www.phparch.com

July 2018
Volume 17 - Issue 7

php[architect]

Navigating State

State in the Stateless World

Making Use of Our Robot Overlords

Pro Parsing Techniques with PHP, Part Two: Fault Tolerance

MySQL Without The SQL —Oh My!

Free Sample Article

ALSO INSIDE

The Dev Lead Trenches: Issue Workflows for Teams

Community Corner: Contributing is More Than Coding

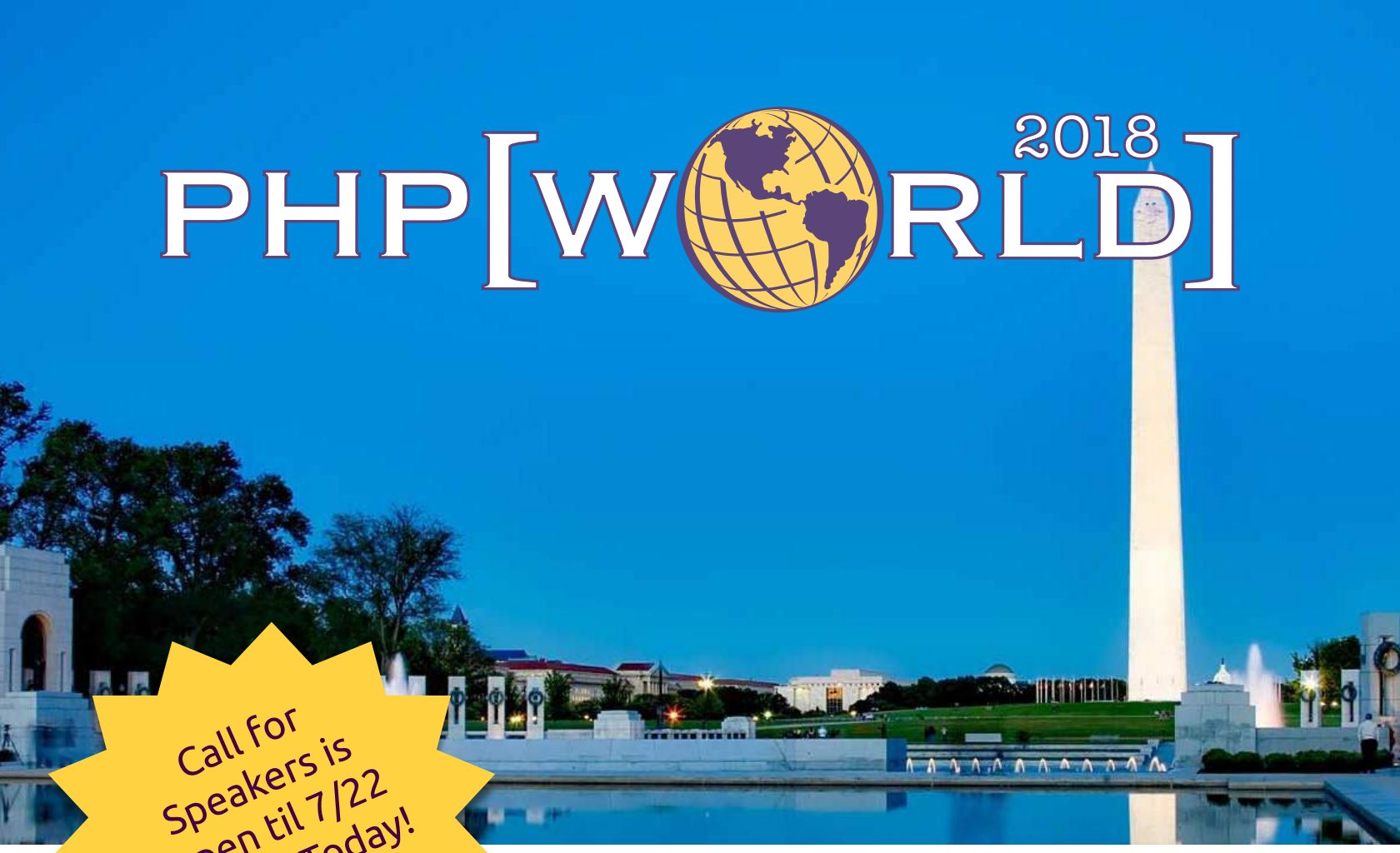
Security Corner: Secure Remote Password Authentication

Education Station: Build an Algorithm

The Workshop: CakePHP, Part Two

finally{}: Do We Need Developers?

PHP[WORLD] 2018



Call for Speakers is Open til 7/22 Submit Today!

November 14–15, 2018
Washington, D.C.

A conference designed to bring together all communities linked by the PHP programming language.

world.phparch.com

CONTENTS

php[architect]

JULY 2018
Volume 17 - Issue 7

Features

Columns

Editor-in-Chief: Oscar Merida

Editor: Kara Ferguson

Managing Partners

Oscar Merida, Sandy Smith

php[**architect**] is published twelve times a year by: musketeers.me, LLC
201 Adams Avenue
Alexandria, VA 22301, USA

Subscriptions

Print, digital, and corporate subscriptions are available. Visit https://www.phparch.com/magazine to subscribe or email contact@phparch.com for more information.

Advertising

To learn about advertising and receive the full prospectus, contact us at ads@phparch.com today!

Contact Information:
General mailbox: contact@phparch.com
Editorial: editors@phparch.com

Print ISSN 1709-7169
Digital ISSN 2375-3544

MySQL Without The SQL—Oh My!

Dave Stokes

Or How to Stop Embedding Ugly Strings of SQL In Your Beautiful Code

Do you work on projects where you begin coding before knowing what your data looks like? Or are you part of the vast majority of developers who have had little or no training in database theory, relational calculus, Structured Query Language, or sets? Could you be working on a project without a database administrator to set up relational tables, indexes, and schemas? Or are you tired of embedding ugly lines of SQL in your pristine PHP code? There is new hope for you.

The MySQL Document Store

MySQL 5.7 introduced a native JSON data type[1] in version 5.7, and it has been greatly enhanced with version 8. Built on top of the JSON data type is the MySQL Document Store which is a NoSQL JSON document store. It enables you to use a schema-less, flexible storage system. You do not need to define the attributes of the data, setup relations, or normalize data. Just create a collection and populate with data.

You can use the new MySQL Shell to login to a server and create a collection. In the example in Output 1, I used the MySQL Shell to connect to the schema demo, create a new collection, and a record. And yes, you can do this in PHP too!

The MySQL Document Store is built on a new protocol with a new API—the X DevAPI[2]. The old MySQL protocol was beginning to show its age after over two decades of use. The new protocol listens on port 33060 as opposed to the traditional MySQL 3306 port. There are several interesting features of the protocol so it can better handle asynchronous queries, management of high availability server clusters, and even make sure instances are ready for upgrade. But for this article we will only look at the functionality of the document store.

What is a Document?

A document is a set of keys and value pairs in a JSON object. The values of the fields can contain other documents, arrays, and lists of documents. The MySQL JSON data type provides roughly a gigabyte of space in a column of a row in a table.

Remember that collection created earlier? If we use SQL to look at the data from the JSON column you will see something like Output 2.

Ignore the _id data for a few moments. The key/value data entered earlier for name/architect are easy to spot.

Collections are a container made up of documents. And you can also access relational tables with the MySQL Document Store too but ignore that for now. You can perform CRUD (create, read, update, and delete operations) on collections or documents. The API and the new protocol ensure the calls to the database look pretty much the same regardless of whether you are coding in C++, Java, Node.JS, JavaScript, .NET, or even PHP.

The X DevAPI PECL Extension

The XDevAPI PECl extension[3] is available online and the documentation and installation instructions[4].

Output 1

```
MySQL JS> \connect root@localhost/demo Creating a session to
'root@localhost/demo'
Enter password: ******
Fetching schema names for autocompletion... Press ^C to stop.
Your MySQL connection id is 9 (X protocol) Server version: 8.0.11
MySQL Community Server - GPL Default schema `demo` accessible
through db.
MySQL [localhost+ ssl/demo] JS> db.createCollection('architect')
 <Collection:architect>
MySQL [localhost+ ssl/demo] JS> db.architect.add(
    {
        'name' : 'PHP Architect'
    }
)
Query OK, 1 item affected (0.1040 sec)
MySQL [localhost+ ssl/demo] JS>
```

1 *JSON data type:* https://phpa.me/mysql8-json

2 *X DevAPI:* https://phpa.me/mysql-xdev-user-guide

3 *XDevAPI PECl extension:* https://pecl.php.net/package/mysql_xdevapi

4 *installation instructions:* https://phpa.me/php-mysql-xdevapi

Example

The example in Listing 1 will look familiar to those who use MySQL or any other relational database. First, you need to authenticate into the server. Next, you select your schema (or database) and then the document collection to use. From there we are looking to find a record where the _id field matches a particular value.

The next item is the big change. With traditional MySQL, your query would look something like:

```
SELECT * FROM countryinfo
   WHERE _id = "USA";
```

If we only wanted the JSON document, it would really look like:

```
SELECT doc FROM countryinfo
   WHERE _id = "USA":
```

With the document store, the query becomes:

```
$collection->find('_id = "USA"')
           ->execute();
```

That has a much cleaner appearance than the corresponding embedded SQL string in our beautiful, pristine PHP code.

Finding Find()

The find function has several parameters that can be passed to it. Besides the search condition (_id = "USA") in the previous example, you can specify which fields you desire, how to group by, a having search condition, sorting options, limits and offsets, parameter binding to variables, and locking on records (exclusive or shared).

The other CRUD functions—add, find, modify, and remove as they are called in the MySQL Document Store—are equally festooned with options. You get a very rich environment to manage your data.

Did I mention you can also use the MySQL Document Store with good ol' relational tables? There are slightly different CRUD functions for relational tables that map pretty well to their standard SQL counterparts—insert, select, update, and delete—with the main difference being the select function also has an order by optional parameter.

Behind the Scenes

Underneath the cover, a collection from the SQL side is a table with two columns. The first is creatively called doc for the JSON document and the second is _id. InnoDB storage engine tables are much happier with a primary key, and you can supply your own value for the _id field or let the server generate it for you. Since InnoDB is transactional and locks at the row level, you can access the same data via SQL or the document store at the same time, perform transactions, replicate data, and all the other usual things you can do with MySQL.

Dave Stokes started using PHP when it was known as Personal Home Page and started working for MySQL AN as a PHP Developer. He is now a MySQL Commmunity Manager for Oracle Corporation. He lives in Texas with the required hound dog and pickup truck. @stoker

Output 2

```
MYSQL [LOCALHOST+ SSL/DEMO] SQL> SELECT DOC FROM ARCHITECT;

+-------------------------------------------------------------------+
| doc                                                               |
+-------------------------------------------------------------------+
| {"_id": "00005b100c730000000000000001", "name": "PHP Architect"} |
+-------------------------------------------------------------------+
1 row in set (0.0017 sec)
MySQL [localhost+ ssl/demo] SQL>
```

Listing 1

```
1.  #!/usr/bin/php
2.  <?php
3.  // Connection parameters
4.  $user = 'root';
5.  $passwd = 'S3cret#';
6.  $host = 'localhost';
7.  $port = '33060';
8.  $connection_uri = 'mysqlx://' . $user . ':' . $passwd
9.                  . '@' . $host . ':' . $port;
10.
11.
12. // Connect as a Node Session
13. $nodeSession = mysql_xdevapi\getNodeSession($connection_uri);
14. // "USE world_x"
15. $schema = $nodeSession->getSchema("world_x");
16. // Specify collection to use
17. $collection = $schema->getCollection("countryinfo");
18. // SELECT * FROM world_x WHERE _id = "USA"
19. $result = $collection->find('_id = "USA"')->execute();
20. // Fetch/Display data
21. $data = $result->fetchAll();
22. var_dump($data);
```

Borrowed this magazine?

Get php[**architect**] delivered to your doorstep or digitally every month!

Each issue of php[**architect**] magazine focuses on an important topic that PHP developers face every day.

We cover topics such as frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

php[**architect**]

Digital and Print+Digital Subscriptions Starting at $49/Year

http://phpa.me/mag_subscribe