

Lekcja: Łączenie Strony WWW z Bazą Danych MySQL przy użyciu PHP i MySQLi w Trybie Proceduralnym

Cel lekcji:

Nauczenie, jak łączyć stronę internetową z bazą danych MySQL za pomocą PHP, korzystając z biblioteki MySQLi w trybie proceduralnym. Lekcja obejmuje łączenie z bazą danych, wykonywanie zapytań, wstawianie, aktualizowanie, usuwanie oraz pobieranie danych.

Plan lekcji:

1. Wprowadzenie do MySQLi w trybie proceduralnym
2. Tworzenie bazy danych i tabeli MySQL
3. Łączenie z bazą danych przy użyciu PHP (MySQLi)
4. Podstawowe operacje CRUD (Create, Read, Update, Delete)
5. Podsumowanie

1. Wprowadzenie do MySQLi w Trybie Proceduralnym

MySQLi (MySQL Improved) jest rozszerzeniem PHP, które służy do komunikacji z bazą danych MySQL. W trybie proceduralnym operacje wykonywane są za pomocą funkcji PHP. Istnieją również tryby obiektowe oraz PDO, ale dzisiaj skupimy się na trybie proceduralnym.

Dlaczego używać MySQLi?

- Lepsze wsparcie i bezpieczeństwo w porównaniu do starszego rozszerzenia `mysql` (już nie używane).
- Obsługuje operacje asynchroniczne.
- Jest łatwe do zrozumienia i wdrożenia.

Wyjaśnienie pojęcia "Operacje Asynchroniczne"

1. Asynchroniczność vs. Synchroniczność

- **Operacje synchroniczne:** W trybie synchronicznym (domyślny sposób działania wielu funkcji w PHP), kiedy wykonujemy zapytanie do bazy danych, skrypt PHP czeka na zakończenie tej operacji, zanim przejdzie do wykonania kolejnych linii kodu. Oznacza to, że skrypt blokuje się, aż serwer bazy danych odpowie, co może wpłynąć na wydajność w przypadku długotrwałych zapytań lub dużej liczby zapytań.
- **Operacje asynchroniczne:** W trybie asynchronicznym zapytania do bazy danych mogą być wysyłane bez oczekiwania na ich zakończenie. Oznacza to, że skrypt PHP może kontynuować wykonywanie innych operacji podczas oczekiwania na odpowiedź z bazy danych. Asynchroniczność może zwiększyć wydajność aplikacji, ponieważ umożliwia jednoczesne przetwarzanie innych zadań, a nie blokuje działania aplikacji.

2. Tworzenie Bazy Danych i Tabeli MySQL

```
CREATE DATABASE lekcja_baza;
USE lekcja_baza;

CREATE TABLE uzytkownicy (
  id INT AUTO_INCREMENT PRIMARY KEY,
  imie VARCHAR(50),
  nazwisko VARCHAR(50),
  email VARCHAR(100),
  wiek INT
);
```

```
CREATE DATABASE lekcja_baza;
USE lekcja_baza;

CREATE TABLE uzytkownicy (
  id INT AUTO_INCREMENT PRIMARY KEY,
  imie VARCHAR(50),
  nazwisko VARCHAR(50),
  email VARCHAR(100),
  wiek INT
);
```

3. Łączenie z Bazą Danych przy użyciu PHP (MySQLi)

Konfiguracja połączenia

Musimy połączyć naszą stronę internetową z bazą danych MySQL. W tym celu użyjemy funkcji `mysqli_connect()`.

```
<?php
// Dane połączenia z bazą danych
$host = 'localhost';
$username = 'root';
$password = '';
$dbname = 'lekcja_baza';

$conn = mysqli_connect($host, $username,
$password, $dbname);

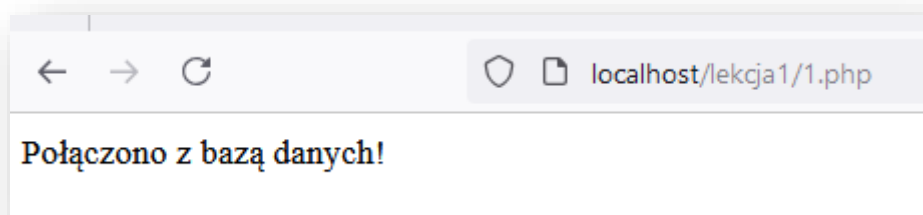
// Sprawdzanie połączenia
if (!$conn) {
    die("Błąd połączenia: " .
mysqli_connect_error());
}
echo "Połączono z bazą danych!";
?>
```

```
<?php
// Dane połączenia z bazą danych
$host = 'localhost';
$username = 'root'; // domyślnie 'root' dla XAMPP/MAMP
$password = ''; // domyślnie puste dla XAMPP/MAMP
$dbname = 'lekcja_baza';

// Nawiązanie połączenia
$conn = mysqli_connect($host, $username, $password, $dbname);

// Sprawdzanie połączenia
if (!$conn) {
    die("Błąd połączenia: " . mysqli_connect_error());
}
echo "Połączono z bazą danych!";
?>
```

`mysqli_connect_error()` jest funkcją w PHP, która zwraca komunikat o błędzie związany z próbą połączenia z bazą danych MySQL za pomocą rozszerzenia MySQLi (MySQL Improved). Jeśli połączenie zakończyło się niepowodzeniem, `mysqli_connect_error()` zwraca opis błędu, co jest bardzo pomocne przy debugowaniu problemów z połączeniem.



Uwaga! Jeśli nie trzeba sprawdzenia czy udało nawiązać się połączenie z serwerem będziemy używać na lekcji np.

```
$c=mysqli_connect('localhost','root','','kwiaciarnia');
```

4. Podstawowe Operacje CRUD (Create, Read, Update, Delete)

CRUD to akronim od Create (tworzenie), Read (odczyt), Update (aktualizacja) i Delete (usuwanie). Pokaże przykłady każdej z tych operacji.

4.1 Tworzenie (Create) - Wstawianie Nowych Danych

Stworzymy formularz HTML do dodawania nowych użytkowników, a następnie użyjemy PHP do wstawienia danych do bazy.

Formularz HTML do wstawiania danych:

```
<form action="dodaj_uzytkownika.php"
method="post">
  Imię: <input type="text" name="imie"><br>
  Nazwisko: <input type="text"
name="nazwisko"><br>
  Email: <input type="email"
name="email"><br>
  Wiek: <input type="number"
name="wiek"><br>
  <input type="submit" value="Dodaj
uzytkownika">
</form>
```

```
<form action="dodaj_uzytkownika.php" method="post">
  Imię: <input type="text" name="imie"><br>
  Nazwisko: <input type="text" name="nazwisko"><br>
  Email: <input type="email" name="email"><br>
  Wiek: <input type="number" name="wiek"><br>
  <input type="submit" value="Dodaj uzytkownika">
</form>
```

Imię:

Nazwisko:

Email:

Wiek:

Kod PHP do obsługi formularza i wstawiania danych:

```
<?php
if ($_SERVER["REQUEST_METHOD"] ==
"POST") {
  $imie = $_POST['imie'];
  $nazwisko = $_POST['nazwisko'];
  $email = $_POST['email'];
  $wiek = $_POST['wiek'];

  $sql = "INSERT INTO uzytkownicy (imie,
nazwisko, email, wiek)
VALUES ('$imie', '$nazwisko',
'$email', '$wiek)";

  if (mysqli_query($conn, $sql)) {
    echo "Nowy użytkownik został
dodany.";
  } else {
    echo "Błąd: " . $sql . "<br>" .
mysqli_error($conn);
  }
}
?>
```

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $imie = $_POST['imie'];
  $nazwisko = $_POST['nazwisko'];
  $email = $_POST['email'];
  $wiek = $_POST['wiek'];

  $sql = "INSERT INTO uzytkownicy (imie, nazwisko, email, wiek)
VALUES ('$imie', '$nazwisko', '$email', $wiek)";

  if (mysqli_query($conn, $sql)) {
    echo "Nowy użytkownik został dodany.";
  } else {
    echo "Błąd: " . $sql . "<br>" . mysqli_error($conn);
  }
}
?>
```

if (\$_SERVER["REQUEST_METHOD"] == "POST") to standardowy sposób sprawdzania, czy dane zostały przesłane metodą POST i umożliwia odpowiednie przetwarzanie tych danych.

← → ↻ localhost/lekcja1/dodaj_uzytkownika.php

Imię:

Nazwisko:

Email:

Wiek:

Połączono z bazą danych! Nowy użytkownik został dodany.

Wyjaśnienie:

1. Czym jest `$_SERVER["REQUEST_METHOD"]`?

`$_SERVER` to superglobalna tablica w PHP, która przechowuje informacje o nagłówkach, ścieżkach i lokalizacjach skryptów. Klucz `"REQUEST_METHOD"` w tej tablicy zwraca metodę HTTP używaną do wysłania żądania do serwera. Możliwe wartości to:

- `GET` – najczęściej używana do pobierania zasobów lub wyświetlania danych. Parametry są przesyłane w adresie URL.
- `POST` – używana do przysyłania danych, takich jak formularze, gdzie dane nie są widoczne w adresie URL.
- `PUT`, `DELETE`, itd. – rzadziej stosowane metody używane głównie w REST API.

2. Jak działa `if ($_SERVER["REQUEST_METHOD"] == "POST")`?

Ten fragment kodu sprawdza, czy metoda żądania użyta do wysłania formularza lub wykonania żądania to `POST`. Jeśli tak, kod wewnątrz tego bloku `if` zostanie wykonany.

3. Przykład zastosowania w kodzie:

Użycie tego warunku jest typowe w przypadku przetwarzania formularzy HTML. Oto przykład:

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Kod wewnątrz tego bloku zostanie wykonany tylko wtedy, gdy żądanie jest metodą POST
    $imie = $_POST['imie'];
    $email = $_POST['email'];

    // Walidacja danych, zapisywanie do bazy danych, wysyłanie emaila, itp.
    echo "Formularz został przesłany metodą POST. Imię: $imie, Email: $email";
}
?>
```

`!empty()` - Funkcja `empty()` sprawdza, czy zmienna jest pusta. Można jej użyć do upewnienia się, że wszystkie wymagane pola formularza zostały wypełnione.

```
// Sprawdzenie, czy wszystkie pola zostały wypełnione
if (!empty($imie) && !empty($nazwisko) && !empty($email) && !empty($wiek)) {
    // Tutaj możesz dodać kod do dodania użytkownika do bazy danych lub innej
    logiki
    echo "Użytkownik został dodany!";
} else {
    echo "Proszę wypełnić wszystkie pola formularza.";
```

Zmodyfikowany poprzedni kod o sprawdzenie czy wszystkie dane zostały wpisane do formularza:

```
<?php
if ($_SERVER['REQUEST_METHOD'] === 'POST') { // Sprawdza, czy formularz został
wysłany metodą POST
    // Pobieranie danych z formularza
    $imie = $_POST['imie'];
    $nazwisko = $_POST['nazwisko'];
    $email = $_POST['email'];
    $wiek = $_POST['wiek'];

    // Sprawdzenie, czy wszystkie pola zostały wypełnione
```

```

if (!empty($imie) && !empty($nazwisko) && !empty($email) && !empty($wiek)) {

    $sql = "INSERT INTO uzytkownicy (imie, nazwisko, email, wiek)
VALUES ('$imie', '$nazwisko', '$email', $wiek)";

    if (mysqli_query($conn, $sql)) {
        echo "Nowy użytkownik został dodany.";
    } else {
        echo "Błąd: " . $sql . "<br>" . mysqli_error($conn);
    }

    echo "Użytkownik został dodany!";
} else {
    echo "Proszę wypełnić wszystkie pola formularza.";
}
}
?>

```

Znaczenie wyrażenia `$_SERVER['REQUEST_METHOD'] === 'POST'`

- `===` jest operatorem porównania w PHP, który sprawdza zarówno wartość, jak i typ zmiennej. W tym przypadku sprawdza, czy wartość `$_SERVER['REQUEST_METHOD']` jest równa ciągowi znaków `POST`.
- Jeśli metoda żądania jest `POST`, to wyrażenie zwraca `true`, co oznacza, że formularz został przesłany metodą `POST`. Jeśli jest inna (np. `GET`), to wyrażenie zwróci `false`.

Metody Fetchowania Rekordów do Tablicy Jednowymiarowej w MySQLi

Przy użyciu **MySQLi** w trybie proceduralnym, wyróżniamy następujące metody:

1. `mysqli_fetch_row()`:
 - Zwraca rekord jako **numeryczną tablicę jednowymiarową**, gdzie każdy element tablicy odpowiada kolejnemu polu w wyniku.

```

<?php
$result = mysqli_query($conn, "SELECT imie, nazwisko FROM uzytkownicy");
while ($row = mysqli_fetch_row($result)) {
    echo $row[0] . " " . $row[1]; // Imię i nazwisko
}
?>

```

<?php

\$result = mysqli_query(\$conn, "SELECT imie, nazwisko FROM uzytkownicy");

while (\$row = mysqli_fetch_row(\$result)) {

echo \$row[0] . " " . \$row[1]; // Imię i nazwisko

}

?>

2. `mysqli_fetch_assoc()`:

- Zwraca rekord jako **asocjacyjną tablicę jednowymiarową**, gdzie klucze to nazwy kolumn, a wartości to dane z odpowiadających im kolumn.
- **Przykład:**

```
<?php
$result = mysqli_query($conn, "SELECT imie, nazwisko FROM uzytkownicy");
while ($row = mysqli_fetch_assoc($result)) {
    echo $row['imie'] . " " . $row['nazwisko']; // Imię i nazwisko
}
-?>
```

```
<?php
```

```
$result = mysqli_query($conn, "SELECT imie, nazwisko FROM uzytkownicy");
```

```
while ($row = mysqli_fetch_assoc($result)) {
```

```
    echo $row['imie'] . " " . $row['nazwisko']; // Imię i nazwisko
```

```
}
```

```
?>
```

3. `mysqli_fetch_array()`:

- Zwraca rekord jako **tablicę jednowymiarową**, która może być zarówno numeryczna, jak i asocjacyjna (domyślnie łączy oba formaty). Można jednak ograniczyć wynik do tylko jednego typu, przekazując drugi parametr: `MYSQLI_ASSOC` lub `MYSQLI_NUM`.
- **Przykład:**

```
<?php
$result = mysqli_query($conn, "SELECT imie, nazwisko FROM uzytkownicy");
while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) { // tylko asocjacyjnie
    echo $row['imie'] . " " . $row['nazwisko']; // Imię i nazwisko
}
-?>
```

```
<?php
```

```
$result = mysqli_query($conn, "SELECT imie, nazwisko FROM uzytkownicy");
```

```
while ($row = mysqli_fetch_array($result, MYSQLI_ASSOC)) { // tylko asocjacyjnie
```

```
    echo $row['imie'] . " " . $row['nazwisko']; // Imię i nazwisko
```

```
}
```

```
?>
```

NUMERYCZNA I ASOCJACYJNA TABLICA

```
<?php
$result = mysqli_query($conn, "SELECT imie, nazwisko FROM uzytkownicy");
while ($row = mysqli_fetch_array($result)) {
    echo $row['imie'] . " " . $row['1'];
}
-?>
```

```
$result = mysqli_query($conn, "SELECT imie, nazwisko FROM uzytkownicy");
while ($row = mysqli_fetch_array($result)) {
```

```
    echo "<br>". $row['imie'] . " " . $row[1]. "<br>";
}
```

4.2 Odczyt (Read) - Wyświetlanie Danych

Wcześniej wstaw 10 przykładowych rekordów:

```
INSERT INTO uzytkownicy (imie, nazwisko, email, wiek) VALUES

('Anna', 'Kowalska', 'anna.kowalska@example.com', 25),

('Piotr', 'Nowak', 'piotr.nowak@example.com', 34),

('Katarzyna', 'Wiśniewska', 'katarzyna.wisniewska@example.com', 29),

('Tomasz', 'Wójcik', 'tomasz.wojcik@example.com', 40),

('Magdalena', 'Kowalczyk', 'magdalena.kowalczyk@example.com', 27),

('Paweł', 'Kamiński', 'pawel.kaminski@example.com', 31),

('Agnieszka', 'Lewandowska', 'agnieszka.lewandowska@example.com', 36),

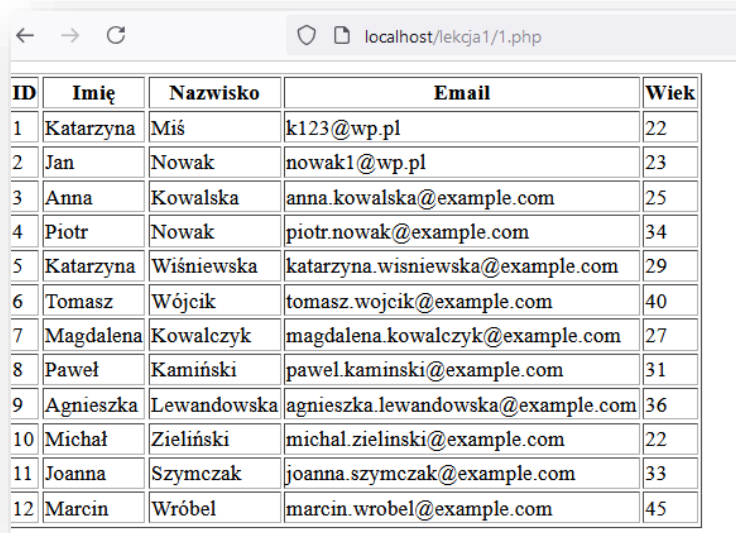
('Michał', 'Zieliński', 'michal.zielinski@example.com', 22),

('Joanna', 'Szymczak', 'joanna.szymczak@example.com', 33),

('Marcin', 'Wróbel', 'marcin.wrobel@example.com', 45);
```

Użyjemy `mysqli_query()` do pobrania danych z tabeli i `mysqli_fetch_assoc()` do ich wyświetlenia.

<pre><?php \$sql = "SELECT * FROM uzytkownicy"; \$result = mysqli_query(\$conn, \$sql); if (mysqli_num_rows(\$result) > 0) { echo "<table border='1'> <tr> <th>ID</th><th>Imię</th><th>Nazwisko</th> <th>Email</th><th>Wiek</th> </tr>"; while (\$row = mysqli_fetch_assoc(\$result)) { echo "<tr> <td>" . \$row['id'] . "</td> <td>" . \$row['imie'] . "</td> <td>" . \$row['nazwisko'] . "</td> <td>" . \$row['email'] . "</td> <td>" . \$row['wiek'] . "</td> </tr>"; } echo "</table>"; } else { echo "Brak danych."; } ?></pre>	<pre><?php \$sql = "SELECT * FROM uzytkownicy"; \$result = mysqli_query(\$conn, \$sql); if (mysqli_num_rows(\$result) > 0) { echo "<table border='1'> <tr> <th>ID</th><th>Imię</th><th>Nazwisko</th> <th>Email</th><th>Wiek</th> </tr>"; while (\$row = mysqli_fetch_assoc(\$result)) { echo "<tr> <td>" . \$row['id'] . "</td> <td>" . \$row['imie'] . "</td> <td>" . \$row['nazwisko'] . "</td> <td>" . \$row['email'] . "</td> <td>" . \$row['wiek'] . "</td> </tr>"; } echo "</table>"; } else { echo "Brak danych."; } ?></pre>
--	--



ID	Imię	Nazwisko	Email	Wiek
1	Katarzyna	Miś	k123@wp.pl	22
2	Jan	Nowak	nowak1@wp.pl	23
3	Anna	Kowalska	anna.kowalska@example.com	25
4	Piotr	Nowak	piotr.nowak@example.com	34
5	Katarzyna	Wiśniewska	katarzyna.wisniewska@example.com	29
6	Tomasz	Wójcik	tomasz.wojcik@example.com	40
7	Magdalena	Kowalczyk	magdalena.kowalczyk@example.com	27
8	Paweł	Kamiński	pawel.kaminski@example.com	31
9	Agnieszka	Lewandowska	agnieszka.lewandowska@example.com	36
10	Michał	Zieliński	michal.zielinski@example.com	22
11	Joanna	Szymczak	joanna.szymczak@example.com	33
12	Marcin	Wróbel	marcin.wrobel@example.com	45

Rysunek 1. Wynik działania powyższego skryptu.

```
$sql = "SELECT * FROM uzytkownicy";
$result = mysqli_query($conn, $sql);
// Sprawdzanie liczby wierszy w wyniku zapytania
if ($result) {
    $num_rows = mysqli_num_rows($result);
    echo "Liczba rekordów w tabeli 'uzytkownicy': " . $num_rows;
} else {
    echo "Błąd zapytania: " . mysqli_error($conn);
}
```

Liczba rekordów w tabeli 'uzytkownicy': 12

4.3 Aktualizacja (Update) - Edycja Danych

Formularz HTML do aktualizacji danych:

<pre><form action="aktualizuj_uzytkownika.php" method="post"> ID Użytkownika: <input type="number" name="id">
 Nowy Email: <input type="email" name="nowy_email">
 <input type="submit" value="Aktualizuj"> </form></pre>	<pre><form action="aktualizuj_uzytkownika.php" method="post"> ID Użytkownika: <input type="number" name="id">
 Nowy Email: <input type="email" name="nowy_email">
 <input type="submit" value="Aktualizuj"> </form></pre>
---	---

Kod PHP do aktualizacji danych:

<pre><?php if (\$_SERVER["REQUEST_METHOD"] == "POST") { \$id = \$_POST['id']; \$nowy_email = \$_POST['nowy_email']; \$sql = "UPDATE uzytkownicy SET email='\$nowy_email' WHERE id=\$id"; if (mysqli_query(\$conn, \$sql)) { echo "Email użytkownika został zaktualizowany."; } else { echo "Błąd: " . \$sql . "
" . mysqli_error(\$conn); } } ?></pre>	<pre><?php if (\$_SERVER["REQUEST_METHOD"] == "POST") { \$id = \$_POST['id']; \$nowy_email = \$_POST['nowy_email']; \$sql = "UPDATE uzytkownicy SET email='\$nowy_email' WHERE id=\$id"; if (mysqli_query(\$conn, \$sql)) { echo "Email użytkownika został zaktualizowany."; } else { echo "Błąd: " . \$sql . "
" . mysqli_error(\$conn); } } ?></pre>
---	---

ID Użytkownika:

Nowy Email:

Aktualizuj

4.4 Usuwanie (Delete) - Usuwanie Danych

Formularz HTML do usuwania danych:

<pre><form action="usun_uzytkownika.php" method="post"> ID Użytkownika: <input type="number" name="id">
 <input type="submit" value="Usuń"> </form></pre>	<pre><form action="usun_uzytkownika.php" method="post"> ID Użytkownika: <input type="number" name="id">
 <input type="submit" value="Usuń"> </form></pre>
--	--

Kod PHP do usuwania danych:

<pre><?php if (\$_SERVER["REQUEST_METHOD"] == "POST") { \$id = \$_POST['id']; \$sql = "DELETE FROM uzytkownicy WHERE id=\$id"; if (mysqli_query(\$conn, \$sql)) { echo "Użytkownik został usunięty."; } else { echo "Błąd: " . \$sql . "
" . mysqli_error(\$conn); } } ?></pre>	<pre><?php if (\$_SERVER["REQUEST_METHOD"] == "POST") { \$id = \$_POST['id']; \$sql = "DELETE FROM uzytkownicy WHERE id=\$id"; if (mysqli_query(\$conn, \$sql)) { echo "Użytkownik został usunięty."; } else { echo "Błąd: " . \$sql . "
" . mysqli_error(\$conn); } } ?></pre>
--	--

ID Użytkownika:

Usuń

Użytkownik został usunięty.

Użytkownik 1 został usunięty:

ID	Imię	Nazwisko	Email	Wiek
2	Jan	Nowak	nowak1@wp.pl	23
3	Anna	Kowalska	anna.kowalska@example.com	25
4	Piotr	Nowak	piotr.nowak@example.com	34

5. Bezpieczeństwo przy korzystaniu z MySQLi

- **Unikaj SQL Injection:** Używaj funkcji `mysqli_real_escape_string()` lub przygotowanych zapytań (prepared statements).
- **Weryfikacja danych wejściowych:** Zawsze sprawdzaj i filtruj dane użytkownika.
- **Obsługa błędów:** Stosuj odpowiednią obsługę błędów, aby wyświetlać informacje diagnostyczne bez ujawniania wrażliwych danych.

6. Podsumowanie

W tej lekcji nauczyliśmy się:

- Jak połączyć stronę WWW z bazą danych MySQL za pomocą PHP i MySQLi.
- Jak wykonać podstawowe operacje CRUD.
- Jak dbać o bezpieczeństwo danych w aplikacji webowej.

Materiał dodatkowy !!!

SQL Injection to jedno z najczęstszych zagrożeń bezpieczeństwa w aplikacjach webowych, które polega na manipulacji zapytaniami SQL przez złośliwe dane wejściowe. Aby tego uniknąć, możemy użyć dwóch podejść:

1. **`mysqli_real_escape_string()`** — ręczne zabezpieczenie danych wejściowych przez dodanie odpowiednich znaków specjalnych.
2. **Przygotowane zapytania (prepared statements)** — bezpieczniejsze podejście, które oddziela dane od logiki SQL.

Poniżej przedstawiam przykłady dla obu podejść.

1. Przykład z `mysqli_real_escape_string()`

`mysqli_real_escape_string()` jest używane do ręcznego zabezpieczenia danych wejściowych przekazywanych do zapytania SQL. Funkcja ta dodaje znaki ucieczki (ang. escaping characters), które chronią przed niezamierzonym zakończeniem lub modyfikacją zapytania SQL.

Przykład:

php

```
<?php
// Połączenie z bazą danych
$conn = mysqli_connect("localhost", "root", "", "lekcja_baza");

// Sprawdzenie połączenia
if (!$conn) {
```

```

        die("Błąd połączenia: " . mysqli_connect_error());
    }

    // Pobranie danych wejściowych od użytkownika (np. z formularza)
    $imie = $_POST['imie'];
    $email = $_POST['email'];

    // Zabezpieczenie danych za pomocą mysqli_real_escape_string()
    $imie = mysqli_real_escape_string($conn, $imie);
    $email = mysqli_real_escape_string($conn, $email);

    // Bezpieczne wykonanie zapytania SQL
    $sql = "INSERT INTO uzytkownicy (imie, email) VALUES ('$imie', '$email')";

    if (mysqli_query($conn, $sql)) {
        echo "Nowy rekord został dodany pomyślnie.";
    } else {
        echo "Błąd: " . mysqli_error($conn);
    }

    // Zamknięcie połączenia
    mysqli_close($conn);
?>

```

Wyjaśnienie:

- Dane wejściowe od użytkownika (zmienne `$imie` i `$email`) są "oczyszczane" przy użyciu `mysqli_real_escape_string()`, co dodaje znaki ucieczki do znaków specjalnych, takich jak apostrofy (`'`), które mogłyby zakłócić strukturę zapytania SQL.
- Następnie zapytanie SQL jest wykonywane bez ryzyka SQL Injection.

2. Przykład z Przygotowanymi Zapytaniami (Prepared Statements)

Przygotowane zapytania (ang. *prepared statements*) to bardziej bezpieczna i zalecana metoda zapobiegania SQL Injection. W tej metodzie zapytanie SQL jest przygotowywane z placeholderami (`?`), a dane wejściowe są do nich wiązane później. Silnik bazy danych wie, że dane i zapytanie to dwa oddzielne komponenty.

Przykład:

```

<?php
// Połączenie z bazą danych
$conn = mysqli_connect("localhost", "root", "", "lekcja_baza");

// Sprawdzenie połączenia
if (!$conn) {
    die("Błąd połączenia: " . mysqli_connect_error());
}

```

```
// Pobranie danych wejściowych od użytkownika (np. z formularza)
$imie = $_POST['imie'];
$email = $_POST['email'];

// Przygotowanie zapytania SQL
$stmt = mysqli_prepare($conn, "INSERT INTO uzytkownicy (imie, email) VALUES (?, ?)");

// Powiązanie zmiennych z zapytaniem SQL (s - string)
mysqli_stmt_bind_param($stmt, "ss", $imie, $email);

// Wykonanie przygotowanego zapytania
if (mysqli_stmt_execute($stmt)) {
    echo "Nowy rekord został dodany pomyślnie.";
} else {
    echo "Błąd: " . mysqli_stmt_error($stmt);
}

// Zamknięcie zapytania i połączenia
mysqli_stmt_close($stmt);
mysqli_close($conn);
?>
```

Wyjaśnienie:

- Funkcja `mysqli_prepare()` przygotowuje zapytanie SQL z dwoma placeholderami (?), które zostaną zastąpione rzeczywistymi danymi wejściowymi.
- `mysqli_stmt_bind_param()` wiąże zmienne `$imie` i `$email` z placeholderami w zapytaniu. Drugi parametr "ss" oznacza, że obie wartości są ciągami znaków (string).
- `mysqli_stmt_execute()` wykonuje zapytanie z podanymi danymi.
- Przygotowane zapytania są odporne na SQL Injection, ponieważ silnik bazy danych traktuje przekazane dane jako wartości, a nie część zapytania SQL.

Porównanie obu metod:

- **`mysqli_real_escape_string()`:**
 - Wymaga ręcznego "oczyszczania" danych.
 - Działa poprawnie, ale jest mniej bezpieczne niż przygotowane zapytania.
 - Może być podatne na błędy w przypadkach, gdy programista zapomni "oczyścić" dane.
- **Przygotowane zapytania (prepared statements):**
 - Automatycznie oddzielają dane od zapytania.
 - Są bardziej bezpieczne i mniej podatne na błędy.
 - Zalecana metoda przy pracy z danymi wejściowymi od użytkowników.

Wniosek:

W miarę możliwości należy zawsze stosować **przygotowane zapytania (prepared statements)** jako bardziej bezpieczną metodę ochrony przed SQL Injection. `mysqli_real_escape_string()` może być użyteczne w prostych przypadkach, ale ma większy potencjał na błędy programistyczne.