PHP PDO tutorial

In this tutorial we show how to program databases in PHP PDO.

PHP PDO

The PHP Data Objects (PDO) defines a lightweight interface for accessing databases in PHP. It provides a data-access abstraction layer for working with databases in PHP. It defines consistent API for working with various database systems.

PHP PDO classes

The PDO represents a connection between PHP and a database server. The PDOStatement represents a prepared statement and, after the statement is executed, an associated result set. The PDOException represents an error raised by PDO.

MySQL database

In this tutorial, we work with MySQL database.

countries_mysql.sql

```
DROP TABLE IF EXISTS countries;
CREATE TABLE countries(id BIGINT NOT NULL PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(255), population INT);

INSERT INTO countries(name, population) VALUES('China', 1382050000);
INSERT INTO countries(name, population) VALUES('India', 1313210000);
INSERT INTO countries(name, population) VALUES('USA', 324666000);
INSERT INTO countries(name, population) VALUES('Indonesia', 260581000);
INSERT INTO countries(name, population) VALUES('Brazil', 207221000);
INSERT INTO countries(name, population) VALUES('Pakistan', 196626000);
INSERT INTO countries(name, population) VALUES('Nigeria', 186988000);
INSERT INTO countries(name, population) VALUES('Bangladesh', 162099000);
INSERT INTO countries(name, population) VALUES('Nigeria', 186988000);
INSERT INTO countries(name, population) VALUES('Russia', 146838000);
INSERT INTO countries(name, population) VALUES('Japan', 126830000);
```

These SQL commands create a countries table.

PHP PDO query

The PDO query() executes an SQL statement in a single function call, returning the result set (if any) returned by the statement.

version.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$stm = $pdo->query("SELECT VERSION()");

$version = $stm->fetch();

echo $version[0] . PHP_EOL;
```

The example returns the version of MySQL.

```
$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";
```

These variables are used to create a connection string to the database. The dsn is the Data Source Name, which contains the information required to connect to the database.

```
$pdo = new PDO($dsn, $user, $passwd);
```

A new PDO object is created. We pass the constructor the data source name and the user name and password. The PDO class represents a connection between PHP and a database server.

```
$stm = $pdo->query("SELECT VERSION()");
```

The query() method executes an SQL statement in a single function call. It returns the result set.

```
$version = $stm->fetch();
```

The PDO statement's fetch() method fetches the next row from a result set. In our case it is a version of MySQL.

```
echo $version[0] . PHP_EOL;
```

The $version is an array; we get its first value.

```
$ php version.php
5.7.22-0ubuntu0.16.04.1
```

This is the output.

PHP PDO exec

The PDO exec() executes an SQL statement and returns the number of affected rows.

affected_rows.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$id = 12;

$nrows = $pdo->exec("DELETE FROM countries WHERE id IN (1, 2, 3)");

echo "The statement affected $nrows rows\n";
```

The code example deletes three rows. It prints the number of affected rows.

```
$nrows = $pdo->exec("DELETE FROM countries WHERE id IN (1, 2, 3)");
```

In this SQL statement, we delete rows with ids 1, 2, and 3. The number of deleted rows is stored in the $nrows variable.

```
echo "The statement deleted $nrows rows\n";
```

We print the number of deleted rows.

PHP PDO fetch style

The fetch style parameter controls how the next row will be returned to the caller. For instance, the PDO::FETCH_ASSOC returns an array indexed by column name, PDO::FETCH_NUM returns an array indexed by column number, and the PDO::FETCH_BOTH returns an array indexed by both column name and indexed column number. The default fetch style is PDO::FETCH_BOTH.

fetch_style_num.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$stm = $pdo->query("SELECT * FROM countries");

$rows = $stm->fetchAll(PDO::FETCH_NUM);

foreach($rows as $row) {

    printf("$row[0] $row[1] $row[2]\n");
}
```

In this code example, we get data in an indexed array.

```
$stm = $pdo->query("SELECT * FROM countries");
```

We select all data from the countries table.

```
$rows = $stm->fetchAll(PDO::FETCH_NUM);
```

We pass the PDO:FETCH_NUM style to the fetchAll() method.

```
foreach($rows as $row) {

    printf("$row[0] $row[1] $row[2]\n");
}
```

We go over the $rows array and print the fields. The fields are accessed via array indexes.

fetch_style_assoc.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$stm = $pdo->query("SELECT * FROM countries");

$rows = $stm->fetchAll(PDO::FETCH_ASSOC);

foreach($rows as $row) {

    printf("{$row['id']} {$row['name']} {$row['population']}\n");
}
```

In this example, we fetch data as an associative array.

```
$rows = $stm->fetchAll(PDO::FETCH_ASSOC);
```

In the fetchAll() method, we use the PDO::FETCH_ASSOC style.

PHP PDO parameter binding

SQL statements are often dynamically built. A user provides some input and this input is built into the statement. We must be cautious every time we deal with an input from a user. It has some serious security implications. The recommended way to dynamically build SQL statements is to use parameter binding.

PDO contains bindParam() and bindValue() method to create parameterized queries.

PDO allows to bind data to question mark or named placeholders.

parameterized_query.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "root";
$passwd = "andrea";

$pdo = new PDO($dsn, $user, $passwd);

$id = 12;

$stm = $pdo->prepare("SELECT * FROM countries WHERE id = ?");
$stm->bindValue(1, $id);
$stm->execute();

$row = $stm->fetch(PDO::FETCH_ASSOC);

echo "Id: " . $row['id'] . PHP_EOL;
echo "Name: " . $row['name'] . PHP_EOL;
echo "Population: " . $row['population'] . PHP_EOL;
```

In the example, we use bindValue() to create a parameterized query. We use question mark placeholder.

```
$id = 12;
```

Say this input comes from a user.

```
$stm = $pdo->prepare("SELECT * FROM countries WHERE id = ?");
$stm->bindValue(1, $id);
$stm->execute();
```

The select statement fetches a specific row from the table. We bind the value with bindValue() to a question mark placeholder.

In the second case, we use bindParam().

parameterized_query2.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$id = 12;

$stm = $pdo->prepare("SELECT * FROM countries WHERE id = :id");
$stm->bindParam(":id", $id, PDO::PARAM_INT);
$stm->execute();

$row = $stm->fetch(PDO::FETCH_ASSOC);

echo "Id: " . $row['id'] . PHP_EOL;
echo "Name: " . $row['name'] . PHP_EOL;
echo "Population: " . $row['population'] . PHP_EOL;
```

The example selects and prints a specific row.

```
$stm = $pdo->prepare("SELECT * FROM countries WHERE id = :id");
$stm->bindParam(":id", $id, PDO::PARAM_INT);
$stm->execute();
```

This time we use named placeholder (:id) and bindParam().

PHP PDO last inserted row Id

PDO lastInsertId() method returns the last inserted row id.

create_table.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$sql = "CREATE TABLE words(id INT PRIMARY KEY AUTO_INCREMENT,
    word VARCHAR(255))";

$ret = $pdo->exec($sql);
$pdo->exec("INSERT INTO words(word) VALUES ('pen')");
$pdo->exec("INSERT INTO words(word) VALUES ('bum')");
$pdo->exec("INSERT INTO words(word) VALUES ('hum')");
$pdo->exec("INSERT INTO words(word) VALUES ('den')");

$rowid = $pdo->lastInsertId();

echo "The last inserted row id is: $rowid\n";
```

In the example, we create a new table. After the table was created, we find out the last inserted Id with lastInsertId().

```
$ php create_table.php
The last inserted row id is: 4
```

This is the output.

```
mysql> select * from words;
+----+------+
| id | word |
+----+------+
|  1 | pen  |
|  2 | bum  |
|  3 | hum  |
|  4 | den  |
+----+------+
4 rows in set (0.01 sec)
```

We verify the data.

PHP PDO Transactions

A *transaction* is an atomic unit of database operations against the data in one or more databases. The effects of all the SQL statements in a transaction can be either all committed to the database or all rolled back.

The PDO beginTransaction() initiates a new transaction. The PDO commit() commits the transaction. And the PDO rollback() rolls back the transaction.

transaction.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

try {

    $pdo->beginTransaction();
    $stm = $pdo->exec("INSERT INTO countries(name, population) VALUES ('Iraq', 38274000)");
    $stm = $pdo->exec("INSERT INTO countries(name, population) VALUES ('Uganda', 37673800)");

    $pdo->commit();

} catch(Exception $e) {

    $pdo->rollback();
    throw $e;
}
```

In the example, we add two new countries to the database table. The insert statements are placed in a transaction: either both inserts are executed or none.

```
} catch(Exception $e) {

    $pdo->rollback();
    throw $e;
}
```

In case of an exception, we roll the trancaction back: no data is written to the database. We throw the exception so that the exception handling continues the usual way.

PHP PDO getting metadata

Metadata is information about the data in a database. Metadata contains information about the tables and columns in which we store data. The number of rows that an SQL statement affects is metadata. The number of rows and columns returned in a result set are metadata as well.

column_count.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$stm = $pdo->query("SELECT name, population FROM countries WHERE id=1");

$ncols = $stm->columnCount();

echo "The result set contains $ncols columns\n";
```

In the example, we print the number of columns in the result set with columnCount() method.

The getAttribute() method retrieves a database connection attribute.

connection_attributes.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$driver = $pdo->getAttribute(PDO::ATTR_DRIVER_NAME);
$server_version = $pdo->getAttribute(PDO::ATTR_SERVER_VERSION);
$autocommit_mode = $pdo->getAttribute(PDO::ATTR_AUTOCOMMIT);

echo "Driver: $driver\n";
echo "Server version: $server_version\n";
echo "Autocommit mode: $autocommit_mode\n";
```

In the example, we get the driver name, server version, and autocommit mode with the getAttribute() method.

```
$ php connection_attributes.php
Driver: mysql
Server version: 5.7.22-0ubuntu0.16.04.1
Autocommit mode: 1
```

This is a sample output.

In the following example, we print the column metadata. The column metadata is retrieved with getColumnMeta() method.

column_metadata.php

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$stm = $pdo->query("SELECT * FROM countries WHERE id=1");

$col_meta = $stm->getColumnMeta(0);

echo "Table name: {$col_meta["table"]}\n";
echo "Column name: {$col_meta["name"]}\n";
echo "Column length: {$col_meta["len"]}\n";
echo "Column flags: {$col_meta["flags"][0]} {$col_meta["flags"][1]} \n";
```

In the example, we get the column table, name, length, and flags.

```
$ php column_metadata.php
Table name: countries
Column name: id
Column length: 20
Column flags: not_null primary_key
```

This is a sample output.

You might also be interested in the following related tutorials: PHP SQLite3 tutorial, CakePHP database tutorial, Doctrine QueryBuilder tutorial, PHP filesystem functions, Twig tutorial, PHP Faker tutorial, PHP tutorial, or list all PHP tutorials.

In this tutorial, we have worked with MySQL database with PHP PDO.

