()) {
  $stmt->bind_result($isbn, $title, $author, $price);
  while ($stmt->fetch()) {
     printf("<p>%s, %s, %s, %d</p>",
                $isbn, $title, $author, $price);
  }
  $stmt->free_result();
}
```

Prepared statements 9/9

- `mysqli_stmt->close()`
  - Closes the prepared statement and deallocates the statement handle.

```
$query = "select isbn, title, author, price from books
            where price < ?";
$stmt = $conn->prepare($query);
$stmt->bind_param("d", $price);
if ($stmt->execute()) {
  $stmt->bind_result($isbn, $title, $author, $price);
  while ($stmt->fetch()) {
     printf("<p>%s, %s, %s, %d</p>",
                $isbn, $title, $author, $price);
  }
  $stmt->free_result();
}
$stmt->close();
```

Handling special chars 1/5

- `mysqli->real_escape_string($string)`
  - Some characters have a special meaning for SQL and if present can cause a syntax error or even inject dangerous behavior (SQL Injection).
  - This method escapes special characters in a string *$string*, considering the current charset of the connection.

```
$city = "'s Hertogenbosch";
/* this query will fail, because we didn't escape $city */
if (!$conn->query("INSERT into City (Name) VALUES ('$city')")){
      printf("Error: %s", $conn->error);
}
```

Handling special chars 2/5

- `mysqli->real_escape_string($string)`

```
$city = "'); DELETE FROM City WHERE 1 or Name = ('";
$query = "INSERT into City (Name) VALUES ('$city')";
/* this query will have catastrophic consequences */
if (!$conn->multi_query($query)) {
    printf("Error: %s", $conn->error);
}
```

Handling special chars 3/5

- `mysqli->real_escape_string($string)`

```
$city = "'s Hertogenbosch";
$city = $conn->real_escape_string($city);

if (!$conn->query("INSERT into City (Name) VALUES ('$city')")){
      printf("Error: %s", $conn->error);
}
```

Note: With prepared statements there's no need to escape strings

Handling special chars 4/5

- `htmlspecialchars($string)`
  - Some characters have a special meaning for HTML and if present can cause a validation error or even inject dangerous behavior (XSS – Cross Site Scripting);
  - This method encode characters present in *$string* that have a special meanings in HTML (&, <, >, ‘, “).

```
$title = "PHP & MySql";
echo "<p>".$title."</p>";
```

HTML validation failure

```
$title = "PHP & MySql";
$title = htmlspecialchars($title);
echo "<p>".$title."</p>";
```

OK

Handling special chars 5/5

- `htmlspecialchars($string)`
  - When displaying textual information retrieved from a database, always call `htmlspecialchars`.

```
$blog_comment_field_from_db =
  "<script>
  document.location='http://someevilpage.com';
  </script>";

echo "<p>".$blog_comment_field_from_db."</p>";
```

EPIC FAIL - XSS (Cross Site Scripting)

Database & Security 1/2

1. Put database access credentials in a file only accessible locally (as a failsafe the file should have extension .php).

dbconnect.php

```
<?php
   $db_server = 'localhost';
   $db_user_name = 'bob';
   $db_password = 'secret';
   $db_name = 'somedb';
?>
```

```
<?php
   require('../code/dbconnect.php);
   @ $conn = new mysqli(
       $db_server, $db_user_name, $db_password, $db_name);
?>
```

Database & Security 2/2

2. Don’t grant DROP, ALTER, GRANT privileges to the user used to access the database;
3. Use prepared statements or call `real_escape_string` for each string used inside a query (SQL Injection);
4. Use prepared statements or call `floatval`, `intval` for numeric values used inside a query (SQL Injection);
5. Call `htmlspecialchars` for each string fetched from a database and before presenting it to the user (XSS);
6. Encrypt sensitive information such as passwords before storing it.

```
<?php
  $password = hash('sha512', $plaintext_password);
?>
```

References

- Robin Nixon, "Learning PHP; MySQL and JavaScript: With jQuery; CSS and HTML5", 5th. Edition. O'Reilly, 2018

- PHP Documentation
  - http://www.php.net/manual/en/mysqli.overview.php
  - http://www.php.net/manual/en/refs.database.vendors.php