PHP MySQL Prepared Statements

❮ Previous Next ❯

Prepared statements are very useful against SQL injections.

Prepared Statements and Bound Parameters

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

1. Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)
2. The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
3. Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values

Compared to executing SQL statements directly, prepared statements have three main advantages:

- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

The following example uses prepared statements and bound parameters in MySQLi:

Example (MySQLi with Prepared Statements)

Get your own PHP Server

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

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

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";
```

Code lines to explain from the example above:

```
"INSERT INTO MyGuests (firstname, lastname, email) VALUES (?, ?, ?)"
```

In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double or blob value.

Then, have a look at the bind_param() function:

```
$stmt->bind_param("sss", $firstname, $lastname, $email);
```

This function binds the parameters to the SQL query and tells the database what the parameters are. The "sss" argument lists the types of data that the parameters are. The s character tells mysql that the parameter is a string.

The argument may be one of four types:

- i - integer
- d - double
- s - string
- b - BLOB

We must have one of these for each parameter.

By telling mysql what type of data to expect, we minimize the risk of SQL injections.

Note: If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

Example (PDO with Prepared Statements)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // prepare sql and bind parameters
  $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES (:firstname, :lastname, :email)");
  $stmt->bindParam(':firstname', $firstname);
  $stmt->bindParam(':lastname', $lastname);
  $stmt->bindParam(':email', $email);

  // insert a row
  $firstname = "John";
  $lastname = "Doe";
  $email = "john@example.com";
  $stmt->execute();

  // insert another row
  $firstname = "Mary";
  $lastname = "Moe";
  $email = "mary@example.com";
  $stmt->execute();

  // insert another row
  $firstname = "Julie";
  $lastname = "Dooley";
  $email = "julie@example.com";
  $stmt->execute();

  echo "New records created successfully";
```

```
?>
```

COLOR PICKER

ADVERTISEMENT

ADVERTISEMENT

ADVERTISEMENT

Top Tutorials

HTML Tutorial
CSS Tutorial
JavaScript Tutorial
How To Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial
PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
Angular Reference
jQuery Reference

Top Examples

HTML Examples
CSS Examples
JavaScript Examples
How To Examples
SQL Examples
Python Examples
W3.CSS Examples
Bootstrap Examples
PHP Examples
Java Examples
XML Examples
jQuery Examples

Get Certified

HTML Certificate
CSS Certificate
JavaScript Certificate
Front End Certificate
SQL Certificate
Python Certificate
PHP Certificate
jQuery Certificate
Java Certificate
C++ Certificate
C# Certificate
XML Certificate