Fetchowanie danych z bazy w PDO

Na egzaminie INF.03 bez problemu możemy używać rozszerzenia *mysqli* do łączenia się z bazą danych – i zdecydowana większość osób rzeczywiście tak postępuje. Moduł *mysqli* to jak najbardziej dobre i akceptowalne rozwiązanie, które pozwala nam nawet miksować funkcje (podejście proceduralne) z kodem obiektowym i jest zupełnie wystarczające do egzaminu.

Jeśli jednak chcemy podejść do zadania egzaminacyjnego nowocześniej i bardziej profesjonalnie, to warto poznać również współpracę z bazą danych realizowaną w rozszerzeniu *PDO*, czyli *PHP Data Objects*. To właśnie temu modułowi poświęcimy cały niniejszy artykuł – potraktuj tę lekcję jako materiał dodatkowy dla osób bardziej ambitnych – jeśli zamierzasz i tak używać *mysqli* na egzaminie, to tej lekcji koniecznie przepracować nie trzeba.

To wyjaśniwszy, przyjrzymy się teraz jak współpracować z bazą MySQL w PHP przy użyciu *PDO*. Poznamy różne metody *fetchowania* rekordów i zobaczymy, jak różni się sposób ich wyjmowania, w zależności od wybranego trybu zwracania wyników. Gorąco polecam w tym miejscu także zapoznanie się z tutorialem video: Poznajemy bibliotekę PDO.

Rozszerzenie *mysqli* kontra *PDO*

PHP Data Objects to biblioteka, która współpracuje z wieloma silnikami baz danych, nie tylko MySQL/MariaDB. Dzięki temu nasz kod staje się bardziej uniwersalny i łatwiejszy w sytuacji ewentualnej migracji zaplecza bazodanowego do nowego silnika. Różni się też tym, że wymaga wyłącznie składni obiektowej, co może początkowo wydawać się trudniejsze, lecz bardzo szybko przekonujemy się, że to świetna droga ku profesjonalizacji kodu.

Właściwość biblioteki	mysqli	PDO
Obsługiwane bazy danych	Tylko MySQL	Wiele: MySQL, SQLite, PostgreSQL, …
Styl kodu	Proceduralny i obiektowy	Tylko obiektowy
Wsparcie dla tzw. prepared statements	Tak	Tak
Fetchowanie jako tablica numeryczna albo asocjacyjna lub jako obiekt	Tak	Tak
Możliwość migracji strony do innych systemów bazodanowych	Niska	Wysoka
Składnia dla początkujących	Bardziej przystępna	Nieco trudniejsza
Skalowalność projektu	Dobra	Świetna

Co lepsze na egzamin INF.03?

Na egzaminie INF.03 bardzo często korzystamy z MySQL/MariaDB, a więc możemy wybrać zarówno rozszerzenie *mysqli*, jak i *PDO*. Jeśli zadanie nie narzuci nam konkretnej biblioteki, to zasadniczo *mysqli* pozwoli nam szybciej napisać kod – zwyczajnie jest on prostszy w zapisie. Natomiast jeśli chcemy pokazać profesjonalne podejście i lepsze zrozumienie zasad bezpieczeństwa i skalowalności – *PDO* będzie lepszym wyborem dla osób ambitnych i pasjonatów.

Połączenie z bazą danych przy użyciu PDO

Oczywiście najpierw trzeba połączyć się z bazą danych – załóżmy, że mamy już lokalnie w XAMPP zaimportowaną bazę o nazwie

`vod` (wypożyczalnia filmów – *ang. video on demand*). Poniżej przedstawiamy kod odpowiedzialny za nawiązanie połączenia i obsługę ewentualnych błędów:

```php
<?php
$host = "localhost";
$dbname = "vod";
$user = "root";
$pass = "";
try {
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $user, $pass);
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Połączono z bazą!";
} catch (PDOException $e) {
    die("Błąd połączenia: " . $e->getMessage());
}
?>
```

Co robią poszczególne części kodu? Najpierw tworzymy cztery zmienne z parametrami połączenia – wiadomo. Następnie wewnątrz bloku

`try..catch` próbujemy utworzyć nowy obiekt klasy `new PDO(...)` reprezentujący połączenie. Dodatkowo ustawiamy tryb zgłaszania błędów: `$pdo->setAttribute(...)`. Dzięki temu jeśli wystąpi błąd, zostanie on obsłużony jako wyjątek oraz wyświetlimy komunikat z błędem: `$e->getMessage();`.

Pobieranie danych – jak to działa?

Aby pobrać dane z bazy, wykonujemy oczywiście klasyczne zapytanie

`SELECT`. Przykładowo, jeśli chcemy pobrać wszystkie rekordy z tabeli o nazwie `filmy`, użyjemy następującego kodu:

```php
$sql = "SELECT * FROM filmy";
$stmt = $pdo->query($sql);
```

Na tym etapie zmienna

`$stmt` (*ang. statement – wyrażenie*) przechowuje obiekt zapytania, z którego będziemy mogli "wyciągać" dane w różnych formatach – za chwilę je poznamy.

Pobieranie danych w tablicy numerycznej – FETCH_NUM

To sposób, który zwraca dane jako tablicę ponumerowaną – czyli każda kolumna z bazy będzie posiadać swój indeks liczbowy (0, 1, 2 itd.), analogicznie jak to ma miejsce w klasycznym fetchowaniu w *mysqli* funkcją

`mysqli_fetch_row()`. Kolejność kolumn w zapytaniu ma tutaj znaczenie – musimy pamiętać, która kolumna odpowiada któremu indeksowi.

```
<?php
while ($row = $stmt->fetch(PDO::FETCH_NUM)) {
    echo "ID: " . $row[0] . " | Tytuł: " . $row[1] . " | Kraj: " . $row[2] . " |
Gatunek: " . $row[3] . " | Cena: " . $row[4] . " zł <br>";
}
?>
```

Pobieranie danych jako tablica asocjacyjna – FETCH_ASSOC

Kolejny często stosowany sposób – dane zwracane są w postaci tablicy asocjacyjnej, a więc poszczególne wartości dostępne są po nazwach kolumn (takich jak w tabeli w bazie danych). Ta metoda odpowiada oczywiście w *mysqli* funkcji

`mysqli_fetch_assoc()`. Dzięki nazwom aoscjacyjnym nie trzeba sprawdzać jakie indeksy odpowiadają którym kolumnom w zapytaniu, tylko odwołujemy się do nich po nazwach np. `$row["Tytul"]`:

```
<?php
while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "ID: " . $row["ID_filmu"] . " | Tytuł: " . $row["Tytul"] . " | Kraj: " .
$row["Kraj_produkcji"] . " | Gatunek: " . $row["Gatunek"] . " | Cena: " .
$row["Cena_w_zl"] . " zł <br>";
}
?>
```

Pobieranie danych jako tablica mieszana – FETCH_BOTH

Ten tryb zwraca dane jako tablicę mieszaną – mamy jednocześnie dostęp do danych zarówno przez indeksy liczbowe, jak i przez nazwy kolumn. Takie podejście odpowiada w *mysqli* funkcji

`mysqli_fetch_array()` i może okazać się przydatne w przypadku rozbudowanych skryptów, w których zarówno iterowanie po indeksach jak i używanie nazw kolumn okaże się użyteczne:

```
<?php
while ($row = $stmt->fetch(PDO::FETCH_BOTH)) {
    echo "ID: " . $row["ID_filmu"] . " (" . $row[0] . ") | ";
    echo "Tytuł: " . $row["Tytul"] . " (" . $row[1] . ") | ";
    echo "Kraj: " . $row["Kraj_produkcji"] . " (" . $row[2] . ") | ";
    echo "Gatunek: " . $row["Gatunek"] . " (" . $row[3] . ") | ";
    echo "Cena: " . $row["Cena_w_zl"] . " zł (" . $row[4] . " zł) <br>";
}
?>
```

Pobieranie danych jako obiekt – FETCH_OBJ

Jeśli lubimy pracować z danymi w sposób obiektowy, to idealnym wyborem będzie

`PDO::FETCH_OBJ`. Odpowiednikiem w *mysqli* jest metoda `fetch_object()`. Każdy rekord z bazy zostanie potraktowany jako obiekt, a poszczególne kolumny będą dostępne jako jego właściwości:

```
<?php
```

```php
while ($row = $stmt->fetch(PDO::FETCH_OBJ)) {
    echo "ID: " . $row->ID_filmu . " | Tytuł: " . $row->Tytul . " | Kraj: " . $row->Kraj_produkcji . " | Gatunek: " . $row->Gatunek . " | Cena: " . $row->Cena_w_zl . " zł <br>";
}
?>
```

Który sposób *fetchowania* danych wybrać?

Oczywiście zależy to stricte od preferencji programisty. W praktyce ludzie najczęściej korzystają z

`FETCH_ASSOC`, ponieważ:

- jest to sposób najbardziej czytelny – po nazwach "widać" co w kodzie np. wypisujemy w danym miejscu,
- nie musimy pamiętać jakie indeksy kolumn odpowiadają którym kolumnom,

Jeśli jednak chcemy po tych szufladach tablicy iterować, to zastosujemy podejście numeryczne, a gdy zależy nam na stylu obiektowym albo planujemy pisać klasy modelujące dane z bazy – wówczas wybierzemy

`FETCH_OBJ`.

Podsumowanie

- PDO to nowoczesne i bezpieczne podejście do pracy z bazą danych w PHP.
- W przeciwieństwie do *mysqli*, *PDO* wspiera wiele różnych silników bazodanowych – nie tylko MySQL/MariaDB. Możemy zatem przenieść nasz kod do innej bazy bez dużych zmian — świetna opcja na przyszłość w realnym projekcie.
- W *PDO* – analogicznie jak w *mysqli* – możemy pobierać dane z bazy do tablic indeksowanych, asocjacyjnych, mieszanych lub obiektów. Wybranie sposobu zależy od preferencji programisty, no chyba że zadanie zmusi nas do wykorzystania konkretnego sposobu.
- Warto poznać docelowo obie biblioteki – *mysqli* jako start, *PDO* jako rozwiązanie długofalowe i docelowe.
- Na INF.03 oba sposoby są poprawne (w *mysqli* kod tworzy się nieco szybciej), chyba że zadanie wymusi użycie któregoś z rozszerzeń.
- Jeśli tylko czas na to pozwala – preferujmy *PDO* w projektach końcowych lub serwisach online.