SUPPLEMENTAL CHAPTER 8

What's New in PHP 8 and MySQL 8

By the end of the second decade of the 21st century, both PHP and MySQL had matured into their eighth versions and can now be considered highly mature products by software technology standards.

According to the w3techs.com website, by early 2021 PHP was used in one way or another by over 79% of all websites, a massive 70% ahead of its nearest rival, ASP.NET. explore-group.com reported that in 2019 MySQL remained the most popular database in use on the web, installed on 52% of websites, while in 2022 Stack Overflow showed the product still had a 47% market share. So, although MySQL has slipped a little in recent years, it has still remained ahead of its nearest competitor, PostgreSQL, which is used on around 40% of websites. It appears this will continue to be the case in the foreseeable future, especially as the almost identical MariaDB also lays claim to about 18% of the market, as of the time of writing, owning around 65% of the market between them.

With the latest version 8 releases of these two technologies, along with JavaScript, these mainstays of modern web development appear set to remain important to web development for many years to come. So let's take a look at what's new in the latest versions of PHP and MySQL.

About This Chapter

This short chapter is more of a summary or update to the main content of this book, detailing some of the more advanced improvements that have been made to PHP and MySQL.

If any of the following is a topic you have not yet covered in this book or elsewhere, don't worry; you don't (yet) need it in your development toolkit. But with those you

have seen before, you'll understand what's been added or changed and have a hint of how to make use of it.

PHP 8

Version 8 of PHP marked a major update and a huge milestone, bringing a number of new features to the following: type, system, syntax error handling, strings, object-oriented programming, and more. Since it's release there have been two further minor updates (as of the time of writing), with the latest version of PHP being 8.2.

The new features make PHP more powerful and easy to use than ever while minimizing changes that could break or modify existing installations.

For example, Named Parameters, Just In Time (JIT) compilation, Attributes, and Constructor Properties bring major improvements and syntax changes, improved error handling, and changes and improvements in operators to help reduce the chances of overlooked bugs.

PHP8 and the AMPPS Stack

At the time of writing, the free version of the AMPPS stack, which you were recommended to install in Chapter 2, comes with MySQL 8.0.31 and PHP 7.4. Newer versions 8.0.29 and 8.1.20 of PHP (along with earlier ones) can be installed by becoming a registered user for a small fee. During the course of publication of this book, it is anticipated that version 8.2 of PHP will also become available in the AMPPS stack. Remember that you can also download other versions of PHP for free directly from the php.net website, following the instructions provided there to install and use them.

Named Parameters

In addition to traditional positional parameters, in which the order in which arguments are passed must always be the same, PHP 8 allows named parameters in function calls, like this:

```
str_contains(needle: 'ian', haystack: 'Antidisestablishmentarianism');
```

This makes the function (or method) parameter names a part of the public API and allows you to pass input data into a function, based on their argument names, instead of the argument order, substantially increasing code clarity. Therefore, because the parameters are passed by name, their order is no longer of importance, which should lead to fewer difficult-to-find bugs. Incidentally, `str_contains` (explained later) is also new to PHP 8.

Attributes

In PHP 8, attributes (such as properties and variables) map to PHP class names, allowing the inclusion of metadata in classes, methods, and variables. Previously you would have had to use DocBloc comments to infer them.

Attributes are a way to give one system or program details of another system, such as a plug-in or an event system, and are simple classes annotated with the `#[Attribute]` attribute, which can be attached to classes, properties, methods, functions, class constants, or function/method parameters.

At runtime attributes do nothing and have no impact on the code. However, they are available to the reflection API, which allows other code to examine the attribute and take additional actions.

You can find a good explanation of attributes (which are somewhat out of the scope of this book) in the PHP.net overview.

As long as they are on a single line, attributes are interpreted as comments in older PHP versions and therefore are safely ignored.

Constructor Properties

With PHP 8 you can now declare class properties right from the class constructor, saving a great deal of boilerplate coding. Take this code, for example:

```
class newRecord
{
    public string $username;
    public string $email;

     public function __construct(
        string $username,
        string $email,
    ) {
        $this->username = $username;
        $this->email    = $email;
    }
}
```

With constructor properties you can now reduce all of that to the following:

```
class newRecord
{
     public function __construct(
        public string $username,
        public string $email,
    ){}
}
```

This is a backward-incompatible feature, so only use it where you know for sure that PHP 8 is installed.

Just-In-Time Compilation

When enabled, Just In Time (JIT) compiles and caches native instructions (as opposed to what's known as the OPcache, which saves file parsing time) to provide a performance boost to CPU-heavy applications.

You can enable JIT in the *php.ini* file like this:

```
opcache.enable          = 1
opcache.jit_buffer_size = 100M
opcache.jit             = tracing
```

Bear in mind that JIT is relatively new to PHP and that it currently makes debugging and profiling harder with the added layer. Also, there were issues with JIT right up to the day before initial release, so be wary that there may remain some undiscovered bugs in the system for a short period until they are ironed out. Therefore, JIT compilation is disabled by default, and you should probably keep it disabled while debugging, just in case.

Union Types

In PHP 8, type declarations can be extended with Union Types to declare more than one type (which also supports `false` as a special type for Boolean `false`) like this:

```
function parse_value(string|int|float): string|null {}
```

Null-safe Operator

In the following, the `?->` null-safe operator will short-circuit the remainder of the section if it encounters a `null` value and will return a `null` immediately without causing an error:

```
return $user->getAddress()?->getCountry()?->isoCode;
```

Previously you would have needed to use several sequential calls to `isset()` for each section, testing them each in turn for having a `null` value.

match Expressions

A `match` expression is like a `switch` block, but it provides type-safe comparisons, supports a return value, does not require a `break` statement to break out, and also supports multiple matching values. So this rather cumbersome `switch` block:

```
switch ($country)
{
  case "UK":
```

```
    case "USA":
    case "Australia":
    default:
        $lang = "English";
        break;
    case "Spain":
        $lang = "Spanish";
        break;
    case "Germany":
    case "Austria":
        $lang = "German";
        break;
}
```

can be replaced with the following much simpler `match` expression:

```
$lang = match($country)
{
  "UK", "USA", "Australia", default => "English",
  "Spain"                           => "Spanish",
  "Germany", "Austria"              => "German",
};
```

Regarding type-safe comparisons, a `switch` statement will get quite confused by the following code, because it will evaluate `'0e0'` as being an exponential value for zero and will echo `'null'`, when it ought to echo `'a'`. However, `match` will not have this issue.

```
$a = '0e0';

switch ($a)
{
  case    0 : echo 'null'; break;
  case '0e0': echo 'a';    break;
}
```

New Functions

PHP 8 provides a number of new functions that offer greater functionality and improvements to the language, covering areas such as string handling, debugging, and error handling.

str_contains

The `str` function will return whether or not a string is contained within another string. It is a better function than `strpos`, because `strpos` returns the position of a string within another, or the value `false` if it's not found. But there's a potential problem because if the string is found at position zero and the value 0 is returned, this evaluates to `false` unless the strict comparison operator (`===`) is used instead of `==`.

Therefore, using `strpos`, you need to write less-than-clear code such as this:

```
if (strpos('Once upon a time', 'Once') !== false)
  echo 'Found';
```

whereas code using `str_contains`, such as the following, is far clearer to understand when quickly scanning (and writing) code and is less likely to lead to obscure bugs:

```
if (str_contains('Once upon a time', 'Once'))
  echo 'Found';
```

The `str_contains` function is case-sensitive, so if you need to perform a case-*in*sensitive check you should run both `$needle` and `$haystack` through a function to remove case first, such as `strtolower`, like this:

```
if (str_contains(strtolower('Once upon a time'),
                 strtolower('once')))
  echo 'Found';
```

Should you wish to use `str_contains` and also ensure your code is backward compatible with older versions of PHP, you can employ the use of a polyfill (code that provides a feature your code expects to be natively available) to create your own version of the `str_contains` function (should it not already exist).

Help with Polyfills

Rather than coming up with your own polyfills, which could inadvertently introduce errors into your code, or be noncompatible with other people's polyfills, you can use the ready-made Symfony polyfill package for PHP 8 available for free on GitHub.

In the following polyfill function for PHP 7+, the check for `$needle` being the empty string is there because PHP 8 considers the empty string to exist at every position within every string (even including within the empty string), so this behavior must be matched by the replacement function:

```
if (!function_exists('str_contains'))
{
  function str_contains(string $haystack, string $needle): bool
  {
    return $needle === '' || strpos($haystack, $needle) !== false;
  }
}
```

str_starts_with

The `str_starts_with` function provides a clearer means of checking whether one string starts with another. Previously you would probably use the `strpos` function and check whether it returns the value zero, but since we have already seen that 0

and `false` can become confused in certain situations, `str_starts_with` reduces this possibility substantially. You use it like this:

```
if (str_starts_with('In the beginning', 'In'))
  echo 'Found';
```

Just as with `str_contains`, the test is made in a case-sensitive manner, so use a function such as `strtolower` on both strings to perform an insensitive test. A PHP 7+ polyfill for this function could be the following:

```
if (!function_exists('str_starts_with'))
{
  function str_starts_with(string $haystack, string $needle): bool
  {
    return $needle === '' || strpos($haystack, $needle) === 0;
  }
}
```

Since PHP 8 considers the empty string to exist at every position in a string, this polyfill always returns `true` if `$needle` is the empty string.

str_ends_with

This function provides a clearer and simpler means of checking whether one string ends with another. Previously you would probably use the `substr` function, passing it the negative length of `$needle`, but `str_ends_with` makes this task far simpler. You use it like this:

```
if (str_ends_with('In the end', 'end'))
  echo 'Found';
```

Just as with the other new string functions, the test is made in a case-sensitive manner, so use a function such as `strtolower` on both strings to perform an insensitive test. A PHP 7+ polyfill for this function could be the following:

```
if (!function_exists('str_ends_with'))
{
  function str_ends_with(string $haystack, string $needle): bool
  {
    return $needle === '' ||
           $needle === substr($haystack, -strlen($needle));
  }
}
```

If the second argument passed to `substr` is negative (as in this case), then the string is matched working backward that number of characters from its end. Once again this polyfill always returns `true` if `$needle` is the null string. Also, note the use of the `===` strict comparison operator to ensure an exact comparison is made between the two strings.

fdiv

The new `fdiv` function is similar to the existing `fmod` (which returns a floating-point remainder after a division) and `intdiv` (which returns the integer quotient of a division) functions, but it allows for division by 0 without issuing a division-by-zero error, but returning one of `INF`, `-INF`, or `NAN`, depending on the case.

For example, `intdiv(1, 0)` will issue a division-by-zero error, as will `1 % 0`, or simply `1 / 0`. But you can safely use `fdiv(1, 0)`, and the result will be a float with the value `INF`, while `fdiv(-1, 0)` returns `-INF`.

Here's a PHP 7+ polyfill that you can use to make your code backward compatible:

```
if (!function_exists('fdiv'))
{
  function fdiv(float $dividend, float $divisor): float
  {
    return @($dividend / $divisor);
  }
}
```

get_resource_id

PHP 8 adds the new `get_resource_id` function that is similar to an `(int) $resource` cast to make it easier to retrieve a resource ID, but the return type is checked to be a resource, and the return value is an integer, and therefore they are type safe.

get_debug_type

The `get_debug_type` function provides more consistent values than the existing `get type` function and is best used to get detailed information on an unexpected variable in exception messages or logs, because it is more verbose and provides additional information. For more information, please refer to the Wiki.

preg_last_error_msg

PHP's `preg_` functions do not throw exceptions, so if there is an error, you must retrieve any message using `preg_last_error` to get the error code. But if you want a friendly error message instead of just an unexplained code, in PHP 8 you can now call `preg_last_error_msg`. If there was no error, then "No error" is returned.

PHP Versions 8.1 and 8.2

The original PHP version 8.0 was released in 2020, while a newer version 8.1 came out at the end of 2021, with an 8.2 version following in late 2022. Between them theses two updates include a number of new features and enhancements that you

need to know about even if you may not start using them quite yet, or at least until the new versions are far more prevalent. That said, if you know for sure your code will be running only on a particular version of PHP, then by all means go ahead and enjoy these new features.

The official documentation for the new versions can be found at the following URLs.

- php.net/releases/8.1/en.php
- php.net/releases/8.2/en.php

Or, for a more easily digestible run-down of the features in these new versions, I recommend the following excellent blog posts:

- stitcher.io/blog/new-in-php-81
- stitcher.io/blog/new-in-php-82

As this book is aimed at the beginner to intermediate level, I have only really scratched the surface of all the great new features in PHP 8, giving you a taste of the main ones that you can start using right away. However, if you are keen to learn absolutely everything about this milestone update, you can get full details at the official web page.

MySQL 8

This section will help you to catch up with all that MySQL 8 has to offer over earlier releases, such as better Unicode support, better JSON and document handling, geographic support, and window functions.

In this summary, you'll get an overview of eight of the areas that have been improved, upgraded, or added to in the latest release.

MySQL 8 is the successor to version 5.7 because version 6 was never really taken up by the development community, and when Sun Microsystems purchased MySQL, development of version 6 was halted. More importantly, until version 8 the biggest change was from 5.6 to 5.7 so, according to Sun, "Due to the many new and important features we were introducing in this MySQL version, we decided to start a fresh new series. As the series numbers 6 and 7 had actually been used before by MySQL, we went to 8.0."

Updates to SQL

MySQL 8 now has window functions (also known as *analytic functions*). They are similar to grouped aggregate functions, which collapse calculations on sets of rows into a single row. However, a window or analytic function performs the aggregation

for each row in the result set and is nonaggregate. As they are not central to your use of MySQL and should be considered advanced extensions, I do not cover them in this book.

The new functions are `RANK`, `DENSE_RANK`, `PERCENT_RANK`, `CUME_DIST`, `NTILE`, `ROW_NUMBER`, `FIRST_VALUE`, `LAST_VALUE`, `NTH_VALUE`, `LEAD`, and `LAG`, and if you need to know more about them, they are fully documented on the MySQL official website.

MySQL 8 also brings recursive Common Table Expressions, enhanced alternatives of `NOWAIT` and `SKIP LOCKED` in the SQL locking clause, Descendent Indexes, a `GROUPING` function, and Optimizer Hints.

All these and more can be viewed on the MySQL website.

JSON (JavaScript Object Notation)

There are a number of new functions for handling JSON, while sorting and grouping of JSON values has been improved. As well as adding extended syntax for ranges in path expressions and improved sorting, there are new table, aggregation, merge, and other functions.

Given these improvements to MySQL and the use of JSON, it could be argued that MySQL can now be used instead of NoSQL databases.

The use of JSON in MySQL is beyond the scope of this book, but if it is an area of interest to you, you can refer to the official documentation on all the new features.

Geography Support

MySQL 8 also brings GIS, or Geography Support, including metadata support for SRS (Spatial Reference System), SRS data types, indexes, and functions. This means that MySQL can now (for example) calculate the distance between two points on the Earth's surface using their latitude and longitude coordinates in any of the supported spatial reference systems.

For further details on how to access MySQL GIS, you can refer to the official website.

Reliability

MySQL is already extremely reliable, but it has been even further improved with version 8 by storing its metadata in the InnoDB transactional storage engine, such that *Users*, *Privileges*, and *Dictionary* tables now reside in `InnoDB`.

In MySQL 8 there is now only a single data dictionary, whereas in version 5.7 and earlier there were two data dictionaries (one for the server and one for the `InnoDB` layer), which could get out of sync.

From version 8 the user is guaranteed that any DDL statement (such as `CREATE TABLE`) will either be executed fully or not at all, preventing Primary and Replica servers from possibly getting out of sync.

Speed and Performance

In MySQL, the Information Schema has been sped up by up to 100 times by storing the tables as simple views on data dictionary tables in `InnoDB`. Additionally, over 100 indexes on Performance Schema tables have been added to further speed up performance, which is also much improved with faster reads and writes, I/O bound workloads, and high-contention workloads, plus you can now map user threads to CPUs to further optimize performance.

MySQL 8 scales better on heavy write workloads by a factor of up to four times compared with version 5.7 and offers even more significant increases for read/write workloads.

With MySQL you can use every storage device at its highest ability and performance is increased on high-contention workloads (where transactions are queuing up to be granted a lock).

In all, the developers of MySQL 8 say that it is up to twice as fast, and you can read their reasoning and tips on how you can achieve this performance increase in your apps at the official website.

Management

With MySQL 8 you can now toggle an index between visible and invisible. An invisible index is not considered by the optimizer when a query plan is created, but the index is still maintained in the background, so it is easy to make it visible again, allowing you to decide whether an index is droppable.

Also, users now have full control over Undo tablespaces, and you can now persist global, dynamic server variables that would be lost upon server restart. Plus there's now an SQL `RESTART` command to allow remote management of a MySQL server over an SQL connection, and in MySQL 8 there's a new `ALTER TABLE...RENAME COLUMN` command provided as an improvement over the previous `ALTER TABLE...CHANGE` syntax.

For further details, please refer to the official website.

Security

Security was definitely not left on the table when version 8 was planned, as there are many new improvements.

To start with, the default authentication plug-in has changed from `mysql_native_password` to `caching_sha2_password`, and OpenSSL has been selected as the default TLS/SSL library in both the Enterprise and now the Community editions and is dynamically linked.

With MySQL 8, the Undo and Redo logs are now encrypted, and SQL roles have been implemented such that you can grant roles to users and privileges to roles. You can also use mandatory roles when new users are created. There is now a password rotation policy configurable as global or user level with a securely stored password history.

In the authentication process, a delay has been added in MySQL 8 based on consecutive unsuccessful login attempts to slow down brute force attempts at attack. Triggering of and the maximum length of delay is configurable.

For more information on MySQL and security, please visit the official website. And for full details on all that's included in MySQL 8.0.31, please check out the release notes at tinyurl.com/mysql8031.