Nawiązanie i zamykanie połączenia PHP z bazą danych

Połączenie skryptu PHP z bazą danych MySQL celem wykonywania zapytań przypomina nieco rozmowę telefoniczną. Po pierwsze trzeba nam nawiązać połączenie z rozmówcą. W świecie telefonii komórkowej aby porozmawiać z konkretnym człowiekiem wystarczy użyć jego numeru telefonu, który jednoznacznie go w systemie identyfikuje. Natomiast w internecie istnieją oczywiście całe miliony serwerów z zainstalowaną usługą MySQL, więc do nawiązania połączenia potrzebne będzie nieco więcej informacji – zazwyczaj są to cztery dane dostępowe:

1. **Nazwa hosta** – jest to adres IP komputera z zainstalowaną usługą MySQL, który przechowuje bazę danych do której chcemy się podłączyć. Na serwerze lokalnym XAMPP jest to adres `127.0.0.1` albo identyfikator: `localhost`
2. **Login konta w usłudze MySQL** – stworzony na serwerze użytkownik, które posiada stosowne uprawnienia do wykonywania zapytań – na serwerze lokalnym XAMPP jest to użytkownik o nazwie: root który oczywiście wszelkie uprawnienia posiada
3. **Hasło dostępowe do konta MySQL** – wyjątkowo w XAMPP jest to pusty łańcuch – nie może więc zawierać żadnych znaków (nawet spacji). Oczywiście w internecie hasło do bazy zawsze istnieje, jedynie na symulowanym serwerze XAMPP możemy go dla prostoty nie stosować.
4. **Nazwa bazy danych** – ponieważ usługa MySQL na tym hoście może przechowywać bardzo wiele baz danych, to określamy na której konkretnie bazie chcemy wykonywać nasze zapytania SQL

Funkcja PHP z biblioteki

`mysqli`, która nawiązuje aktywne połączenie z serwerem bazodanowym nazywa się `mysqli_connect()` i używa kolejno właśnie tych czterech parametrów:

```
$connection = mysqli_connect("localhost", "root", "", "baza123") or die("Błąd połączenia!");
```

Wydaje się to bardzo proste, jednak koniecznie zwróćmy uwagę na kilka ważnych detali – nie pomijaj więc przepracowania całej niniejszej lekcji.

Nieudane połączenie?

Ponieważ nawiązanie połączenia nie zawsze musi się udać – na przykład:

- pod adresem IP serwera nie ma usługi bazodanowej,
- wpisano niepoprawne dane dostępowe (login lub hasło konta MySQL),
- żądana baza danych o wpisanej nazwie nie istnieje na serwerze,

to na końcu funkcji – jak zapewne zauważyliśmy – pojawił się dodatkowy zapis:

```
or die("Błąd połączenia!");
```

Oznacza to w praktyce: Spróbuj się połączyć, lecz jeśli się nie uda, to “zabij” skrypt i wypisz na stronie komunikat umieszczony w nawiasie. A zatem w razie wykrycia błędu połączenia dalsze wykonanie skryptu PHP (kolejnych linii) zostanie całkowicie wstrzymane. Jest to oczywiście dość brutalne podejście – mamy tak po prostu zabijać cały dalszy skrypt w razie problemów z połączeniem z bazą danych? A co jeśli poniżej połączenia znajdowały się w skrypcie ważne linie – np. znaczniki HTML określające jak w ogóle ta podstrona wygląda? Cóż, zabijanie bez pardonu całego skryptu to metoda dość drakońska, lecz na egzaminie jak najbardziej spełnimy dzięki niej wymóg obsłużenia przez nas – programistów webowych – ewentualnych błędów połączenia PHP z MySQL.

Gdybyśmy jednak zamiast tej leciwej metody “zabijania” całego skryptu chcieli reagować na błędy połączenia bardziej elastycznie, to oczywiście możemy to zrobić – wystarczy do tego celu zastosować tzw. wyłapywanie wyjątków

`try ... catch`. Pokażemy teraz jak to zrobić:

```
<?php
    try {
        $connection = mysqli_connect("localhost", "root", "", "baza123");

        if ($connection->connect_errno!=0) {
            throw new Exception(mysqli_connect_errno());
        }

    } catch (Exception $error) {
        echo 'Błąd bazy danych: '.$error->getMessage();
    }
?>
```

Jak widzimy – ten zapis jest wyraźnie dłuższy, więc jeśli interesuje Cię tylko zdanie egzaminu, to możesz pozostać przy stosowaniu metody "zabijania" skryptu w razie nieudanego połączenia. Jeśli jednak masz ochotę użyć metody współczesnej (tzw. dobra praktyka, *ang. good practices*) to warto teraz wyjaśnić w pełni te zapisy – oto wyjaśnienie:

Najpierw pojawiła się sekcja

`try`, czyli po ang. "spróbuj" – standardowo podejmujemy próbę nawiązania połączenia z bazą funkcją `mysqli_connect()` – zapisujemy to wszystko w klamrach, tak jakby to było ciało instrukcji warunkowej if albo własnej funkcji.

Następnie, w razie wystąpienia problemów (gdy kod błędu przechowywany w atrybucie

`connect_errno` obiektu `$c` jest inny niż zero) dokonujemy tzw. rzucenia wyjątku: `throw new Exception`. Taki wyjątek, mający unikalny identyfikator, można już później "wyłapać" w specjalnej sekcji `catch`, czyli z angielskiego dosłownie "złap". W naszym kodzie następuje wtedy proste wypisanie komunikatu o błędzie, a dodatkowo dzięki metodzie `getMessage()` wywołanej na rzecz obiektu `$error` otrzymamy nieco informacji o naturze błędu.

W profesjonalnych aplikacjach webowych przetworzeniem obiektu o nazwie

`$error` można się zająć za pomocą specjalnie napisanych własnych funkcji lub metod. Wykrycie błędu, połączone z unikatową reakcją na jego rodzaj (osiąganą za sprawą identyfikatora numerycznego), daje możliwość zaprogramowania inteligentnej reakcji skryptu na problemy z komunikacją z serwerem. Nie musimy już zwyczajnie "zabijać" dalszego przetwarzania dokumentu – możemy użytkownika ostrzec, przekierować gdzieś albo wyświetlić zawartość zastępczą. Współczesne aplikacje internetowe zapewniają internaucie właśnie taką "miękką" reakcję na kłopoty z bazą danych, zamiast brutalnie umożliwiać przeglądanie witryny jedynie do miejsca wystąpienia błędu połączenia.

Identyfikator połączenia

Połączenie musi posiadać swoją unikalną nazwę! Przypomnijmy zapis:

```
$connection = mysqli_connect("localhost", "root", "", "baza123") or die("Błąd połączenia!");
```

Nasze połączenia nosi tutaj nazwę

`$connection`, choć oczywiście można to połączenie nazwać dowolnie – inne popularne nazwy to na przykład: `$c`, `$con`, `$p`, `$pol`. Aczkolwiek – uwaga! Egzaminatorzy CKE zwracają uwagę na zastosowanie

znaczących nazw zmiennych i funkcji w skryptach – stąd na egzaminie lepiej użyć `$connection` zamiast `$c`.

Taki identyfikator połączenia (a dokładniej mówiąc obiekt klasy

`$mysqli` reprezentujący połączenie) jest koniecznie potrzebny, ponieważ jeden skrypt PHP może korzystać z wielu połączeń do różnych baz danych. I stąd każde połączenie musi być jednoznacznie nazwane.

Gdybyśmy na przykład tworzyli kod księgarni internetowej, w której z jakiegoś powodu dostępne książki zapisano w innej bazie danych (i na kompletnie innym serwerze) aniżeli dane klientów, to mogłoby to wyglądać tak:

```
// Połączenie z bazą klientów
$connection_clients = mysqli_connect("45.66.92.36", "usr1255", "Tt$w]dDin)#",
"klienci") or die("Błąd bazy klientów!");
// Połączenie z bazą książek
$connection_books = mysqli_connect("45.60.74.29", "adm4765", "&IZ8a]#bw90", "ksiazki")
or die("Błąd bazy książek!");
```

Dzięki różnym identyfikatorom:

`$connection_clients` oraz `$connection_books` od razu wiadomo, którego połączenia w danej chwili używamy. A zwróćmy uwagę, iż potem – przy wykonywaniu zapytań funkcją `$mysqli_query()` (o czym więcej opowiemy w następnej lekcji) – zawsze definiujemy także z użyciem którego połączenia dane zapytanie wykonać:

```
$results = mysqli_query($connection_clients, "SELECT nazwisko FROM uzytkownicy WHERE
id = 76");
```

Powyższe zapytanie zostanie wykonane na tylko na bazie klientów księgarni, ponieważ użyto identyfikatora

`$connection_clients` jako pierwszy parametr funkcji. PHP bez problemu wie którego połączenia użyć, mimo iż aktualnie nawiązane mamy jednocześnie dwa połączenia.

Dane dostępowe do bazy w osobnym, zewnętrznym pliku

Nawiązanie połączenia z bazą danych warto zawsze umieścić w osobnym, zewnętrznym pliku Dlaczego? Ponieważ w przypadku tworzenia całych witryn, a więc wielu skryptów realizujących w aplikacji przeróżne zadania, wielokrotne powtarzanie tych danych dostępowych w wielu różnych plikach wydaje się nieoptymalne. W przypadku konieczności zmiany konfiguracji serwisu (na przykład wskutek przeniesienia na inny serwer) jesteśmy wówczas zmuszeni poprawiać dane dostępowe w każdym skrypcie osobno.

Jeżeli podczas przenoszenia witryny na inny serwer (tzw. migracja) dojdzie do przeoczenia konieczności wprowadzenia zmian w którymkolwiek z wielu plików, powstanie serwis internetowy, w którym poprawnie działa tylko wybrana część skryptów (a my w ogóle nie będziemy o tym wiedzieć). Konieczność testowania każdej funkcjonalności osobno okaże się zadaniem wyjątkowo czasochłonnym.

Współcześnie postępujemy więc inaczej: opis aktu nawiązania połączenia oraz wszystkie dane dostępowe umieszczamy w specjalnie wydzielonym, zewnętrznym pliku. W razie konieczności dokonania zmian w danych dostępowych poprawki aplikujemy tylko w jednym skrypcie, zamiast w wielu rozproszonych zasobach. Specjalnie utworzony, dodatkowy plik możemy nazwać np.

`connect.php`. Oto jego przykładowa zawartość:

```
<?php
    // Konfiguracja dostępu do bazy danych
    $host = "localhost";
    $user = "root";
    $pass = "";
    $db = "nazwa_bazy";
```

```
?>
```

Następnie, w każdym skrypcie PHP, w którym łączymy się z bazą danych, użyjemy instrukcji jednokrotnego żądania specjalnego pliku:

```
require_once "connect.php";
```

W ten sposób dane konfiguracyjne zostaną „podpięte” we wszystkich potrzebnych dokumentach. Jest to idea znana z technologii CSS – style elementów witryny również przenosimy do zewnętrznego arkusza, aby w razie chęci dokonania zmian uniknąć wprowadzania dużej liczby poprawek w wielu miejscach jednocześnie.

Ciekawostka: wiele współczesnych frameworków webowych umieszcza kluczowe informacje dostępowe do usługi MySQL w jednym, zewnętrznym pliku. W przypadku popularnego systemu WordPress jest to zasób o nazwie

`wp-config.php` .

Poprawna obsługa polskich znaków w połączeniu z bazą danych

Już po nawiązaniu połączenia, włączamy obsługę standardu UTF-8 dla przesyłanych danych, używając poniższej funkcji:

```
mysqli_set_charset($connection, "utf8");
```

Argumentami tej funkcji są kolejno:

1. **Identyfikator połączenia**, które musi zostać powyżej tej linii poprawnie nawiązane i pozostawać aktualnie aktywne (niezamknięte),
2. **Nazwa zestawu znaków** – współcześnie używamy standardu UTF-8.

Uwaga! Nazwę standardu wewnątrz funkcji zapisujemy bez myślnika:

`utf8` , inaczej niż to ma miejsce w języku HTML, gdzie zazwyczaj w tagu `<meta>` stosujemy zapis: `utf-8` .

Dodatkowo, aby zapewnić w całej witrynie poprawne kodowanie polskich znaków (w tym również tekstów wyjmowanych z bazy bądź wkładanych do niej) należy:

- użyć w kodzie źródłowym witryny, wewnątrz znacznika `<html>` atrybutu: `<html lang="pl">` ,
- użyć w kodzie źródłowym witryny, w sekcji `<head>` , ponad tytułem `<title>` znacznika `<meta charset="utf-8">` ,
- ustawić kodowanie pliku z rozszerzeniem `<html>` .php na standard `<html>` UTF-8. Na przykład w edytorze Notepad++ zmiany sposobu kodowania dokumentu dokonujemy wybierając z menu głównego: `Format (Encoding)` -> `Koduj w UTF-8` ,
- upewnić się, iż wszystkie pola typu tekstowego (znajdujące się w strukturze tabeli) bazy danych posiadają ustawioną metodę porównywania napisów `utf8_bin` albo `utf8_polish_ci` .

Zamknięcie aktywnego połączenia z usługą MySQL

Po zakończeniu wszystkich działań dokonanych w skrypcie na bazie danych, wskazane jest zakończenie połączenia poniższą funkcją – jej jedynym parametrem jest identyfikator zamykanego połączenia:

```
mysqli_close($connection);
```

Co jednak ciekawe – wcale nie jest to działanie obiektywnie konieczne, gdyż po zakończeniu wykonywania skryptu połączenie zostanie i tak automatycznie przerwane, jednak na egzaminie często wymaga się od nas wstawienia do kodu tej funkcji.

Podobnie zresztą akceptowane w PHP jest pominięcie zamykającego tagu

`?>` po uprzednim otwarciu go znacznikiem `<?php` o ile tylko kod PHP kończy skrypt (nie ma już np. HTML poniżej). Niemniej jednak również unikałbym takiej praktyki na samym egzaminie – mamy udowodnić, iż jako programiści potrafimy domykać otwarte tagi.

Wersje obiektowe funkcji obsługujących połączenie

We współczesnym programowaniu wyróżniamy dwa podstawowe tzw. paradygmaty, czyli po prostu przyjęte konwencje budowania aplikacji:

1. **Podejście proceduralne** – stosujemy własne funkcje, czyli wydzielone fragmenty kodu, które realizują konkretne zadania.
2. **Podejście obiektowe** – bardziej abstrakcyjna i złożona, gdyż używamy obiektów skonstruowanych na bazie szablonów zwanych klasami. Każdy obiekt ma przypisane pewne atrybuty (zmienne) oraz metody (funkcje zdefiniowane wewnątrz klas).

Biblioteka

`mysqli` w PHP pozwala nam na używanie zarówno funkcji jak i obiektów posiadających atrybuty i metody. Możemy także "mieszać" oba podejścia, tzn. niektóre linie zapisać proceduralnie, a inne obiektowo. Oczywiście do egzaminu spokojnie wystarczy nam znajomość jedynie funkcji, lecz gwoli udzielenia Wam pełnej informacji przedstawmy teraz także wersje obiektowe zapisów poznanych w tej lekcji.

Obiektowe nawiązanie połączenia z bazą danych

W wersji obiektowej pojawia się dodatkowa klauzula

`new`, która tworzy nowy obiekt o nazwie `$connection`. Metoda `mysqli()` to tzw. **konstruktor** obiektu. Kolejność argumentów wewnątrz nawiasów okrągłych pozostaje w obu wariantach identyczna.

Wersja proceduralna:

```php
$connection = mysqli_connect("localhost", "root", "", "baza123");
```

Wersja obiektowa:

```php
$connection = new mysqli("localhost", "root", "", "baza123");
```

Obiektowa obsługa polskich znaków

Podobnie jak w wersji proceduralnej, odpowiednią linię kodu umieszczamy już po nawiązaniu aktywnego połączenia. Tym razem jednak zapis ma następującą postać:

```
obiekt->metoda(argumenty);
```

Wersja proceduralna:

```php
mysqli_set_charset($connection, "utf8");
```

Wersja obiektowa:

```php
$connection->set_charset("utf8");
```

Obiektowe zamknięcie połączenia z bazą danych

Na rzecz obiektu

`$connection` reprezentującego nawiązane połączenie wywołujemy metodę o nazwie `close()` . W tym przypadku argument staje się zbędny, gdyż po lewej stronie operatora `->` znajduje się już informacja, które z istniejących połączeń zamierzamy zakończyć:

Wersja proceduralna:

```
mysqli_close($connection);
```

Wersja obiektowa:

```
$connection->close();
```

Podsumowanie

Co warto zapamiętać z tej lekcji? Przedstawmy najważniejsze wnioski:

- Do połączenia z MySQL potrzebujemy zazwyczaj czterech danych: host, login, hasło, nazwa bazy.
- Na lokalnym XAMPP domyślne dane to: `"localhost", "root", "", "nazwa_bazy"` .
- Podstawowa funkcja do łączenia się z bazą danych to: `mysqli_connect()` .
- Warto na egzaminie używać zapisu: `or die("Błąd połączenia!")` by (przynajmniej w taki podstawowy sposób) obsłużyć błędy.
- Bardziej zaawansowana metoda obsługi błędów to `try...catch` z wyjątkiem `throw new Exception()` .
- Identyfikator połączenia powinien mieć na egzaminie czytelną nazwę, np. `$connection` .
- Można mieć kilka połączeń jednocześnie (np. do różnych baz), każdy musi mieć własną nazwę.
- Dane dostępowe warto trzymać w osobnym pliku (np. `connect.php`) i dołączać przez `require_once` .
- Po połączeniu należy ustawić kodowanie znaków dzięki `mysqli_set_charset()` .
- Dla poprawnej obsługi polskich znaków ważne są też: `<meta charset="utf-8">` i odpowiednie kodowanie plików.
- Zamykamy połączenie funkcją `mysqli_close($connection)` – mimo, że PHP zrobi to samodzielnie na końcu skryptu.
- Rozszerzenie mysqli pozwala używać zarówno funkcji (podejście proceduralne), jak i metod obiektowych (np. `$connection->close()`).
- Wersja obiektowa połączenia wygląda tak: `$connection = new mysqli(...)` .