(Tylko właściwy) poradnik PDO

1. Dlaczego PDO?
2. Złączony. DSN
3. Uruchamianie zapytań. PDO :: query ()
4. Przygotowane sprawozdania. Ochrona przed SQL injection
    - sposoby wiązania
    - Części zapytań można powiązać
5. Przygotowane sprawozdania. Wielokrotne wykonanie
6. Uruchamianie SELECT INSERT, UPDATE lub DELETE
7. Pobieranie danych z rachunku. dla każdego()
8. Pobieranie danych z rachunku. sprowadzać()
    - Powrót typy.
9. Pobieranie danych z rachunku. fetchColumn ()
10. Pierwsze dane z rachunku w dziesiątkach różnych formatach. fetchAll ()
    - Pierwsze zwykły tablicy.
    - Pierwsze kolumny.
    - Pierwsze par klucz-wartość.
    - Pierwsze wiersze indeksowane przez unikalnej dziedzinie
    - Pierwsze wiersze pogrupowane według jakiejś dziedzinie
11. obsługa błędów. wyjątki
    - Raportowanie błędów PDO
    - Łapanie wyjątków PDO
12. Uzyskiwanie liczba wierszy z PDO
13. Wierszy i wstawić identyfikator
14. Przygotowane sprawozdania i podobnie jak klauzula
15. Przygotowane sprawozdania i klauzula
16. Przygotowane oświadczenia i nazwy tabel
17. Problem z klauzuli LIMIT
18. transakcje
19. Wywoływanie procedur przechowywanych w PDO
20. Uruchamianie wielu zapytań z PDO
21. Tryb emulacji. PDO :: ATTR_EMULATE_PREPARES
    - Gdy tryb emulacji jest włączony
    - Gdy tryb emulacji jest wyłączony
22. Mysqlnd i buforowane zapytań. Ogromne zbiory danych.
23. Komentarze (86)

Istnieje wiele tutoriali na PDO już, niestety, większość z nich nie wyjaśniają rzeczywiste korzyści PDO, a nawet promować raczej złe praktyki. Jedyne dwa wyjątki są phptherightway.com i hashphp.org , ale brakuje wielu ważnych informacji. W rezultacie połowa funkcji PDO pozostawać w ukryciu i prawie nigdy nie są używane przez programistów PHP, którzy w rezultacie nieustannie próbuje wyważać otwartych drzwi, które *już istnieje w PDO.*

W przeciwieństwie do nich, ten poradnik został napisany przez kogoś, kto używane PDO przez wiele lat, wykopanych przez niego, i odpowiedział na tysiące pytań na przepełnienie stosu (The podeszwa złota odznaka PDO okaziciela). W następstwie misji z tej strony , ten artykuł będzie obalić różne urojenia i złych praktyk, pokazując właściwą drogę zamiast.

Chociaż ten poradnik jest oparta na **mysql** sterownika, informacje na ogół stosuje się do każdego kierowcy wspieranego.

Dlaczego PDO?

Pierwsze rzeczy pierwsze. Dlaczego PDO w ogóle?

PDO jest bazą danych dostępu do warstwy abstrakcji . Abstrakcja, jednak jest dwojaki: jeden jest powszechnie znany, ale mniej znaczące, podczas gdy inny jest niejasny, ale największe znaczenie.

Każdy wie, że PDO oferuje jednolity interfejs dostępu do wielu różnych baz danych . Chociaż ta funkcja jest wspaniały sam w sobie, to nie ma wielkiego dla danego zastosowania, gdzie tylko jeden backend bazy danych używanej w każdym razie. I pomimo plotek, że jest niemożliwe, aby przełączyć się z bazami danych poprzez zmianę pojedynczej linii w PDO config - z powodu różnych smakach SQL (aby to zrobić, trzeba użyć uśrednionego jak język zapytań DQL). Tak więc, dla przeciętnego LAMP dewelopera, ten punkt jest raczej niewielkie, a do niego PDO jest po prostu bardziej skomplikowana wersja znanej `mysql(i)_query()` funkcji. Jednak to nie jest; to jest o wiele, wiele więcej.

PDO abstrakty nie tylko API bazy danych, ale również podstawowe operacje, które w przeciwnym razie muszą być powtarzane setki razy w każdej aplikacji, dzięki czemu kod niezwykle WET . W przeciwieństwie do *mysql* i *mysqli* , z których oba mają niski poziom nagie API nie jest przeznaczony do bezpośredniego stosowania (ale tylko jako materiał budowlany na pewnym poziomie abstrakcji wyższej warstwy) `PDO` *jest* już takie abstrakcją. Chociaż nadal niekompletna, ale przynajmniej użyteczny.

Rzeczywiste korzyści PDO są:

- **bezpieczeństwo** (*nadające* Przygotowane sprawozdania)
- **Użyteczność** (wiele funkcji pomocniczych do automatyzacji rutynowych czynności)
- **ponownego użycia** (zunifikowane API do dostępu wiele baz danych, od SQLite do Oracle)

Należy zauważyć, że choć PDO jest najlepsza z rodzimych db kierowców, dla nowoczesnej aplikacji internetowej rozważyć użycie ORM z Query Builder lub innego wyższego poziomu abstrakcji biblioteki, tylko okazjonalnego awaryjnej Vanilla PDO. Dobre ORMs są Doctrine, elokwentny, Redbean i Yii :: AR. Aura.SQL jest dobrym przykładem owijki PDO z wieloma dodatkowymi funkcjami.

Tak czy inaczej, dobrze jest znać podstawowe narzędzia w pierwszej kolejności. Więc zacznijmy:

Złączony. DSN

PDO ma ochotę metodę połączenia o nazwie DSN . To nic skomplikowanego, choć - zamiast jednego zwykłego i prostego listy opcji, PDO poprosi wejściowych różnych dyrektyw konfiguracyjnych w trzech różnych miejscach:

- `database driver` , `host` , `db (schema) name` I `charset` , a także rzadziej używane `port` i `unix_socket` przejść do DSN;
- `username` i `password` przejść do konstruktora;
- Wszystkie inne opcje przejść do opcji tablicy.

gdzie DSN jest ciąg rozdzielany średnikami, składa się z `param=value` pary, który rozpoczyna się od nazwiska kierowcy i dwukropkiem:

```
      mysql:host=localhost;dbname=test;port=3306;charset=utf8
driver^    ^ colon         ^param=value pair    ^semicolon
```

Należy pamiętać, że ważne jest, aby postępować zgodnie z właściwego formatu - **bez spacji lub cytaty lub inne dekoracje muszą być wykorzystane w DSN,** ale tylko parametry, wartości i ograniczniki, jak pokazano w instrukcji .

Tu idzie przykład dla MySQL:

```
$host = '127.0.0.1';
$db   = 'test';
$user = 'root';
$pass = '';
$charset = 'utf8';

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";
$opt = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];
$pdo = new PDO($dsn, $user, $pass, $opt);
```

Ze wszystkie zmienne wspomniane prawidłowo ustawiona, mamy właściwą instancję PDO `$pdo` zmiennej.

Ważne informacje dla użytkowników późnych rozszerzenia mysql:

1. W przeciwieństwie do dawnych `mysql_*` funkcji, które mogą być stosowane w dowolnym miejscu w kodzie, `PDO` wystąpienie jest przechowywana w zwykłej zmiennej, co oznacza, że może być niedostępny wewnątrz funkcji - tak, trzeba udostępnić go poprzez

przepuszczenie go poprzez parametry funkcyjne lub przy użyciu więcej zaawansowane techniki, takie jak IoC kontenera.
2. Połączenie musi być wykonane tylko raz! Nie łączy się w każdej funkcji. Nie łączy się w każdym konstruktora klasy. W przeciwnym wypadku zostanie utworzony wiele połączeń, które będą ostatecznie zabić serwer bazy danych. Zatem jedyny przypadek PDO ma zostać utworzony, a następnie wykorzystywane przez cały wykonywania skryptu.
3. Jest bardzo ważne, aby **ustawić charset poprzez DSN** - to jedyna właściwa droga, ponieważ mówi PDO, które charset będzie używany. Dlatego też zapomnieć o prowadzeniu `SET NAMES` kwerendy ręcznie, albo za pomocą `query()` lub `PDO::MYSQL_ATTR_INIT_COMMAND` . Tylko wtedy, gdy wersja PHP jest zbyt przestarzały (czyli poniżej 5.3.6), trzeba użyć `SET NAMES` kwerendy i zawsze włączyć tryb emulacji off.

Uruchamianie zapytań. PDO :: query ()

Istnieją dwa sposoby, aby uruchomić kwerendę w PDO. Jeśli nie ma zmienne będą używane w kwerendzie, można skorzystać z PDO :: query () metody. Będzie ona trwać zapytanie i zwraca szczególną obiekt klasy PDOStatement które można z grubsza w porównaniu do zasobu, zwrócony przez `mysql_query()` , zwłaszcza w sposób można uzyskać rzeczywiste wiersze z niego:

```
$stmt = $pdo->query('SELECT name FROM users');
while ($row = $stmt->fetch())
{
    echo $row['name'] . "\n";
}
```

Ponadto, `query()` metoda ta pozwala nam korzystać schludny metodę łańcucha dla zapytań SELECT, które zostaną przedstawione poniżej.

Przygotowane sprawozdania. Ochrona przed SQL injection

Jest to główny i jedyny ważny powód, dla którego zostały pozbawione swojej ukochanej `mysql_query()` funkcji i rzucony w surowym świecie obiektów danych: PDO przygotował sprawozdanie wsparcie po wyjęciu z pudełka. Przygotowane oświadczenie jest **jedynym właściwym sposobem, aby uruchomić kwerendę** , jeżeli zmienna ma być użyty w nim. Powodem, dla którego tak ważne jest szczegółowo wyjaśnione w Autostopem podręczniku SQL zapobiegania wtrysku .

Tak więc, dla każdego zapytania biegać, jeżeli co najmniej jedna zmienna ma być używany, trzeba zastąpić go z **zastępczy** , a następnie przygotować zapytanie, a następnie uruchom go, przechodząc zmiennych osobno.

Krótko mówiąc, to nie jest tak trudne, jak się wydaje. W większości przypadków, trzeba tylko dwie funkcje - przygotowanie () i execute () .

Przede wszystkim, trzeba zmienić zapytanie, dodając zastępczych w miejsce zmiennych. Powiedzieć, jak ten kod

```
$sql = "SELECT * FROM users WHERE email = '$email' AND status='$status'";
```

stanie się

```
$sql = 'SELECT * FROM users WHERE email = ? AND status=?';
```

lub

```
$sql = 'SELECT * FROM users WHERE email = :email AND status=:status';
```

Zauważ, że PDO obsługuje pozycyjnych (`?`) i nazwie (`:email`) zastępcze, ten ostatni zawsze zaczyna się od dwukropka i mogą być zapisywane za pomocą liter, cyfr i podkreśleń tylko. Należy również pamiętać, że **żadne cytaty** muszą być zawsze używany na całym zastępcze.

Mając na zapytanie zastępcze, trzeba go przygotować, stosując `PDO::prepare()` metodę. Ta funkcja zwróci ten sam `PDOStatement` obiekt Mówiliśmy o wyżej, ale *bez jakichkolwiek danych dołączonych do niego* .

Wreszcie, aby kwerenda wykonana, należy uruchomić `execute()` metodę tego obiektu, przechodząc zmienne w nim, w postaci tablicy. A potem, będziesz w stanie uzyskać dane wynikające z rachunku (jeśli dotyczy):

```
$stmt = $pdo->prepare('SELECT * FROM users WHERE email = ? AND status=?');
$stmt->execute([$email, $status]);
$user = $stmt->fetch();
// or
$stmt = $pdo->prepare('SELECT * FROM users WHERE email = :email AND status=:status');
$stmt->execute(['email' => $email, 'status' => $status]);
$user = $stmt->fetch();
```

Jak widać, dla pozycyjnych zastępcze, trzeba dostarczyć regularną tablicę z wartościami, natomiast dla wymienionych elementów zastępczych, to musi być tablicą asocjacyjną, gdzie klucze muszą dopasować nazwy zastępcze w zapytaniu. Nie można mieszać pozycyjne i nazwane zastępcze w tej samej kwerendy.

Należy pamiętać, że pozycyjne zastępcze pozwalają napisać krótszy kod, ale są wrażliwe na kolejność argumentów (które muszą być dokładnie taka sama jak kolejność odpowiednich symboli zastępczych w zapytaniu). Chociaż wymienione zastępcze uczynić kod bardziej gadatliwy, pozwalają one losowej kolejności wiązania.

Należy również pamiętać, że pomimo powszechnego złudzenia, nie „ `:` ” w kluczami jest wymagany.

Po wykonaniu można zacząć uzyskiwać dane przy użyciu wszystkich obsługiwanych metod, jak opisane w tym artykule.

sposoby wiązania

Przekazywanie danych do `execute()` (jak przedstawiono powyżej), należy rozważyć domyślnie i najbardziej wygodny sposób. Przy zastosowaniu tej metody, wszystkie wartości zostaną zobowiązane jako **ciągi** (za wyjątkiem `NULL` wartości, które zostaną wysłane do zapytania jak jest, czyli jak SQL `NULL`), ale większość czasu to wszystko jest w porządku i nie spowoduje żadnych problemów.

Jednak czasami lepiej jest ustawić typ danych jednoznacznie. Możliwe są przypadki:

- LIMIT klauzula w trybie emulacji lub innej klauzuli SQL, które po prostu nie może zaakceptować argument ciąg.
- złożonych zapytań z niebanalną planu kwerend, które mogą być dotknięte przez niewłaściwy typ operandu
- osobliwe typy kolumn, jak `BIGINT` i `BOOLEAN` które wymagają dokładnego argument typu na związanie (zauważ, że w celu związania wartości BIGINT z PDO :: PARAM_INT trzeba mysqlnd -na instalacji).

W takim przypadku wyraźnego wiązania muszą być stosowane, na którym masz do wyboru dwie funkcje, bindValue () i bindParam () . Pierwsza z nich musi być korzystne, ponieważ, w przeciwieństwie `bindParam()` to nie ma skutków ubocznych do czynienia.

Części zapytań można powiązać

Jest bardzo ważne, aby zrozumieć, które części zapytania można wiązać z użyciem przygotowanych sprawozdań, a które nie. W rzeczywistości, lista jest zdecydowanie krótki: tylko sznurek i literały liczbowe mogą być związane. Więc można powiedzieć, że tak długo, jak dane mogą być przedstawione w zapytaniu w postaci numerycznej lub w cudzysłowie dosłownym - to może być związany. We wszystkich innych przypadkach nie można użyć PDO przygotowane oświadczeń na wszystkich: ani identyfikatora lub listę oddzielonych przecinkami, lub część cudzysłowie dosłowne lub cokolwiek innego dowolna część zapytanie nie może być związany z użyciem przygotowane oświadczenie.

Obejścia najczęstszych przypadkach zastosowania można znaleźć w odpowiedniej części artykułu

Przygotowane sprawozdania. Wielokrotne wykonanie

Czasami można użyć sprawozdania przygotowanego dla wielokrotnej realizacji przygotowanego zapytania. Jest nieco szybciej niż kółko wykonywania tego samego zapytania, jak to robi parsowania kwerendy tylko raz. Funkcja ta byłaby bardziej przydatna, jeśli to było możliwe, aby wykonać instrukcję przygotowaną w innej instancji PHP. Ale niestety - nie jest. Tak, jesteś ograniczony do powtarzania tego samego zapytania tylko w obrębie tej samej instancji, który jest rzadko potrzebne w zwykłych skryptów PHP i która jest ograniczenie stosowania tej funkcji do powtarzających wkładek lub aktualizacji:

```
$data = [
    1 => 1000,
    5 =>  300,
    9 =>  200,
];
$stmt = $pdo->prepare('UPDATE users SET bonus = bonus + ? WHERE id = ?');
foreach ($data as $id => $bonus)
```

```
{
    $stmt->execute([$bonus, $id]);
}
```

Należy pamiętać, że funkcja ta jest nieco przereklamowane. Nie tylko jest to potrzebne zbyt rzadko o tym mówić, ale przyrost wydajności nie jest duża - parsowanie zapytanie jest *prawdziwy* szybko te czasy.

Należy pamiętać, że można dostać tylko tę zaletę, gdy tryb emulacji jest włączony **off** .

Uruchamianie SELECT INSERT, UPDATE lub DELETE

Przyjdź na ludzi. Nie ma absolutnie nic specjalnego w tych zapytań. Do PDO one wszystkie takie same. To nie ma znaczenia, który zapytań używasz.

Podobnie jak wykazano powyżej, co potrzebne jest do przygotowania zapytania zastępcze, a następnie uruchom go, wysyłając zmienne osobno. Albo do `DELETE` i `SELECT` zapytania procesu jest zasadniczo taka sama. Jedyną różnicą jest to (jak DML kwerendy nie zwraca żadnych danych), które można zastosować metodę łączenia, a tym samym wywołać `execute()` prawo wraz z `prepare()` :

```
$sql = "UPDATE users SET name = ? WHERE id = ?";
$pdo->prepare($sql)->execute([$name, $id]);
```

Jednakże, jeśli chcesz uzyskać liczbę wierszy, kod musi być takie same boresome trzy linie:

```
$stmt = $pdo->prepare("DELETE FROM goods WHERE category = ?");
$stmt->execute([$cat]);
$deleted = $stmt->rowCount();
```

Pobieranie danych z rachunku. dla każdego()

Sposób najbardziej prosty i bezpośredni, aby dostać wiele wierszy z oświadczeniem
byłaby `foreach()` pętla. Dzięki przesuwny interfejs, `PDOStatement` można powtórzyć w ciągu za pomocą `foreach()` operatora:

```
$stmt = $pdo->query('SELECT name FROM users');
foreach ($stmt as $row)
{
```

```
    echo $row['name'] . "\n";
}
```

Należy zauważyć, że metoda ta jest pamięć z dziećmi, ponieważ nie załadować wszystkie wiersze powstałe w pamięci, ale dostarcza je jeden po drugim (choć należy pamiętać, to problem).

Pobieranie danych z rachunku. sprowadzać()

Widzieliśmy już tę funkcję, ale weźmy się bliżej. To pobiera jeden wiersz z bazy danych, i przesuwa wewnętrzny wskaźnik w zestawie wyników, więc wynikające wywołania tej funkcji powróci wszystkimi wynikającymi rzędach jeden za drugim. Co sprawia, że metoda ta szorstka analogowo `mysql_fetch_array()` ale działa w nieco inny sposób: zamiast wielu oddzielnych funkcji (`mysql_fetch_assoc()` , `mysql_fetch_row()` itp), jest tylko jeden, ale jego zachowanie może być zmieniona przez parametr. Istnieje wiele pobierające tryby w PDO, i omówimy je później, ale oto kilka na przystawkę:

- `PDO::FETCH_NUM` Zwraca wymienione Tablica
- `PDO::FETCH_ASSOC` powraca asocjacyjnej
- `PDO::FETCH_BOTH` - oba powyższe
- `PDO::FETCH_OBJ` zwraca object
- `PDO::FETCH_LAZY` Pozwala wszystkie trzy (numerycznych asocjacyjny i przedmiot) metody bez narzutu pamięci.

Z powyższego można powiedzieć, że ta funkcja musi być stosowany w dwóch przypadkach:

1. Jeśli przewidywany jest tylko jeden rząd - aby dostać się, że tylko rząd. Na przykład,

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

   Daje jeden wiersz z oświadczeniem, w postaci tablicy asocjacyjnej.

2. Kiedy trzeba przetwarzać dane zwracane jakoś przed użyciem. W tym przypadku musi być prowadzony poprzez zwykły pętla, jak pokazany powyżej .

Innym sposobem jest użyteczny `PDO::FETCH_CLASS` , który może utworzyć obiekt danej klasy

```
$news = $pdo->query('SELECT * FROM news')->fetchAll(PDO::FETCH_CLASS, 'News');
```

spowoduje szereg wypełnioną obiektów wiadomości klasy, ustawiania właściwości klasy zwróconych wartości. Zauważ, że w tym trybie

- właściwości są ustawione *przed* wywołanie konstruktora
- dla wszystkich właściwości niezdefiniowanych `__set` metoda zostanie wywołana magia
- jeśli nie ma `__set` metody w klasie, to nowy obiekt zostanie utworzony

- Właściwości prywatne zostaną wypełnione, a także, co jest nieco nieoczekiwane, ale bardzo przydatny

Zauważ, że domyślnym trybem jest `PDO::FETCH_BOTH` , ale można go zmienić za pomocą `PDO::ATTR_DEFAULT_FETCH_MODE` opcji konfiguracji, jak pokazano na przykładzie połączenia. Tak więc, po zbiorze, może zostać pominięty przez większość czasu.

Powrót typy.

Dopiero gdy PDO jest zbudowany na mysqlnd *i* tryb emulacji jest **wyłączony** , a następnie powróci PDO `int` i `float` wartości z odpowiednich typów. Powiedzieć, czy tworzymy tabelę

```
create table typetest (string varchar(255), `int` int, `float` float, `null` int);
insert into typetest values('foo',1,1.1,NULL);
```

A następnie zapytać go z PDO mysqlnd oparte o emulacji wyłączony, wyjście będzie

```
array(4) {
  ["string"] => string(3) "foo"
  ["int"]    => int(1)
  ["float"]  => float(1.1)
  ["null"]   => NULL
}
```

W przeciwnym razie znane `mysql_fetch_array()` zachowanie nastąpi - wszystkie wartości zwracane jako łańcuchy z tylko `NULL` powrócił jako `NULL` .

Jeśli z jakiegoś powodu nie podoba to zachowanie i wolą starego stylu z łańcuchów i wartości null tylko, a następnie można użyć następującej opcji konfiguracji, aby go zastąpić:

```
$pdo->setAttribute(PDO::ATTR_STRINGIFY_FETCHES, true);
```

Zauważ, że dla `DECIMAL` typu ciąg jest zawsze zwrócony ze względu na charakter tego typu przeznaczonego do zatrzymywania dokładnej wartości, w przeciwieństwie celowo bez precyzyjnego FLOAT i dwukrotnie typów.

Pobieranie danych z rachunku. fetchColumn ()

Zgrabny pomocnik funkcja, która zwraca wartość pola przypalić zwróconych rzędu. Bardzo przydatne, gdy mamy do wyboru tylko jedno pole:

```php
// Getting the name based on id
$stmt = $pdo->prepare("SELECT name FROM table WHERE id=?");
$stmt->execute([$id]);
$name = $stmt->fetchColumn();

// getting number of rows in the table utilizing method chaining
$count = $pdo->query("SELECT count(*) FROM table")->fetchColumn();
```

Pierwsze dane z rachunku w dziesiątkach różnych formatach. fetchAll ()

To najbardziej interesująca funkcja, z większości funkcji zdumiewające. Głównie dzięki jednej istnienie może wywołać PDO otoki, jak ta funkcja może zautomatyzować wiele czynności wykonywanych ręcznie inaczej.

`PDOStatement::fetchAll()` Zwraca układ, który składa się **z wierszy** zwróconych w zapytaniu. Z tego faktu możemy zrobić dwa wnioski:

1. Funkcja ta nie powinna być stosowana, jeśli wiele wierszy został wybrany. W takim przypadku konwencjonalnych pętla Avenue być stosowane pobierania rzędach jeden po drugim, a nie wszystkie się je w tablicy na raz. „Wielu" oznacza więcej niż nadaje się do umieszczenia na stronie internetowej średniej.
2. Funkcja ta jest przydatna w większości nowoczesnych aplikacji internetowych, które nigdy nie wyprowadza dane od razu podczas pobierania, lecz przekazuje je do szablonu.

Byłbyś zaskoczony, jak w wielu różnych formatach, funkcja ta może zwrócić dane (i jak mało przeciętny użytkownik PHP wie o nich), kontrolowane przez `PDO::FETCH_*` zmiennych. Niektórzy z nich są:

Pierwsze zwykły tablicy.

Domyślnie ta funkcja zwróci tylko proste wyliczeniowy tablica składa się ze wszystkich zwracanych wierszy. Row formatowania stałe, takie jak `PDO::FETCH_NUM` , `PDO::FETCH_ASSOC` , `PDO::FETCH_OBJ` itp może zmienić format wiersza.

```php
$data = $pdo->query('SELECT name FROM users')->fetchAll();
var_export($data);
/*
array (
  0 => array('John'),
  1 => array('Mike'),
  2 => array('Mary'),
  3 => array('Kathy'),
)*/
```

Pierwsze kolumny.

Jest to często bardzo przydatne, aby uzyskać zwykły tablicę jednowymiarową prosto z zapytaniem, czy tylko jedna kolumna z wielu wierszy jest naciągane. Proszę bardzo:

```
$data = $pdo->query('SELECT name FROM users')->fetchAll(PDO::FETCH_COLUMN);
/* array (
  0 => 'John',
  1 => 'Mike',
  2 => 'Mary',
  3 => 'Kathy',
)*/
```

Pierwsze par klucz-wartość.

Format również niezwykle przydatna, gdy trzeba uzyskać te same kolumny, ale nie indeksowane przez numery w porządku, ale w innym zakresie. Tu idzie `PDO::FETCH_KEY_PAIR` stałe:

```
$data = $pdo->query('SELECT id, name FROM users')->fetchAll(PDO::FETCH_KEY_PAIR);
/* array (
  104 => 'John',
  110 => 'Mike',
  120 => 'Mary',
  121 => 'Kathy',
)*/
```

Pamiętaj, że masz do wyboru tylko dwie kolumny w tym trybie, z których pierwsza musi być niepowtarzalny.

Pierwsze wiersze indeksowane przez unikalnej dziedzinie

Tak samo jak powyżej, ale coraz nie jedną kolumnę, ale pełny wiersz, jeszcze indeksowane przez wyjątkowej ostrości, dzięki `PDO::FETCH_UNIQUE` stałej:

```
$data = $pdo->query('SELECT * FROM users')->fetchAll(PDO::FETCH_UNIQUE);
/* array (
  104 => array (
```

```
    'name' => 'John',
    'car' => 'Toyota',
  ),
  110 => array (
    'name' => 'Mike',
    'car' => 'Ford',
  ),
  120 => array (
    'name' => 'Mary',
    'car' => 'Mazda',
  ),
  121 => array (
    'name' => 'Kathy',
    'car' => 'Mazda',
  ),
)*/
```

Należy pamiętać, że pierwszą kolumnę wybraną muszą być unikalne (w tym zapytaniu zakłada się, że pierwsza kolumna jest id, ale mieć pewność, że wyraźnie lepiej lista).

Pierwsze wiersze pogrupowane według jakiejś dziedzinie

`PDO::FETCH_GROUP` będzie rzędy grupy do zagnieżdżonej tablicy, gdzie indeksy będzie unikatowe wartości z pierwszej kolumny, a wartości będą tablice podobne do tych zwróconych przez regularne `fetchAll()` . Poniższy kod, na przykład, będzie oddzielić chłopców od dziewcząt i umieścić je w różnych układach:

```
$data = $pdo->query('SELECT sex, name, car FROM users')->fetchAll(PDO::FETCH_GROUP);
array (
  'male' => array (
    0 => array (
      'name' => 'John',
      'car' => 'Toyota',
    ),
    1 => array (
      'name' => 'Mike',
      'car' => 'Ford',
    ),
  ),
  'female' => array (
```

```
    0 => array (
      'name' => 'Mary',
      'car' => 'Mazda',
    ),
    1 => array (
      'name' => 'Kathy',
      'car' => 'Mazda',
    ),
  ),
)
```

Tak, jest to idealne rozwiązanie dla tak popularne popytu, jak „imprez grupowych według daty” lub „grupy towarów według kategorii”. Niektóre przypadki użycia prawdziwe życie:

- Jak wielu wyników zapytania w celu zmniejszenia liczby zapytań?
- Zapisy Lista pogrupowane według nazwy kategorii

Inne tryby

Oczywiście, nie jest to `PDO::FETCH_FUNC` dla fanów funkcjonalnych programowania.

Inne tryby są dostępne wkrótce.

obsługa błędów. wyjątki

Chociaż istnieje kilka trybów obsługi błędów w PDO, jedyną właściwą jeden jest `PDO::ERRMODE_EXCEPTION` . Tak, jeden powinien zawsze ustawić go w ten sposób, albo przez dodanie tej linii po utworzeniu instancji PDO,

```
$dbh->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

lub jako opcja połączenia, jak pokazano w powyższym przykładzie. I to jest *wszystko* , czego potrzebujesz do podstawowego raportowania błędów.

Raportowanie błędów PDO

TL; DR:
Wbrew temu, co wszystkie inne samouczki powiedzieć, **nie trzeba się `try..catch` z operatorem, aby zgłosić błędy PDO** . Złapać wyjątek tylko jeśli masz scenariusz zabiegom innym niż tylko zgłoszenie go. W przeciwnym razie po prostu niech to bańka aż do obsługi całej witrynie (zauważ, że nie trzeba pisać jeden, istnieje podstawowa wbudowanej obsługi w PHP, który jest dość dobrze).

Długa rant na ten temat:

Pomimo powszechnego złudzenia, nigdy nie powinien wychwycić błędy, aby je zgłosić. Moduł (jak warstwa bazy danych) nie powinny zgłosić swoje błędy. Funkcja ta musi być przekazana do obsługi całej aplikacji. Wszystko, czego potrzebujemy jest podniesienie błąd (w postaci wyjątku) - co już zrobił. To wszystko. Nie należy „ *zawsze owinąć operacje PDO w* `try/catch` ” jak najbardziej popularnego poradnika z tutsplus zaleca. Wręcz przeciwnie, łowienie wyjątek powinien być raczej wyjątkowy przypadek (gra słów nie przeznaczonych).

W rzeczywistości nie ma nic szczególnego w PDO wyjątkami - są błędy takie same. Tak więc, trzeba je traktować dokładnie tak samo, jak inne błędy. Jeśli miałeś obsługi błędu przed, nie należy tworzyć jeden dedykowany dla PDO. Jeśli nie obchodzi - to wszystko w porządku zbyt, jak PHP jest dobry z podstawowej obsługi błędów i wyjątków przeprowadzi PDO w porządku.

Obsługa wyjątków jest jednym z problemów z tutoriali PDO. Zapoznaniu się z wyjątkami po raz pierwszy, gdy wychodząc z PDO, autorzy uważają wyjątki poświęconej tej biblioteki i zacząć pilnie (ale nieprawidłowo) Obsługa wyjątków dla PDO tylko. To jest bzdura. Jeśli ktoś nie zwracał szczególną uwagę na wszelkie wyjątki wcześniej, że nie powinna się zmienić swoje przyzwyczajenia do PDO. Jeśli ktoś nie używał `try..catch` wcześniej, należy je trzymać z tym, w końcu nauczyć się wykorzystywać wyjątki, a gdy jest to odpowiednie, aby je złapać.

Więc teraz można powiedzieć, że podręcznik PHP jest niewłaściwy, stwierdzając, że

> Jeśli aplikacja nie złapać rzucony wyjątek od konstruktora PDO, domyślna akcja podjęta przez silnik zend jest przerwanie skryptu i wyświetlić tylną śladu. Ten ślad powrotem będzie prawdopodobnie ujawniają szczegóły połączenia z bazą danych, w tym nazwę użytkownika i hasło.

Jednakże, **nie ma czegoś takiego jak „wyświetlanie z powrotem śladu”!** Co Zend silnik *naprawdę* robi jest po prostu przekształcić przechwycony wyjątek w fatalnym błędem. A potem to błąd krytyczny jest traktowany **jak każdy inny błąd** - tak będzie wyświetlany tylko wtedy, gdy odpowiednie `php.ini` dyrektywy jest ustawiony. Tak więc, chociaż może lub może nie złapać wyjątek, to nie ma absolutnie nic wspólnego z wyświetlania poufnych informacji, bo to **zupełnie inna konfiguracja ustawień** w odpowiedzi na to. Tak więc, nie złapać wyjątki PDO je zgłosić. Zamiast tego, prawidłowo skonfigurować serwer:

Na serwerze rozwoju wystarczy włączyć wyświetlanie błędów na:

```
ini_set('display_errors', 1);
```

Podczas gdy na przełomie serwerze produkcyjnym wyświetlanie błędów podczas logowania się na błędy:

```
ini_set('display_errors', 0);
ini_set('log_errors', 1);
```

- należy pamiętać, że *istnieją inne błędy, które nie powinny zostać ujawnione dla użytkownika, jak również.*

Łapanie wyjątków PDO

Może chcesz złapać błędy PDO tylko w dwóch przypadkach:

1. Jeśli piszesz otoki dla PDO, i chcesz poszerzyć info o błędzie z niektórych dodatkowych danych, takich jak ciąg kwerendy. W tym przypadku, złapać wyjątek, zebrać potrzebne informacje i **ponownie rzucać inny wyjątek** .
2. Jeśli masz **pewien scenariusz** do obsługi błędów w określonej części kodu. Oto kilka przykładów:
    - jeżeli błąd można obejść, można użyć try..catch do tego. Jednak nie przyzwyczajaj. Pusty haczyk w każdym aspekcie działa jako operator tłumienia błędów, a więc równie zła jest .
    - jeśli jest to działanie, które ma być podjęte w przypadku awarii, czyli transakcji wycofania.
    - Jeśli czekasz na konkretnego błędu w obsłudze. W tym przypadku, złapać wyjątek, sprawdzić, czy błąd jest jednym szukasz, a następnie obsłużyć ten jeden. W przeciwnym razie po prostu wyrzucić go ponownie - tak to będzie bubble do obsługi w zwykły sposób.

Na przykład:

```
try {
    $pdo->prepare("INSERT INTO users VALUES (NULL,?,?,?,?)")->execute($data);
} catch (PDOException $e) {
    if ($e->getCode() == 1062) {
        // Take some action if there is a key constraint violation, i.e. duplicate name
    } else {
        throw $e;
    }
}
```

Jednak ogólnie rzecz biorąc, nie ma dedykowanego leczenie wyjątki PDO jest zawsze potrzebna. W skrócie, aby mieć PDO błędy prawidłowo zgłoszone:

1. Ustaw PDO w trybie wyjątku.
2. Nie należy używać `try..catch` do zgłaszania błędów.
3. Skonfigurować PHP do prawidłowego raportowania błędów
    - na **żywo** zestawu miejscu `display_errors=off` i `log_errors=on`
    - na **rozwoju** witryny, może chcesz ustawić `display_errors=on`
    - Oczywiście, `error_reporting` musi być ustawiony na E_ALL w obu przypadkach

W rezultacie, będzie zawsze powiadamiany o wszystkich błędów baz danych bez jednej linii z dodatkowym kodem! Dalsze czytanie .

Uzyskiwanie liczba wierszy z PDO

Go nie potrzebował.

Chociaż PDO oferuje funkcję powrocie liczbę wierszy znalezionych przez zapytania `PDOstatement::rowCount()` , to rzadko potrzebne. Naprawdę.

Jeśli myślisz, że to koniec, widać, że jest to najbardziej nadużywane funkcja w internecie. Większość czasu jest używany, aby nie *liczyć* cokolwiek, ale jedynie jako flaga - wystarczy, aby zobaczyć, czy nie było żadnych danych zwracanych. Ale w takim przypadku masz samych danych! Po prostu swoje dane, korzystając z jednej `fetch()` lub `fetchAll()` - i będzie służyć jako takiej flagi wszystko w porządku! Powiedzieć, aby sprawdzić, czy istnieje użytkownik o takiej nazwie, wystarczy wybrać wiersz:

```
$stmt = $pdo->prepare("SELECT 1 FROM users WHERE name=?");
$stmt->execute([$name]);
$userExists = $stmt->fetchColumn();
```

Dokładnie to samo z coraz albo pojedynczy wiersz lub tablicę z wierszy:

```
$data = $pdo->query("SELECT * FROM table")->fetchAll();
if ($data) {
    // You have the data! No need for the rowCount() ever!
}
```

Pamiętaj, że tu nie trzeba się *liczyć* , rzeczywistą liczbę wierszy, lecz logiczną flagi. Więc masz go.

Nie wspominając o tym, że drugi najbardziej popularny przypadek użycia tej funkcji nigdy nie powinny być wykorzystywane w ogóle. Nigdy nie należy używać `rowCount()` do zliczania wierszy w bazie danych! Zamiast tego, trzeba zapytać bazę je policzyć, i zwraca wynik w **pojedynczym** rzędzie:

```
$count = $pdo->query("SELECT count(1) FROM t")->fetchColumn();
```

Jest to jedyny właściwy sposób.

W istocie:

- jeśli trzeba wiedzieć, jak wiele wierszy w tabeli, użyj `SELECT COUNT(*)` zapytania.
- jeśli chcesz wiedzieć, czy zapytanie zwróciło żadnych danych - sprawdzić, czy dane.
- jeśli trzeba jeszcze wiedzieć, ile wierszy zostało zwrócone przez jakiegoś zapytania (chociaż ja ledwo mogę sobie wyobrazić przypadek), a następnie można użyć `rowCount()` albo po prostu zadzwonić `count()` na tablicy zwróconej przez `fetchAll()` (jeśli dotyczy).

Tak więc można powiedzieć, że top odpowiedź na to pytanie na przepełnienie stosu jest w zasadzie bezcelowe i szkodliwe - wywołanie `rowCount()` nigdy nie mógł być podstawiony `SELECT count(*)` zapytania - ich celem jest zasadniczo różna, natomiast uruchomiony dodatkowy zapytanie tylko, aby uzyskać liczbę wierszy zwróconych przez inne zapytania ma absolutnie żadnego sensu.

Wierszy i wstawić identyfikator

PDO wykorzystuje tę samą funkcję dla powracających zarówno liczbę wierszy zwróconych przez SELECT i liczbę wierszy dotkniętych DML zapytań - `PDOstatement::rowCount()` . Dlatego, aby uzyskać liczbę wierszy, po prostu wywołać tę funkcję po wykonaniu zapytania.

Innym często zadawane pytanie jest spowodowane faktem, że MySQL nie aktualizuje wiersz, czy nowa wartość jest taka sama jak starego. Zatem liczba wierszy może się różnić od liczby rzędów dopasowanych klauzulą gdzie. Czasami wymagane jest, aby wiedzieć, ten ostatni numer.

Chociaż można powiedzieć `rowCount()` , aby powrócić liczbę wierszy dopasowanych zamiast wierszy ustawiając `PDO::MYSQL_ATTR_FOUND_ROWS` opcję true, ale, jak to jest opcja połączenia tylko, a więc nie można zmienić to zachowanie w trakcie wykonywania, trzeba będzie trzymać się tylko jeden tryb dla aplikacji, która może być utrudniona.

Niestety, nie ma odpowiednika PDO dla `mysql(i)_info()` funkcji, które wyjście może być łatwo analizowany i żądany numer znaleziony. Jest to jeden z drobnych wad PDO.

Auto wygenerowany identyfikator z sekwencji lub pola auto_inclement w mysql można uzyskać z PDO :: lastInsertId funkcji. Odpowiedzi na najczęściej zadawane pytania, „czy ta funkcja jest bezpieczny w użyciu w środowisku współbieżne?” jest pozytywna: tak, to jest bezpieczne. Będąc tylko interfejs do MySQL C API mysql_insert_id () funkcjonować to całkowicie bezpieczne.

Przygotowane sprawozdania i podobnie jak klauzula

Pomimo ogólnej łatwości PDO dotyczące użytkowania, istnieje kilka pułapek i tak, i mam zamiar wyjaśnić niektóre.

Jednym z nich jest użycie zastępczych z `LIKE` klauzulą SQL. Początkowo można by pomyśleć, że takie zapytania zrobi:

```
$stmt = $pdo->prepare("SELECT * FROM table WHERE name LIKE '%?%'");
```

ale wkrótce okazuje się, że spowoduje to błąd. Aby zrozumieć jego naturę trzeba zrozumieć, że, jak to zostało powiedziane powyżej , *symbol zastępczy muszą reprezentować kompletne dane tylko dosłowne* - ciąg lub numer mianowicie. I w żadnym wypadku nie może to stanowić zarówno część dosłowną lub jakąś dowolną część SQL. Tak więc, podczas pracy z podobnych, musimy przygotować nasz **kompletny dosłowne** pierwszy, a następnie wysłać go do kwerendy w zwykły sposób:

```
$search = "%$search%";
```

```php
$stmt  = $pdo->prepare("SELECT * FROM table WHERE name LIKE ?");

$stmt->execute([$search]);
$data = $stmt->fetchAll();
```

Przygotowane sprawozdania i klauzula

Tak jak to zostało powiedziane powyżej, nie jest możliwe, aby zastąpić dowolną część zapytania z zastępczy. A zatem, dla wartości oddzielony przecinkami, jak na `IN()` operatora SQL, trzeba utworzyć zestaw `?` s ręcznie i umieścić je w zapytaniu:

```php
$arr = [1,2,3];
$in  = str_repeat('?,', count($arr) - 1) . '?';
$sql = "SELECT * FROM table WHERE column IN ($in)";
$stm = $db->prepare($sql);
$stm->execute($arr);
$data = $stm->fetchAll();
```

Nie bardzo wygodne, ale w porównaniu do mysqli to *niezwykle* zwięzły .

W przypadku, gdy istnieją inne zastępcze w zapytaniu można użyć `array_merge()` funkcji dołączyć wszystkie zmienne w jednej tablicy, dodać inne zmienne w postaci tablic, w kolejności ich występowania w zapytaniu:

```php
$arr = [1,2,3];
$in  = str_repeat('?,', count($arr) - 1) . '?';
$sql = "SELECT * FROM table WHERE foo=? AND column IN ($in) AND bar=? AND baz=?";
$stm = $db->prepare($sql);
$params = array_merge([$foo], $arr, [$bar, $baz]);
$stm->execute($params);
$data = $stm->fetchAll();
```

W przypadku korzystania z nazwanych zastępcze, kod będzie trochę bardziej skomplikowane, bo trzeba stworzyć sekwencję wymienionych zastępcze, np `:id0,:id1,:id2` . Więc kod byłoby:

```php
// other parameters that are going into query
$params = ["foo" => "foo", "bar" => "bar"];

$ids = [1,2,3];
$in = "";
```

```
foreach ($ids as $i => $item)
{
    $key = ":id".$i;
    $in .= "$key,";
    $in_params[$key] = $item; // collecting values into key-value array

}
$in = rtrim($in,","); // :id0,:id1,:id2

$sql = "SELECT * FROM table WHERE foo=:foo AND id IN ($in) AND bar=:bar";
$stm = $db->prepare($sql);
$stm->execute(array_merge($params,$in_params)); // just merge two arrays
$data = $stm->fetchAll();
```

Na szczęście dla wymienionych zastępcze nie musimy przestrzegać ścisłego porządku, więc możemy połączyć nasze tablice w dowolnej kolejności.

Przygotowane oświadczenia i nazwy tabel

Na przepełnienie stosu Widziałem przytłaczającą liczbę użytkowników PHP wykonawczych najbardziej śmiertelny kod PDO , myśląc, że tylko wartości danych muszą być chronione. Ale oczywiście nie jest.

Niestety, nie ma PDO zastępczy dla identyfikatorów (stół i nazwy pól), a więc programista musi ręcznie formatować je.

Dla **mysql** formatować identyfikator, wykonaj te dwie reguły:

- Identyfikator ująć w backticks.
- Ucieczka odwrócone, pojedyncze apostrofy wewnątrz przez ich podwojenie.

więc kod byłoby:

```
$table = "`".str_replace("`","``",$table)."`";
```

Po takim formatowania, jest to bezpieczne, aby wstawić `$table` zmienną do zapytania.

Dla innych baz zasady będą różne, ale ważne jest, aby zrozumieć, że przy użyciu tylko ograniczniki nie wystarczy - należy ograniczniki sami uciekli.

Ważne jest również, aby zawsze sprawdzić dynamiczne identyfikatory na liście dozwolonych wartości. Oto krótki przykład:

```
$orders  = ["name","price","qty"]; //field names
$key     = array_search($_GET['sort'],$orders); // see if we have such a name
$orderby = $orders[$key]; //if not, first one will be set automatically. smart enuf :)
$query   = "SELECT * FROM `table` ORDER BY $orderby"; //value is safe
```

Lub rozszerzenie tego podejścia dla instrukcji INSERT / UPDATE (jak MySQL obsługuje SET dla obu),

```
$data = ['name' => 'foo', 'submit' => 'submit']; // data for insert
$allowed = ["name", "surname", "email"]; // allowed fields
$values = [];
$set = "";
foreach ($allowed as $field) {
    if (isset($data[$field])) {
        $set.="`".str_replace("`", "``", $field)."`". "=:$field, ";
        $values[$field] = $data[$field];
    }
}
$set = substr($set, 0, -2);
```

Kod ten będzie wytwarzać prawidłową sekwencję operatora zestaw, który będzie zawierał jedynie dozwolone pola i zastępczych tak:

```
`name`=:foo
```

a także `$values` tablicę dla `execute()` , które mogą być używane w ten sposób

```
$stmt = $pdo->prepare("INSERT INTO users SET $set");
$stmt->execute($values);
```

Tak, to wygląda bardzo brzydki, ale to wszystko PDO może zaoferować.

Problem z klauzuli LIMIT

Kolejny problem wiąże się z SQL `LIMIT` klauzuli. Gdy w trybie emulacji (która jest domyślnie włączona), PDO zastępuje zastępcze z rzeczywistych danych, zamiast wysyłać go oddzielnie. Oraz „leniwa” wiązanie (za pomocą tablicy w `execute()`), PDO traktuje każdy parametr jako ciąg znaków. W rezultacie przygotowane `LIMIT ?,?` zapytanie się `LIMIT '10', '10'` co jest nieprawidłowe składnia powoduje kwerendy nie.

Istnieją dwa rozwiązania:

Jeden obrót emulację off (jak MySQL można sortować wszystkie elementy zastępcze prawidłowo). Aby to zrobić można uruchomić ten kod:

```
$conn->setAttribute( PDO::ATTR_EMULATE_PREPARES, false );
```

I parametry mogą być przechowywane w `execute()` :

```
$conn->setAttribute( PDO::ATTR_EMULATE_PREPARES, false );
$stmt = $pdo->prepare('SELECT * FROM table LIMIT ?, ?');
$stmt->execute([$offset, $limit]);
$data = $stmt->fetchAll();
```

Innym sposobem byłoby powiązać te zmienne wyraźnie podczas ustawiania odpowiedniego typu param:

```
$stmt = $pdo->prepare('SELECT * FROM table LIMIT ?, ?');
$stmt->bindParam(1, $offset,PDO::PARAM_INT);
$stmt->bindParam(2, $limit,PDO::PARAM_INT);
$stmt->execute();
$data = $stmt->fetchAll();
```

Jedna szczególna rzeczą `PDO::PARAM_INT` : z jakiegoś powodu nie wymusza casting typu. W ten sposób, wykorzystując go na numer, który ma typ string spowoduje wspomniany błąd:

```
$stmt = $pdo->prepare("SELECT 1 LIMIT ?");
$stmt->bindValue(1, "1", PDO::PARAM_INT);
$stmt->execute();
```

Ale zmiany `"1"` w tym przykładzie do `1` - i wszystko pójdzie gładko.

transakcje

Aby skutecznie przeprowadzić transakcję, trzeba upewnić się, że tryb błędu jest ustawiona na wyjątki i dowiedz się trzy metody kanonicznych:

- `beginTransaction()` aby rozpocząć transakcję
- `commit()` do popełnienia jednego
- `rollback()` anulować wszystkie zmiany wprowadzone od początku transakcji.

Wyjątkiem są niezbędne dla transakcji, ponieważ można je złapać. Tak więc w przypadku, gdy jeden z zapytaniami zawiodły, wykonanie zostanie zatrzymany i przeniósł się prosto do bloku catch, gdzie cała transakcja zostanie wycofana.

Więc typowym przykładem będzie jak

```
try {
    $pdo->beginTransaction();
    $stmt = $pdo->prepare("INSERT INTO users (name) VALUES (?)");
    foreach (['Joe','Ben'] as $name)
    {
        $stmt->execute([$name]);
    }
    $pdo->commit();
}catch (Exception $e){
    $pdo->rollback();
    throw $e;
}
```

Proszę zwrócić uwagę na następujące ważne rzeczy:

- trzeba złapać `Exception` , a nie `PDOException` , jak to nie ma znaczenia, co szczególnie wyjątek przerwana wykonanie.
- należy ponownie wyjątek po wycofania, aby otrzymywać powiadomienia o problemie w zwykły sposób.
- również upewnić się, że silnik tabela obsługuje transakcje (tj dla MySQL to powinno być InnoDB, nie MyISAM).

Wywoływanie procedur przechowywanych w PDO

Jest jedna rzecz, o wszelkich procedur przechowywanych w potyka programatora Po pierwsze: każda procedura przechowywana zawsze zwraca **jeden dodatkowy zestaw wyników** : jeden (lub wielu) wyników z rzeczywistymi danymi i tylko jeden pusty. Co oznacza, że w przypadku próby wywołania procedury, a następnie przystąpić do innego zapytania, a następnie **„Nie można wykonać zapytania podczas gdy inne niebuforowane zapytania są aktywny”** wystąpi błąd, bo trzeba usunąć ten dodatkowy pusty wynik pierwszego. Tak więc, po wywołaniu procedury przechowywanej, który jest przeznaczony do powrotu tylko jeden zestaw wyników, zadzwoń `PDOStatement::nextRowset()` raz (oczywiście po pobraniu wszystkich zwracanych danych z oświadczeniem, czy zostanie ona odrzucona):

```
$stmt = $pdo->query("CALL bar()");
$data = $stmt->fetchAll();
$stmt->nextRowset();
```

Natomiast dla procedur składowanych powracających wiele wynik ustawia zachowanie będzie taka sama jak w przypadku wielu zapytań wykonanie :

```
$stmt = $pdo->prepare("CALL foo()");
$stmt->execute();
do {
    $data = $stmt->fetchAll();
    var_dump($data);
} while ($stmt->nextRowset() && $stmt->columnCount());
```

Jednak, jak widać tutaj jest kolejna sztuczka muszą być stosowane: należy pamiętać, że zestaw dodatkowy wynik? Jest to więc w zasadzie *pusty* , że nawet próba pobrać z niej będzie produkować błąd. Więc nie możemy używać tylko `while ($stmt->nextRowset())` . Zamiast tego, musimy sprawdzić również na pusty wynik. Dla którego celem `PDOStatement::columnCount()` jest po prostu doskonała.

Ta cecha jest jedną z zasadniczych różnic między starym wew mysql i nowoczesnych bibliotek: po wywołaniu procedury przechowywanej z `mysql_query()` nie było sposobu, aby kontynuować pracę z tego samego połączenia, ponieważ nie istnieje `nextResult()` funkcja `mysql ext` . Trzeba było zamknąć połączenie i następnie otworzyć nowe ponownie w celu uruchomienia innych zapytań po wywołaniu procedury przechowywanej.

Wywołanie procedury przechowywanej jest to rzadki przypadek, gdzie `bindParam()` zastosowanie jest uzasadnione, ponieważ jest to jedyny sposób, aby obsługiwać `OUT` i `INOUT` parametry. Przykład można znaleźć w odpowiednim ręcznym rozdziału . Jednak **dla mysql nie działa.** Trzeba uciekać się do zmiennej SQL oraz dodatkowe połączenia .

Uruchamianie wielu zapytań z PDO

Gdy w trybie emulacji , PDO można uruchomić mutiple zapytań w tej samej instrukcji, albo poprzez zapytania () lub `prepare()/execute()` . Aby uzyskać dostęp wynik konsekwencji zapytaniami trzeba użyć `PDOStatement::nextRowset()` :

```
$stmt = $pdo->prepare("SELECT ?;SELECT ?");
$stmt->execute([1,2]);
do {
    $data = $stmt->fetchAll();
    var_dump($data);
} while ($stmt->nextRowset());
```

Wewnątrz tej pętli będziesz w stanie zebrać wszystkie informacje związane z każdym zapytaniem, jak wierszy, automatycznie wygenerowany identyfikator lub wystąpiły błędy.

Ważne jest, aby zrozumieć, że w punkcie `execute()` PDO zgłosi błąd **tylko pierwszego zapytania** . Ale jeśli błąd wystąpił w żadnej z konsekwencji zapytań, aby uzyskać ten błąd trzeba iteracyjne nad wynikami. Pomimo pewnych ignorantów opinii , PDO nie może i nie powinien zgłaszać wszystkie błędy na raz. Niektórzy ludzie po prostu nie może pojąć tego problemu w całości, a nie rozumieją, że komunikat o błędzie nie

jest tylko wynikiem z zapytania. Nie może być zwrócony zbiór danych lub niektóre metadane jak insert id. Aby uzyskać te, trzeba iteracyjne nad wynikowych, jeden po drugim. Jednak, aby móc natychmiast wyrzucić błąd, PDO musiałby automatycznie iteracji, a tym samym **odrzucić pewne rezultaty** . Co byłoby oczywiste bzdury.

W przeciwieństwie do `mysqli_multi_query()` PDO nie uczynić asynchroniczne wywołanie, więc nie można „ogień i zapomnij” - masowego wysyłania zapytań do MySQL i ścisłym związku, PHP będzie czekać, aż ostatnie zapytanie zostanie wykonany.

Tryb emulacji. PDO :: ATTR_EMULATE_PREPARES

Jednym z najbardziej kontrowersyjnych opcji konfiguracyjnych PDO `PDO::ATTR_EMULATE_PREPARES` . Co to robi? PDO można uruchomić zapytań na dwa sposoby:

1. Można go używać **prawdziwego** lub natywną przygotowane oświadczenie:
   Podczas przygotowania () jest wywoływana, zapytanie zastępcze zostanie wysłane do bazy danych, jak jest, ze wszystkie znaki zapytania można umieścić w (w przypadku nazwanych zastępcze są używane, są one podstawionej s? jak dobrze), podczas gdy rzeczywiste dane przechodzi później, kiedy execute () jest wywoływana.
2. Można go używać **emulowane** przygotowane oświadczenie, gdy zapytanie jest wysyłane do mysql jako właściwego SQL, wszystkie dane na swoim miejscu, **odpowiednio sformatowane** . W tym przypadku tylko jeden obie strony do bazy dzieje, z `execute()` rozmowy. Dla niektórych kierowców (w tym MySQL) tryb emulacji jest włączona `ON` domyślnie.

Obie metody ma swoje wady i zalety, ale i - muszę podkreślić na nim - obie są **równie bezpieczne** , jeżeli są stosowane prawidłowo. Pomimo dość atrakcyjne tonem popularnego artykułu na przepełnienie stosu , w końcu mówi, że **jeśli używasz obsługiwane wersje PHP i MySQL prawidłowo, są w 100% bezpieczne** . Wszystko co musisz zrobić, to ustawić kodowanie w DSN, jak to pokazano w przykładzie powyżej , a Twoje emulowane Przygotowane sprawozdania będą tak bezpieczne, jak prawdziwych.

Zauważ, że gdy używany jest tryb natywny, **dane nigdy nie pojawi się w zapytaniu** , który jest analizowany przez silnik jak jest, ze wszystkimi zastępcze w miejscu. Jeśli szukasz do Mysql dziennika zapytań do przygotowanego zapytania, musisz zrozumieć, że to jest po prostu sztuczny zapytania, który został stworzony wyłącznie do logowania cel, ale nie prawdziwy, który został wykonany.

Inne problemy z trybu emulacji w następujący sposób:

Gdy tryb emulacji jest włączony

można użyć poręczną funkcję nazwanych sporządzanych sprawozdań - symbol zastępczy o takiej samej nazwie można wykorzystać dowolną ilość razy w tej samej kwerendy, podczas gdy odpowiednia zmienna musi być związana tylko raz. Z jakiegoś niejasnego powodu ta funkcja jest wyłączona, gdy tryb emulacji jest wyłączona:

```
$stmt = $pdo->prepare("SELECT * FROM t WHERE foo LIKE :search OR bar LIKE :search");
$stmt->execute(['search'] => "%$search%");`
```

Również, gdy jest emulacja `ON` , PDO jest w stanie uruchomić wiele zapytań w jednym przygotowanym oświadczeniu .

Ponadto, jak rodzime Przygotowane sprawozdania obsługują tylko niektóre rodzaje zapytań, można uruchomić kilka zapytań z przygotowanych sprawozdań tylko wtedy, gdy jest emulacja `ON` . Poniższy kod zwróci nazwy tabel w trybie emulacji i błędów inaczej:

```
$stmt = $pdo->prepare("SHOW TABLES LIKE ?");
$stmt->execute(["%$name%"]);
var_dump($stmt->fetchAll());
```

Gdy tryb emulacji jest wyłączony

Można nie przejmować typów parametrów, jak MySQL będzie uporządkować wszystkie typy prawidłowo. Tak więc, nawet łańcuch może być związany z wartości drogi, jak to odnotowano w odpowiednim rozdziału .

Również ten tryb pozwoli wykorzystać przewagę pojedynczego przygotowania-wielokrotnego wykonywania funkcji.

Trudno zdecydować, który tryb ma być korzystna, ale ze względu na użyteczność wolałbym włączyć go `OFF` , aby uniknąć kłopotów z `LIMIT` klauzuli. Inne kwestie mogą być uważane za nieistotne w porównaniu.

Mysqlnd i buforowane zapytań. Ogromne zbiory danych.

Ostatnio wszystkie rozszerzenia PHP, które współpracują z bazy mysql zostały zaktualizowane w oparciu o biblioteki niskiego poziomu o nazwie `mysqlnd` , która zastąpiła starą `libmysql` klienta. Tak więc pewne zmiany w zachowaniu PDO, w większości opisane powyżej i jeden, który następuje:

Jest jedna rzecz, o nazwie buforowane zapytań . Chociaż prawdopodobnie nie zauważył go uzywasz im całą drogę. Niestety, tutaj są złe wieści dla ciebie: w przeciwieństwie do starszych wersji PHP, gdzie były praktycznie stosując buforowane zapytań za darmo, nowoczesne wersje zbudowane na sterowniku mysqlnd nie pozwoli Ci to zrobić już:

> Podczas korzystania libmysqlclient jako limit pamięci bibliotekę PHP nie będzie liczyć pamięć używaną dla zestawów wyników, chyba że dane są pobierane do zmiennych PHP. **Z mysqlnd pamięci stanowiły będą obejmować pełny zestaw wyników.**

Cała sprawa jest o wynikowego, który stoi do wszystkich danych znalezionych w zapytaniu.

Gdy SELECT zapytania zostanie wykonany, istnieją dwa sposoby, aby osiągnąć wyniki w skrypcie: buforowane i niebuforowane jeden. Kiedy stosowana jest metoda buforowane, wszystkie dane zwracane przez zapytanie **zostanie skopiowany do pamięci skryptu** naraz. W trybie niebuforowanym serwer bazy danych karmi znalezionego rzędach jeden za drugim.

Więc można powiedzieć, że w trybie buforowanej zestaw wyników jest zawsze obciążając się w pamięci na serwerze , *nawet jeśli pobieranie nie rozpoczęło w ogóle* . Dlatego też nie jest wskazane, aby wybrać ogromnych zbiorów danych, jeśli nie trzeba wszystkie dane z niego.

Niemniej jednak, gdy użyto starych klientów libmysql oparte, problem ten nie przeszkadzało uers PHP zbyt dużo, ponieważ pamięć spożywane przez wynikowego nie liczyć w The `memory_get_usage()` a `memory_limit` .

Ale z mysqlnd sprawy się zmieniły, a resultset zwracany przez buforowanej zapytania będą liczyły się zarówno `memory_get_usage()` i `memory_limit` , bez względu na to, w jaki sposób wybrać, aby uzyskać wynik:

```
$pdo->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, FALSE);
$stmt = $pdo->query("SELECT * FROM Board");
$mem = memory_get_usage();
while($row = $stmt->fetch());
echo "Memory used: ".round((memory_get_usage() - $mem) / 1024 / 1024, 2)."M\n";

$pdo->setAttribute(PDO::MYSQL_ATTR_USE_BUFFERED_QUERY, TRUE);
$stmt = $pdo->query("SELECT * FROM Board");
$mem = memory_get_usage();
while($row = $stmt->fetch());
echo "Memory used: ".round((memory_get_usage() - $mem) / 1024 / 1024, 2)."M\n";
```

da ci (za moich danych)

```
Memory used: 0.02M
Memory used: 2.39M
```

co oznacza, że z zapytaniem zbuforowaną pamięć jest spożywany **nawet jeśli pobieranie rzędach jeden za drugim!**

Tak więc, należy pamiętać, że jeśli wybranie naprawdę ogromną ilość danych, zawsze ustawiony `PDO::MYSQL_ATTR_USE_BUFFERED_QUERY` na `FALSE` .

Oczywiście, istnieją pewne wady, dwie nieznaczne:

1. Przy zapytaniu niebuforowanym nie można użyć `rowCount()` metody (co jest bezużyteczne, jak dowiedzieliśmy się powyżej)
2. Przenoszenie (poszukiwanie) aktualny wewnętrzny wskaźnik resultset iz powrotem (który jest bezużyteczny, jak również).

A raczej ważna:

1. Podczas kwerendy unbufferered jest aktywny, nie można wykonać żadnej innej kwerendy. Tak więc korzystać z tego trybu mądrze.