# PHP na egzaminie INF.03

Szczegółowe objaśnienia składni, formularzy i współpracy z bazą MySQL / MariaDB. Przykłady zapisano przede wszystkim w **proceduralnym stylu MySQLi**, dzięki czemu łatwo prześledzić kolejność wykonywanych czynności.

## Od pierwszego echo do kompletnej aplikacji

| Etap | Czego się nauczysz |
|---|---|
| 1. Podstawy | Uruchamiać PHP, rozumieć zmienne, warunki, pętle i tablice. |
| 2. Formularze | Odbierać GET i POST, sprawdzać dane, liczyć wyniki i wyświetlać komunikaty. |
| 3. Baza danych | Nawiązać połączenie, wykonać SELECT, odczytać rekordy oraz używać INSERT, UPDATE i DELETE. |
| 4. Praktyka | Połączyć formularz z bazą, rozpoznawać błędy i samodzielnie testować rozwiązanie. |

## Jak korzystać z podręcznika

Czytaj kod od góry do dołu. Najpierw przewiduj wynik, następnie uruchamiaj przykład i zmieniaj jedną rzecz naraz. Objaśnienia odnoszą się do konkretnych instrukcji, nazw pól i wartości. Przepisując fragment, sprawdź, jakie zmienne i pliki muszą już istnieć.

**Założenia:** PHP 8.x, serwer obsługujący PHP, rozszerzenie mysqli oraz lokalna baza MySQL lub MariaDB. Fragmenty bez znacznika otwierającego PHP należy umieszczać wewnątrz bloku **<?php ... ?>**. Pełne pliki zostały oznaczone nazwami.

**Zakres:** praktyczne przygotowanie do części dotyczącej PHP, a nie kompletny kurs całej kwalifikacji. INF.03 obejmuje również inne obszary, m.in. HTML, CSS, JavaScript i SQL. W konkretnym zadaniu zawsze obowiązują jego dokładne wymagania.

**Zapamiętaj:** W PHP mówimy głównie o instrukcjach, operatorach i funkcjach. Znaczniki takie jak form, input i table należą do HTML. PHP może generować ich kod.

Przykłady i ćwiczenia są autorskie. Materiał zawiera także dobre praktyki bezpieczeństwa i dodatki wykraczające poza pojedyncze zadanie egzaminacyjne. Źródła i dokumentację znajdziesz na końcu. Wydanie: 16.09.2026.

# Spis treści i ścieżka nauki

**03** Podstawy: uruchamianie, składnia, zmienne

**06** Operatory, warunki, pętle i tablice

**10** Tekst, liczby, daty i własne funkcje

**13** HTML formularza; GET i POST

**16** Walidacja, checkbox, select i kalkulator

**19** Bezpieczne wyświetlanie danych

**20** Baza ćwiczeniowa i podstawy SQL

**22** Połączenie z bazą i obsługa błędów

**23** SELECT, pobieranie rekordów i tabela HTML

**26** Zapytania przygotowane i wyszukiwanie

**28** INSERT, UPDATE i DELETE

**30** Projekt: katalog produktów

**34** Sesje, cookies, pliki i hasła - dodatki

**36** Diagnostyka i lista kontrolna

**37** Zadania z odpowiedziami

**39** Ściąga: funkcje PHP i MySQLi

**41** Źródła i dalsza nauka

## Trzy przejścia przez materiał

**Pierwsze:** strony 3-19. Zrozum dane wysyłane z formularza, warunki oraz sposób generowania HTML. **Drugie:** strony 20-33. Prześledź drogę od połączenia z bazą do wyświetlenia wyniku. **Trzecie:** rozwiąż ćwiczenia i skorzystaj ze ściągi bez przepisywania gotowych rozwiązań.

**Zapamiętaj:** Kod SQL uruchamiaj w narzędziu bazy, np. phpMyAdmin, a kod PHP przez adres serwera, np. http://localhost/nauka/index.php.

# 1. Co wykonuje PHP?

PHP działa **po stronie serwera**. Przeglądarka wysyła żądanie, serwer wykonuje skrypt i odsyła wynik, najczęściej HTML. Przeglądarka nie wykonuje samego kodu PHP. Obejrzenie źródła prawidłowo obsłużonej strony pokaże wynik, a nie instrukcje PHP.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>Pierwszy skrypt</title>
</head>
<body>
    <h1>Witaj na stronie</h1>
    <?php
    echo "<p>Ten akapit utworzył PHP.</p>";
    ?>
</body>
</html>
```

## Uruchomienie krok po kroku

1. W lokalnym środowisku, np. XAMPP, uruchom serwer WWW. Bazę uruchom dopiero wtedy, gdy przykład jej wymaga.
2. W katalogu dokumentów serwera utwórz folder **nauka**. W typowym XAMPP jest to podfolder **htdocs**.
3. Zapisz kod jako **index.php**, w kodowaniu UTF-8.
4. Otwórz **http://localhost/nauka/index.php**. Jeżeli serwer używa innego portu, dodaj go do adresu.
5. Po zmianie pliku zapisz go i odśwież stronę.

## Co zobaczysz?

Nagłówek „Witaj na stronie” i akapit „Ten akapit utworzył PHP.”. Funkcja środowiska serwera polega tu na zastąpieniu bloku PHP wygenerowanym tekstem HTML.

**Zapamiętaj:** Dwuklik pliku i adres file:/// nie uruchamiają PHP. Jeśli widzisz surowy kod PHP albo przeglądarka pobiera plik, sprawdź konfigurację serwera. Zwykły podgląd HTML w edytorze nie wystarcza.

# 2. Znaczniki i składnia PHP

| Zapis | Znaczenie |
|---|---|
| <?php | Rozpoczyna blok PHP. Zalecana pełna forma. |
| ?> | Kończy blok PHP; dalej może występować HTML. |
| <?= $x ?> | Skrót wyświetlenia wartości, odpowiednik echo. |
| ; | Kończy większość instrukcji. |
| { ... } | Grupuje instrukcje, np. ciało warunku lub pętli. |
| // komentarz | Komentarz do końca linii. |
| /* komentarz */ | Komentarz mogący obejmować kilka linii. |

```
<?php
// Obliczamy cenę dwóch zeszytów.
$cena = 7;
$ilosc = 2;
$suma = $cena * $ilosc;
echo "Do zapłaty: " . $suma . " zł";
?>
```

**$cena = 7;** przypisuje liczbę do zmiennej. ***** oznacza mnożenie. **echo** wysyła wynik do odpowiedzi. **.** łączy fragmenty tekstu. Wynik: **Do zapłaty: 14 zł**.

Instrukcje wykonują się kolejno. Wcięcia nie zmieniają znaczenia tego przykładu, ale pomagają rozpoznać strukturę. Brak średnika może spowodować komunikat błędu wskazujący dopiero następną linię.

## Plik zawierający wyłącznie PHP

W pliku pomocniczym, np. db.php, zwykle pomija się końcowe **?>**. Chroni to przed przypadkowym wysłaniem pustych znaków po zamknięciu PHP, co może utrudnić użycie header() lub session_start().

**Zapamiętaj:** Nie używaj krótkiego otwarcia <? zależnego od konfiguracji. Bezpieczny, przenośny zapis to <?php. echo i isset są konstrukcjami języka, choć często omawia się je razem z funkcjami.

# 3. Zmienne, typy i napisy

```
$imie = "Ala";          // string: tekst
$wiek = 18;              // int: liczba całkowita
$cena = 12.50;           // float: liczba zmiennoprzecinkowa
$dostepny = true;        // bool: prawda albo fałsz
$brak = null;            // brak wartości
$kolory = ["biały", "czarny"]; // array: tablica

echo "Cześć, $imie!";
echo 'Cześć, $imie!';
```

Pierwsze echo wyświetli **Cześć, Ala!**, a drugie **Cześć, $imie!**. Podwójny cudzysłów umożliwia podstawianie prostych zmiennych; pojedynczy pozostawia zapis zmiennej dosłownie. Bez znacznika <br> oba wyniki pojawią się obok siebie.

## Zasady nazw i wartości

Nazwa zmiennej zaczyna się od **$**. Wybieraj czytelne nazwy, np. $liczbaSztuk. Nie zaczynaj nazwy po znaku $ od cyfry. Wielkość liter ma znaczenie: **$cena** i **$Cena** to różne zmienne. W kodzie ułamek dziesiętny zapisuj z kropką.

```
$liczba = "12";       // tekst, a nie int
$wartosc = (int) $liczba;
var_dump($liczba, $wartosc);

$produkt = "Notes";
echo '<p>Produkt: ' . $produkt . '</p>';
```

**(int)** konwertuje wartość na liczbę całkowitą. **var_dump()** pokazuje typ i wartość, tutaj string(2) "12" oraz int(12). To narzędzie diagnostyczne; usuń taki wydruk z gotowej strony.

## Konwersja to nie walidacja

Rzutowanie może zmienić niepoprawny tekst w liczbę, np. (int) "abc" daje 0. Nie dowodzi więc, że użytkownik wpisał prawidłową liczbę. Dane wejściowe najpierw sprawdzaj, a dopiero potem wykorzystuj.

**Zapamiętaj:** Dane zwykłych pól GET i POST trafiają do PHP jako napisy, a nazwy z [] mogą tworzyć tablice. Typ input="number" nie gwarantuje typu int po stronie PHP.

# 4. Operatory i porównania

| Grupa | Przykład | Znaczenie |
|---|---|---|
| Arytmetyczne | + - * / | Dodawanie, odejmowanie, mnożenie, dzielenie. |
| Reszta i potęga | 7 % 2; 2 ** 3 | Wyniki: 1 oraz 8. |
| Przypisanie | = += -= | $x += 2 to zwiększenie $x o 2. |
| Łączenie tekstu | . .= | $s .= "!" dopisuje wykrzyknik. |
| Równość | == === | == może konwertować typy; === wymaga zgodności typu i wartości. |
| Nierówność | != !== | !== sprawdza różnicę wartości lub typu. |
| Porządek | < <= > >= | Mniejszy, nie większy, większy, nie mniejszy. |
| Logiczne | && \|\| ! | I, LUB, zaprzeczenie. |
| Zwiększenie | ++ | $i++ zwiększa $i o jeden. |

```
var_dump(5 == "5");    // true
var_dump(5 === "5");   // false

$wiek = 18;
$zgoda = true;
$moze = ($wiek >= 18) && $zgoda;

$ilosc = 3;
$cena = 10;
$suma = ($ilosc * $cena) + 12; // 42
```

Wyrażenie po prawej stronie **=** jest obliczane, a wynik trafia do zmiennej po lewej. Dla **&&** oba warunki muszą być prawdziwe; dla **||** wystarczy co najmniej jeden.

**Zapamiętaj:** if ($x = 5) przypisuje wartość zamiast porównywać. Pisz if ($x === 5), jeśli oczekujesz liczby całkowitej 5. Używaj nawiasów, aby kolejność działań była czytelna. Przed dzieleniem sprawdź, czy dzielnik nie jest zerem.

# 5. Warunki: if i switch

```
$punkty = 65;
if ($punkty >= 80) {
    echo "Wysoki wynik";
} elseif ($punkty >= 50) {
    echo "Próg osiągnięty";
} else {
    echo "Poćwicz jeszcze";
}
```

PHP sprawdza warunki po kolei i wykonuje **pierwszą pasującą gałąź**. Dla 65 wyświetli „Próg osiągnięty”. Te progi są wyłącznie przykładem programistycznym, nie zasadami oceniania egzaminu. Osobne instrukcje if byłyby sprawdzane niezależnie.

```
$dostawa = "kurier";
switch ($dostawa) {
    case "kurier":
        $koszt = 20;
        break;
    case "odbior":
        $koszt = 0;
        break;
    default:
        $koszt = null;
}
```

**switch** porównuje wartość z kolejnymi przypadkami. **break** kończy jego wykonanie; pominięcie break powoduje przejście do kolejnych instrukcji. **default** obsługuje brak dopasowania. Switch stosuje porównania luźne; dla ścisłych warunków użyj np. if z ===.

## Krótsze zapisy

```
$opis = ($wiek >= 18) ? "pełnoletni" : "niepełnoletni";
$szukaj = $_GET["szukaj"] ?? "";
```

Operator **? :** wybiera jedną z dwóch wartości. Operator **??** zwraca wartość po prawej, gdy lewa nie istnieje lub ma wartość null. Nie sprawdza jednak, czy otrzymany tekst jest poprawny, ani czy nie jest tablicą.

# 6. Pętle: for, while, foreach

```
for ($i = 1; $i <= 3; $i++) {
    echo "<p>Numer: $i</p>";
}
```

**$i = 1** wykona się raz przed pętlą. **$i <= 3** jest sprawdzane przed każdym obiegiem. **$i++** wykonuje się po ciele pętli. Powstaną trzy akapity: Numer: 1, Numer: 2, Numer: 3.

```
$licznik = 1;
while ($licznik <= 3) {
    echo $licznik . " ";
    $licznik++;
}
```

**while** działa, dopóki warunek jest prawdziwy. Wynik: **1 2 3**. Bez zwiększania licznika ten przykład byłby pętlą nieskończoną. Przy pobieraniu rekordów warunek zmienia kolejne wywołanie funkcji fetch.

```
$ceny = [10, 20, 30];
$suma = 0;
foreach ($ceny as $cena) {
    $suma += $cena;
}
echo $suma; // 60
```

**foreach** kolejno podstawia elementy tablicy do zmiennej $cena. Akumulator $suma musi zostać ustawiony na 0 **przed** pętlą, inaczej wynik może być błędny.

| Konstrukcja | Kiedy ją wybrać |
|---|---|
| for | Znasz liczbę powtórzeń lub potrzebujesz licznika. |
| while | Powtarzasz do wyczerpania danych, np. wierszy zapytania. |
| foreach | Przechodzisz przez elementy tablicy. |
| break / continue | break przerywa pętlę; continue pomija resztę bieżącego obiegu. |
| do ... while | Sprawdza warunek po wykonaniu ciała; wykona się co najmniej raz. |

# 7. Tablice i klucze

```
$owoce = ["jabłko", "gruszka", "śliwka"];
echo $owoce[0];       // jabłko
$owoce[] = "banan";   // dopisanie elementu

echo count($owoce);   // 4

$produkt = ["id" => 7, "nazwa" => "Notes", "cena" => 12];
echo $produkt["nazwa"]; // Notes
```

**Tablica indeksowana** przechowuje elementy pod indeksami liczbowymi, zwykle od 0. **Tablica asocjacyjna** używa nazwanych kluczy. Operator **=>** łączy klucz z wartością. Podobną tablicę otrzymasz z mysqli_fetch_assoc().

```
$produkty = [
    ["nazwa" => "Notes", "cena" => 12],
    ["nazwa" => "Długopis", "cena" => 4]
];
foreach ($produkty as $produkt) {
    echo $produkt["nazwa"] . ": " . $produkt["cena"];
    echo "<br>";
}
```

To tablica tablic: zewnętrzna zawiera dwa produkty, a wewnętrzna opisuje jeden produkt. Wynik: Notes: 12 i Długopis: 4 w osobnych liniach.

```
sort($owoce);                    // sortowanie wartości
$jest = in_array("banan", $owoce, true);
$klucze = array_keys($produkt);
foreach ($produkt as $klucz => $wartosc) {
    echo $klucz . ": " . $wartosc . "<br>";
}
```

**sort()** zmienia tablicę i nadaje ponownie indeksy liczbowe. **asort()** sortuje wartości z zachowaniem kluczy, a **ksort()** sortuje według kluczy. **in_array(..., true)** szuka wartości ze ścisłym porównaniem typów.

**Zapamiętaj:** Nie odczytuj klucza, który może nie istnieć, bez sprawdzenia lub operatora ??. count() liczy elementy, a nie ostatni indeks. Usunięcie elementu może pozostawić przerwę w indeksach.

# 8. Funkcje tekstowe

| Funkcja i przykład | Co otrzymasz / do czego służy |
|---|---|
| trim(" Ala ") | "Ala"; usuwa określone białe znaki z obu końców. |
| strlen("PHP") | 3; długość w bajtach, nie zawsze liczba znaków. |
| mb_strlen("Łódź", "UTF-8") | 4; długość w znakach. Wymaga rozszerzenia mbstring. |
| strtolower("PHP") | "php"; dla polskich liter użyj mb_strtolower(). |
| strtoupper("php") | "PHP"; odpowiednik dla wielkich liter. |
| substr("abcdef", 1, 3) | "bcd"; pozycje liczone od 0, operacja bajtowa. |
| strpos("informatyka", "mat") | 5; pozycja znalezionego fragmentu lub false. |
| str_replace("kot", "pies", "kot śpi") | "pies śpi"; zastępuje podany tekst. |
| explode(";", "Ala;Ola") | Tablica ["Ala", "Ola"]. |
| implode(", ", ["Ala", "Ola"]) | Tekst "Ala, Ola". |

```
$email = "  uczen@example.com  ";
$email = trim($email);

$tekst = "PHP jest przydatny";
if (strpos($tekst, "PHP") !== false) {
    echo "Znaleziono PHP";
}
```

Fragment „PHP” zaczyna się na pozycji **0**. Zwykłe if (strpos(...)) uznałoby zero za fałsz. Dlatego wynik trzeba porównać z **false** za pomocą !==.

**Zapamiętaj:** trim() porządkuje tekst, lecz nie zabezpiecza SQL ani HTML. Przy polskich znakach stosuj funkcje mb_* do operacji na znakach, jeśli rozszerzenie mbstring jest dostępne.

# 9. Liczby, daty i formatowanie

```
$cena = 12.5;
$ilosc = 3;
$suma = $cena * $ilosc;
echo number_format($suma, 2, ",", " ") . " zł";
// 37,50 zł

echo round(3.456, 2); // 3.46
echo ceil(3.1);       // 4
echo floor(3.9);      // 3
echo abs(-7);         // 7
```

**number_format()** zwraca tekst do wyświetlenia: wartość, liczba miejsc dziesiętnych, separator dziesiętny i separator tysięcy. Obliczenia wykonuj na liczbach, a formatowanie zastosuj na końcu.

| Funkcja | Zastosowanie |
|---|---|
| min(), max() | Najmniejsza i największa wartość. |
| sqrt(25), pow(2, 3) | Pierwiastek 5 i potęga 8. Można też użyć 2 ** 3. |
| rand(1, 6) | Losowa liczba całkowita od 1 do 6; do prostych ćwiczeń. |
| random_int(1, 6) | Losowanie liczby całkowitej odpowiednie także do zastosowań wymagających bezpiecznego generatora. |
| is_numeric($x) | Sprawdza, czy wartość jest liczbą lub napisem liczbowym; nie wymusza całkowitości ani zakresu. |

```
date_default_timezone_set("Europe/Warsaw");
echo date("Y-m-d");  // rok-miesiąc-dzień
echo date("H:i:s");  // godzina:minuta:sekunda
$teraz = time();     // znacznik czasu Unix
```

**Y** oznacza czterocyfrowy rok, **m** miesiąc, **d** dzień, **H** godzinę w systemie 24-godzinnym, **i** minuty, **s** sekundy. Nie myl miesiąca m z minutami i.

**Zapamiętaj:** Float ma ograniczoną precyzję. W prostym kalkulatorze wystarcza, ale w rzeczywistych rozliczeniach używa się np. całkowitej liczby groszy. W bazie kwotę można przechowywać jako DECIMAL.

# 10. Własne funkcje i osobne pliki

```
function poleProstokata(float $a, float $b): float
{
    return $a * $b;
}

$pole = poleProstokata(4, 5);
echo "Pole: " . $pole; // Pole: 20
```

**function** rozpoczyna definicję. **$a i $b** to parametry, czyli nazwy używane wewnątrz funkcji. **4 i 5** to argumenty przekazane przy wywołaniu. **return** zwraca wynik i kończy działanie funkcji. Adnotacje float określają oczekiwane typy; nie są konieczne w najprostszym zapisie.

Zmienne utworzone wewnątrz funkcji mają zwykle zasięg lokalny. Nie zakładaj, że funkcja automatycznie widzi każdą zmienną z głównego skryptu. Przekaż potrzebne dane w argumentach.

## Przydatny pomocnik do wyświetlania

```
function e(string $tekst): string
{
    return htmlspecialchars($tekst, ENT_QUOTES, "UTF-8");
}

echo "<p>" . e("Ala & Ola") . "</p>";
```

Funkcja e() zwraca tekst z zamienionymi znakami specjalnymi HTML. W kodzie odpowiedzi powstanie Ala & Ola, ale użytkownik zobaczy Ala & Ola. Funkcja nie wysyła wyniku sama - robi to echo.

| Instrukcja | Zachowanie |
|---|---|
| include "plik.php"; | Dołącza plik; brak pliku powoduje ostrzeżenie, skrypt może działać dalej. |
| require "plik.php"; | Dołącza wymagany plik; niepowodzenie prowadzi do błędu przerywającego zwykły przebieg. |
| require_once "plik.php"; | Jak require, ale nie dołącza ponownie już dołączonego pliku. |

**Zapamiętaj:** Dla stałej ścieżki używaj np. require_once __DIR__ . "/db.php";. __DIR__ to katalog bieżącego pliku. Nie buduj ścieżki include bezpośrednio z danych formularza.

# 11. Formularz HTML - budowa

```
<form action="wynik.php" method="post">
    <label for="imie">Imię:</label>
    <input type="text" id="imie" name="imie" required>

    <label for="ilosc">Liczba sztuk:</label>
    <input type="number" id="ilosc" name="ilosc"
           min="1" max="100" step="1" required>

    <button type="submit">Wyślij</button>
</form>
```

| Element / atrybut | Objaśnienie |
|---|---|
| form | Obejmuje pola należące do formularza. |
| action="wynik.php" | Adres skryptu, który odbierze dane. Bez action formularz wysyłany jest na bieżący adres. |
| method="post" | Wskazuje sposób przesłania danych. Odczyt: $_POST. |
| name="ilosc" | Klucz przesyłanej wartości: $_POST["ilosc"]. |
| id="ilosc" | Identyfikator elementu w HTML, CSS i JS. Sam nie wyznacza klucza w PHP. |
| label for="ilosc" | Łączy etykietę z elementem mającym id="ilosc". |
| required | Wymaga uzupełnienia pola podczas zwykłej walidacji przeglądarki. |
| min / max / step | Granice oraz krok wartości pola liczbowego. |
| type="submit" | Uruchamia wysłanie formularza. |

## Które dane zostaną wysłane?

Zwykłe pole musi mieć **name**, aby jego wartość znalazła się w danych formularza. Pole **disabled** nie jest wysyłane. Pole **readonly** zwykle jest wysyłane. Niezaznaczony checkbox jest pomijany. Przycisk z name i value przesyła swoją parę, gdy został użyty do wysłania.

**Zapamiętaj:** Walidacja HTML poprawia wygodę użytkownika, ale można ją ominąć. Wszystkie istotne warunki trzeba sprawdzić ponownie w PHP. Hidden, readonly i select nie czynią przesłanej wartości zaufaną.

# 12. GET, POST i superglobalne

| Cecha | GET | POST |
|---|---|---|
| Odczyt w PHP | $_GET["nazwa"] | $_POST["nazwa"] |
| Typowe miejsce danych | Część adresu po znaku ? | Treść żądania HTTP |
| Zastosowanie | Wyszukiwanie, filtrowanie, wybór rekordu | Zapis danych i operacje zmieniające stan |
| Zakładka / udostępnienie | Parametry można zachować w adresie | Treść POST nie jest częścią zakładki |
| Poufność | Dane mogą być w historii i logach adresów | Nie są w URL, ale POST sam nie szyfruje transmisji |

```
<!-- Po wysłaniu: szukaj.php?q=notes -->
<form action="szukaj.php" method="get">
    <input type="search" name="q">
    <button type="submit">Szukaj</button>
</form>
```

Znak **?** rozpoczyna parametry URL, a **&** oddziela pary, np. lista.php?kategoria=2&limit=10. Kodowanie adresu obsługuje przeglądarka. Nie traktuj parametrów URL jako zaufanych.

| Tablica | Co zawiera |
|---|---|
| $_GET | Parametry zapytania w adresie URL. |
| $_POST | Dane typowego formularza POST. |
| $_SERVER | Informacje o żądaniu i środowisku, np. REQUEST_METHOD. |
| $_FILES | Informacje o przesłanych plikach. |
| $_SESSION / $_COOKIE | Dane sesji / ciasteczka otrzymane od przeglądarki. |

```
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    // To żądanie przesłano metodą POST.
}
```

**Zapamiętaj:** POST nie oznacza szyfrowania - transmisję chroni HTTPS. Unikaj $_REQUEST, gdy wiesz, czy oczekujesz GET, czy POST: jawny wybór tablicy ułatwia analizę źródła danych.

# 13. Odbiór POST krok po kroku

Ten kompletny plik **powitanie.php** zawiera formularz i jego obsługę. Przy pierwszym wejściu przeglądarka zwykle wysyła GET, więc blok obsługi POST nie zostaje wykonany.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>Powitanie</title>
</head>
<body>
<form method="post">
    <label for="imie">Imię:</label>
    <input id="imie" name="imie" required>
    <button type="submit">Przywitaj</button>
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $imie = $_POST["imie"] ?? "";
    if (!is_string($imie) || trim($imie) === "") {
        echo "<p>Podaj imię.</p>";
    } else {
        $imie = trim($imie);
        $bezpieczne = htmlspecialchars(
            $imie, ENT_QUOTES, "UTF-8"
        );
        echo "<p>Witaj, " . $bezpieczne . "!</p>";
    }
}
?>
</body>
</html>
```

**1.** Warunek REQUEST_METHOD oddziela pierwsze otwarcie strony od przetwarzania formularza.
**2.** ?? podstawia pusty tekst, jeśli nie ma pola imie.
**3.** is_string() odrzuca np. tablicę przesłaną zamiast tekstu. Operator || nie oblicza drugiej części, gdy pierwsza jest prawdziwa.
**4.** trim() usuwa skrajne białe znaki; samo wpisanie spacji nie wystarczy.
**5.** htmlspecialchars() przygotowuje tekst do HTML.
**6.** echo tworzy akapit, np. „Witaj, Ala!”.

**Zapamiętaj:** Spotkasz też if (isset($_POST["wyslij"])). Taki kod zależy od przesłania konkretnego przycisku. Sprawdzanie metody POST lepiej opisuje zamiar: obsłuż dane formularza.

# 14. Walidacja: tekst i liczby

| Narzędzie | Co rzeczywiście sprawdza |
|---|---|
| isset($x) | Czy zmienna / klucz istnieje i nie ma wartości null. |
| empty($x) | Czy wartość jest uznawana za pustą, np. "", "0", 0, false, null lub []. |
| is_string($x) | Czy wartość jest tekstem; przydatne przed trim(). |
| filter_input(...) | Pobiera i filtruje wartość wejściową. Domyślny filtr nie oznacza walidacji. |
| filter_var(...) | Filtruje przekazaną wartość, np. już pobraną zmienną. |

```
$ilosc = filter_input(
    INPUT_POST,
    "ilosc",
    FILTER_VALIDATE_INT,
    ["options" => ["min_range" => 1, "max_range" => 100]]
);

if ($ilosc === null || $ilosc === false) {
    echo "Podaj liczbę całkowitą od 1 do 100.";
} else {
    echo "Poprawna liczba sztuk: " . $ilosc;
}
```

**INPUT_POST** wybiera dane POST; „ilosc” jest nazwą pola. **FILTER_VALIDATE_INT** sprawdza liczbę całkowitą. Opcje ograniczają zakres. Przy takim wywołaniu **null** oznacza brak pola, a **false** błąd walidacji. Poprawny wynik jest liczbą całkowitą.

```
$email = $_POST["email"] ?? "";
if (!is_string($email) ||
    filter_var(trim($email), FILTER_VALIDATE_EMAIL) === false) {
    echo "Niepoprawny adres e-mail.";
}
```

Sprawdzenie formatu e-maila nie dowodzi istnienia skrzynki. Podobnie poprawna liczba nie oznacza, że użytkownik ma prawo zmienić wybrany rekord.

**Zapamiętaj:** Nie używaj empty() do sprawdzenia, czy przesłano dozwolone zero: empty("0") daje true. Ściśle odróżniaj false od 0. Walidacja odpowiada na pytanie „czy dane są dopuszczalne?”, a nie „jak je bezpiecznie wyświetlić?”.

# 15. Select, radio i checkbox

```
<select name="dostawa">
    <option value="kurier">Kurier - 20 zł</option>
    <option value="odbior">Odbiór - 0 zł</option>
</select>
<label>
    <input type="radio" name="platnosc" value="karta">
    Karta
</label>
<label>
    <input type="radio" name="platnosc" value="gotowka">
    Gotówka
</label>
<label>
    <input type="checkbox" name="zgoda" value="tak">
    Akceptuję warunki
</label>
```

W select przesyłane jest **value wybranej opcji**, nie jej widoczny opis. Pola radio o tym samym name tworzą grupę wyboru. Checkbox przesyła swoją wartość tylko wtedy, gdy jest zaznaczony.

```
$cennik = ["kurier" => 20, "odbior" => 0];
$dostawa = $_POST["dostawa"] ?? "";
if (!is_string($dostawa) || !isset($cennik[$dostawa])) {
    echo "Wybierz prawidłową dostawę.";
} else {
    $kosztDostawy = $cennik[$dostawa];
}
$zgoda = ($_POST["zgoda"] ?? "") === "tak";
```

Koszt jest wybierany z tablicy po stronie serwera. Nie pobieramy ceny dostawy z ukrytego pola, bo użytkownik mógłby ją zmienić. isset() prawidłowo rozpoznaje wartość 0 jako istniejącą.

## Kilka wartości pod jedną nazwą

```
<input type="checkbox" name="dodatki[]" value="pudelko">
<input type="checkbox" name="dodatki[]" value="wstazka">
```

Zapis **dodatki[]** tworzy tablicę $_POST["dodatki"]. Gdy nic nie zaznaczono, użyj wartości domyślnej []. Sprawdź is_array(), a potem zweryfikuj każdy element względem listy dozwolonych napisów. Nie podawaj całej tablicy do echo.

# 16. Kalkulator zamówienia

Poniższe części umieść w pliku **kalkulator.php**, wewnątrz body zwykłego szkieletu HTML. Cenę jednostkową 15 zł ustala serwer. Formularz podaje tylko ilość i wariant dostawy.

```
<form method="post">
    <label>Liczba sztuk:
        <input type="number" name="ilosc" min="1"
               max="100" step="1" required>
    </label>
    <select name="dostawa">
        <option value="kurier">Kurier - 20 zł</option>
        <option value="odbior">Odbiór - 0 zł</option>
    </select>
    <button type="submit">Oblicz</button>
</form>
<?php
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $ilosc = filter_input(INPUT_POST, "ilosc",
        FILTER_VALIDATE_INT,
        ["options" => ["min_range" => 1, "max_range" => 100]]);
    $dostawa = $_POST["dostawa"] ?? "";
    $koszty = ["kurier" => 20, "odbior" => 0];

    if ($ilosc === null || $ilosc === false ||
        !is_string($dostawa) || !isset($koszty[$dostawa])) {
        echo "<p>Sprawdź ilość i sposób dostawy.</p>";
    } else {
        $cena = 15;
        $suma = $ilosc * $cena + $koszty[$dostawa];
        echo "<p>Do zapłaty: " .
            number_format($suma, 2, ",", " ") . " zł</p>";
    }
}
?>
```

**Kolejność:** sprawdzenie metody → odbiór danych → walidacja → obliczenie → formatowanie → HTML. Dla 3 sztuk i kuriera wynik wynosi 3 × 15 + 20 = **65,00 zł**. Dla odbioru: **45,00 zł**.

**Zapamiętaj:** Przetestuj również brak pola, 0, -2, 101, ułamek i nieznaną dostawę. Przeglądarka może nie dopuścić części prób; to nie zwalnia skryptu z walidacji żądań po stronie serwera.

# 17. Bezpieczne wyświetlanie

Dane mogą zawierać znaki mające znaczenie w HTML, np. <, >, & i cudzysłowy. Jeśli wstawisz je bez przygotowania do odpowiedzi, przeglądarka może potraktować je jako kod strony. Dotyczy to również tekstów odczytanych z bazy.

```
$tekst = '<b>Ala & Ola</b>';
echo htmlspecialchars($tekst, ENT_QUOTES, "UTF-8");
```

Na stronie pojawi się dosłownie **Ala & Ola**, bez pogrubienia pochodzącego z tekstu użytkownika. ENT_QUOTES obejmuje oba rodzaje cudzysłowów; UTF-8 określa kodowanie wejściowe.

## Zachowanie wpisanej wartości w formularzu

```
<?php
$imie = $_POST["imie"] ?? "";
if (!is_string($imie)) {
    $imie = "";
}
?>
<input name="imie" value="<?= htmlspecialchars(
    $imie, ENT_QUOTES, 'UTF-8'
) ?>">
```

Atrybut value jest objęty cudzysłowami. Przetworzenie tekstu chroni przed zamknięciem wartości atrybutu przez cudzysłów wpisany przez użytkownika. W zwykłym elemencie textarea zawartość wpisuje się między znacznikami, również po przetworzeniu.

| Problem | Właściwe narzędzie |
|---|---|
| Czy ilość jest dodatnią liczbą? | Walidacja typu i zakresu w PHP. |
| Czy tekst może zmienić strukturę HTML? | htmlspecialchars() w tekście HTML i cytowanych atrybutach. |
| Czy dane zmienią strukturę SQL? | Parametry zapytania przygotowanego. |
| Czy transmisja jest szyfrowana? | HTTPS. |
| Czy użytkownik ma prawo zmienić rekord? | Kontrola uprawnień w aplikacji. |

**Zapamiętaj:** htmlspecialchars() nie zabezpiecza zapytań SQL. Nie jest też uniwersalną ochroną dla JavaScript, CSS czy adresów URL. Przykłady w tym podręczniku używają go w tekście HTML i zwykłych atrybutach w cudzysłowach.

# 18. Baza do ćwiczeń

Kolejne przykłady korzystają z jednej bazy **inf03_nauka**. W phpMyAdmin otwórz zakładkę SQL i uruchom poniższy kod. Twórz tę bazę w lokalnym środowisku ćwiczeniowym. Jeżeli istnieje już baza o tej nazwie, użyj nowej nazwy i odpowiednio zmień konfigurację PHP.

```
CREATE DATABASE inf03_nauka
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;
USE inf03_nauka;

CREATE TABLE produkty (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    nazwa VARCHAR(80) NOT NULL,
    cena DECIMAL(8,2) NOT NULL,
    stan INT UNSIGNED NOT NULL DEFAULT 0
);

INSERT INTO produkty (nazwa, cena, stan) VALUES
('Notes', 12.50, 8),
('Długopis', 4.00, 20),
('Plecak', 99.90, 3);
```

| Element | Znaczenie |
|---|---|
| Baza / tabela / rekord / kolumna | Baza zawiera tabele. Tabela produkty ma rekordy opisujące produkty i kolumny określające ich cechy. |
| PRIMARY KEY | Jednoznacznie identyfikuje rekord. W przykładzie: id. |
| AUTO_INCREMENT | Serwer nadaje kolejne identyfikatory; nie wpisujesz id przy INSERT. |
| VARCHAR(80) | Tekst do 80 znaków. |
| DECIMAL(8,2) | Liczba dziesiętna: 8 cyfr łącznie, z czego 2 po przecinku. |
| NOT NULL / DEFAULT 0 | Wartość nie może być NULL / domyślnie przyjmuje zero. |
| utf8mb4 | Zestaw znaków obsługujący m.in. polskie litery. |

**Zapamiętaj:** SQL i PHP to różne języki. Zmienna $sql w PHP przechowuje tekst zapytania; dopiero wysłanie go funkcją MySQLi powoduje wykonanie na serwerze bazy.

# 19. SQL potrzebny w PHP

```
SELECT id, nazwa, cena FROM produkty;

SELECT nazwa, cena FROM produkty
WHERE cena <= 20 AND stan > 0
ORDER BY cena ASC, id ASC;

SELECT id, nazwa FROM produkty
ORDER BY cena DESC, id ASC LIMIT 2;

SELECT COUNT(*) AS ile, AVG(cena) AS srednia
FROM produkty;
```

**SELECT** wybiera kolumny, **FROM** wskazuje tabelę, **WHERE** filtruje rekordy, **ORDER BY** sortuje, a **LIMIT** ogranicza liczbę wyników. ASC to rosnąco, DESC - malejąco. Drugi warunek sortowania rozstrzyga remisy.

Dla danych początkowych drugie zapytanie zwróci Długopis (4,00) i Notes (12,50). Trzecie zwróci Plecak i Notes. **AS** nadaje alias kolumnie wyniku, np. później odczytasz $wiersz["ile"]. COUNT(*) liczy rekordy, a AVG() oblicza średnią.

## Łączenie tabel - osobny przykład schematu

```
-- Założenie: istnieją tabele kategorie oraz towary,
-- a towary.kategoria_id wskazuje kategorie.id.
SELECT t.nazwa, k.nazwa AS kategoria
FROM towary AS t
JOIN kategorie AS k ON t.kategoria_id = k.id;
```

JOIN zestawia pasujące rekordy na podstawie relacji. Alias kategoria usuwa niejednoznaczność dwóch kolumn o nazwie nazwa. Ten fragment ilustruje inny schemat i nie uruchomi się na samej tabeli produkty.

## Modyfikacje danych

```
INSERT INTO produkty (nazwa, cena, stan)
VALUES ('Teczka', 8.50, 10);
UPDATE produkty SET stan = 9 WHERE id = 1;
DELETE FROM produkty WHERE id = 3;
```

**Zapamiętaj:** UPDATE i DELETE bez WHERE mogą objąć wszystkie rekordy. Najpierw upewnij się przez SELECT z tym samym warunkiem, które wiersze zostaną wybrane. Tekst w SQL ujmuje się zwykle w pojedyncze apostrofy; liczb nie trzeba.

# 20. Połączenie z MySQL / MariaDB

Zapisz poniższy kompletny plik jako **db.php**. W następnych fragmentach require_once dołącza ten plik i udostępnia zmienną **$db**. Dane połączenia muszą odpowiadać lokalnej konfiguracji.

```
<?php
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);

try {
    $db = mysqli_connect(
        "localhost", "root", "", "inf03_nauka"
    );
    mysqli_set_charset($db, "utf8mb4");
} catch (mysqli_sql_exception $e) {
    error_log($e->getMessage());
    http_response_code(500);
    exit("Nie można połączyć z bazą danych.");
}
```

**mysqli_report()** jawnie włącza raportowanie błędów przez wyjątki; zachowanie nie zależy od domyślnego ustawienia danej wersji PHP.
**mysqli_connect()** otrzymuje kolejno host, użytkownika, hasło i nazwę bazy. Po udanym połączeniu zwraca obiekt mysqli przechowywany w $db.
**mysqli_set_charset()** ustawia kodowanie połączenia, niezależne od meta charset strony.
**try / catch** przechwytuje błąd połączenia lub ustawienia kodowania.
**error_log()** zapisuje szczegóły w logu, a **exit()** kończy skrypt z krótkim komunikatem.

## Wykorzystanie połączenia

```
require_once __DIR__ . "/db.php";
// Tutaj zapytania do bazy.
mysqli_close($db);
```

Połączenie zamknij po ostatniej operacji na bazie. Nie zamykaj go przed drugim skryptem korzystającym z tej samej zmiennej. Przy końcu żądania zwykłe połączenie jest zwalniane automatycznie, ale jawne zamknięcie pokazuje pełną kolejność pracy.

**Zapamiętaj:** root z pustym hasłem jest przykładem lokalnej konfiguracji ćwiczeniowej, a nie zaleceniem dla publicznej strony. Na egzaminie użyj danych podanych w zadaniu. W PHP 8.1+ domyślny tryb MySQLi zgłasza wyjątki; stare if (!$db) nie muszą przejąć błędu.

# 21. SELECT i obiekt wyniku

```
require_once __DIR__ . "/db.php";

$sql = "SELECT id, nazwa, cena FROM produkty ORDER BY id";
$wynik = mysqli_query($db, $sql);

$ile = mysqli_num_rows($wynik);
echo "Liczba produktów: " . $ile;

while ($wiersz = mysqli_fetch_assoc($wynik)) {
    echo "<p>" . htmlspecialchars(
        $wiersz["nazwa"], ENT_QUOTES, "UTF-8"
    ) . "</p>";
}

mysqli_free_result($wynik);
mysqli_close($db);
```

## Pięć różnych rzeczy, których nie wolno pomylić

| Nazwa | Co zawiera |
|---|---|
| $db | Połączenie z serwerem bazy danych. |
| $sql | Napis zawierający polecenie SQL; samo utworzenie napisu nie uruchamia zapytania. |
| $wynik | Dla poprawnego SELECT: obiekt mysqli_result zawierający zestaw wyników. |
| $wiersz | Jednorazowo pobrany rekord jako tablica. |
| $wiersz["nazwa"] | Wartość jednej kolumny w aktualnie pobranym rekordzie. |

Dla początkowej bazy liczba produktów wynosi **3**, a poniżej pojawią się trzy akapity. **mysqli_num_rows()** liczy wiersze wyniku, nie kolumny. **mysqli_free_result()** zwalnia pamięć wyniku i nie usuwa danych z bazy.

**Zapamiętaj:** Poprawny SELECT bez pasujących rekordów nadal zwraca obiekt wyniku - z liczbą wierszy 0. To nie jest błąd SQL. INSERT, UPDATE i DELETE bez klauzul zwracających dane dają przy powodzeniu true, więc nie wolno przekazywać ich wyniku do fetch_assoc().

# 22. fetch_assoc, row i array

Załóżmy zapytanie **SELECT nazwa, cena FROM produkty ORDER BY id**. Pierwszy rekord zawiera Notes i 12.50. Funkcje fetch pobierają następny wiersz i przesuwają pozycję w wyniku.

| Funkcja | Sposób odczytu jednego wiersza |
|---|---|
| mysqli_fetch_assoc($wynik) | $w["nazwa"], $w["cena"] - klucze według nazw kolumn / aliasów. |
| mysqli_fetch_row($wynik) | $w[0], $w[1] - indeksy według kolejności w SELECT. |
| mysqli_fetch_array($wynik) | Domyślnie MYSQLI_BOTH: dostęp przez nazwy i indeksy. |
| mysqli_fetch_array($wynik, MYSQLI_ASSOC) | Tylko klucze tekstowe; odpowiednik stylu fetch_assoc. |

```
while ($w = mysqli_fetch_row($wynik)) {
    $nazwa = $w[0];
    $cena = $w[1];
    echo htmlspecialchars($nazwa, ENT_QUOTES, "UTF-8");
    echo " - " . number_format((float) $cena, 2, ",", " ");
    echo "<br>";
}
```

W tym fragmencie $wynik musi pochodzić z podanego wyżej zapytania. Po wyczerpaniu rekordów fetch zwraca **null** i pętla się kończy; API przewiduje też false przy niepowodzeniu. Warunek wykorzystuje przypisanie, więc tutaj znak **=** jest celowy.

## Częsta pułapka: zgubiony pierwszy rekord

```
$pierwszy = mysqli_fetch_assoc($wynik);
// Pierwszy rekord został już pobrany!
while ($w = mysqli_fetch_assoc($wynik)) {
    // Pętla zacznie od drugiego rekordu.
}
```

Nie pobieraj wiersza „na próbę” przed pętlą, jeżeli później chcesz pokazać wszystkie. Ponowne przejście wymaga np. ponownego wykonania SELECT albo przestawienia pozycji buforowanego wyniku przez mysqli_data_seek($wynik, 0).

**Zapamiętaj:** fetch_array() bez drugiego argumentu nie jest wyłącznie tablicą asocjacyjną. Domyślnie tworzy również indeksy liczbowe. Dla czytelności wybieraj fetch_assoc() albo jawnie określaj tryb.

# 23. Wynik w tabeli, liście i karcie

Fragment umieść wewnątrz body pliku PHP. Plik db.php musi znajdować się w tym samym katalogu. Nagłówki tabeli powstają raz, a jeden rekord bazy tworzy jeden wiersz HTML.

```
<?php
require_once __DIR__ . "/db.php";
$wynik = mysqli_query($db,
    "SELECT nazwa, cena FROM produkty ORDER BY id");
?>
<table>
    <tr><th>Produkt</th><th>Cena</th></tr>
    <?php while ($w = mysqli_fetch_assoc($wynik)): ?>
        <tr>
            <td><?= htmlspecialchars(
                $w["nazwa"], ENT_QUOTES, "UTF-8"
            ) ?></td>
            <td><?= number_format(
                (float) $w["cena"], 2, ",", " "
            ) ?> zł</td>
        </tr>
    <?php endwhile; ?>
</table>
<?php
if (mysqli_num_rows($wynik) === 0) {
    echo "<p>Brak produktów.</p>";
}
mysqli_free_result($wynik);
mysqli_close($db);
?>
```

**while (...): ... endwhile;** to alternatywna składnia pętli, wygodna przy przeplataniu PHP z HTML. Działa jak wersja z klamrami. Każde wypisuje wyrażenie w bieżącym miejscu dokumentu.

## Jak zmienić sposób prezentacji?

**Lista:** otwórz <ul> przed pętlą, dla każdego rekordu twórz <li> z nazwą, a </ul> umieść po pętli.
**Karta:** wewnątrz pętli utwórz <article>, nagłówek <h2> i akapit z ceną.
**Obraz:** używaj kontrolowanej nazwy pliku lub sprawdzonej ścieżki; samo kodowanie HTML nie weryfikuje bezpieczeństwa adresu.

**Zapamiętaj:** Zamykaj każdą komórkę i każdy wiersz. Nie twórz osobnej tabeli w każdym obiegu, chyba że właśnie taki rezultat jest wymagany. CSS formatuje wygenerowany HTML niezależnie od tego, skąd pochodziły dane.

# 24. Zapytania przygotowane

Jeżeli wartość pochodzi z formularza lub URL, przekaż ją jako **parametr**. Zapytanie przygotowane oddziela polecenie SQL od danych. Samo korzystanie z biblioteki MySQLi nie zabezpiecza przed SQL Injection.

```
require_once __DIR__ . "/db.php";
$limitCeny = 20.0;

$stmt = mysqli_prepare($db,
    "SELECT nazwa, cena FROM produkty WHERE cena <= ?");
mysqli_stmt_bind_param($stmt, "d", $limitCeny);
mysqli_stmt_execute($stmt);
mysqli_stmt_bind_result($stmt, $nazwa, $cena);

while (mysqli_stmt_fetch($stmt)) {
    echo htmlspecialchars($nazwa, ENT_QUOTES, "UTF-8");
    echo " - " . number_format((float) $cena, 2, ",", " ");
    echo "<br>";
}
mysqli_stmt_close($stmt);
mysqli_close($db);
```

**1. prepare:** tworzy instrukcję z miejscem na wartość oznaczonym znakiem ?. Nie wpisuj apostrofów wokół pytajnika.
**2. bind_param:** wiąże zmienną $limitCeny z parametrem; „d” oznacza liczbę zmiennoprzecinkową. Przekazuje się zmienną, nie gotowe wyrażenie.
**3. execute:** uruchamia instrukcję z aktualnymi wartościami parametrów.
**4. bind_result:** łączy kolumny SELECT z podanymi zmiennymi w tej samej kolejności.
**5. fetch:** pobiera kolejne rekordy do $nazwa i $cena.
**6. close:** zwalnia instrukcję; osobno zamykamy połączenie.

| Litera | Znaczenie | Przykład wiązania |
|---|---|---|
| i | Liczba całkowita | "i", $id |
| d | Liczba zmiennoprzecinkowa | "d", $cena |
| s | Tekst | "s", $nazwa |
| b | Dane binarne / duże bloki | Specjalistyczne użycie, m.in. send_long_data(). |

**Zapamiętaj:** Liczba liter typu i liczba zmiennych muszą odpowiadać liczbie ?. Parametry zastępują wartości, a nie nazwy tabel, kolumn ani kierunek ASC / DESC. Takie elementy wybieraj z listy dozwolonych stałych.

# 25. Wyszukiwanie z GET

Formularz można umieścić nad kodem PHP w body pliku **szukaj.php**. Wyszukiwany fragment nazwy przekazujemy do LIKE jako parametr.

```
<form method="get">
    <input type="search" name="q" maxlength="80">
    <button type="submit">Szukaj produktu</button>
</form>
<?php
$q = $_GET["q"] ?? "";
if (!is_string($q)) {
    exit("Niepoprawne zapytanie.");
}
$q = trim($q);
require_once __DIR__ . "/db.php";
$wzorzec = "%" . $q . "%";
$stmt = mysqli_prepare($db,
    "SELECT nazwa FROM produkty WHERE nazwa LIKE ?
     ORDER BY nazwa");
mysqli_stmt_bind_param($stmt, "s", $wzorzec);
mysqli_stmt_execute($stmt);
mysqli_stmt_bind_result($stmt, $nazwa);
$ile = 0;
while (mysqli_stmt_fetch($stmt)) {
    echo "<p>" . htmlspecialchars(
        $nazwa, ENT_QUOTES, "UTF-8"
    ) . "</p>";
    $ile++;
}
if ($ile === 0) {
    echo "<p>Nie znaleziono produktów.</p>";
}
mysqli_stmt_close($stmt);
mysqli_close($db);
?>
```

**%** w LIKE oznacza dowolny ciąg znaków. Wzorzec %not% może dopasować Notes, zależnie od reguł porównywania w bazie. Puste q daje %% i w tym przykładzie wyświetla wszystkie produkty. _ w LIKE oznacza pojedynczy znak. Znaki % i _ wpisane przez użytkownika również działają jako symbole wzorca; parametr zabezpiecza strukturę SQL, ale nie wyłącza ich znaczenia.

**Zapamiętaj:** Inny wariant to mysqli_stmt_get_result($stmt), a potem fetch_assoc(). Wymaga mysqlnd. W przykładach używamy bind_result() + fetch(), aby nie uzależniać od tego sposobu pobierania wyniku.

# 26. INSERT - zapis formularza

Po sprawdzeniu danych formularza możesz dodać rekord. Poniższy samodzielny fragment PHP ilustruje sam zapis ustalonych wartości; odbiór i walidację danych zawiera kompletny projekt na stronach 30-33.

```
require_once __DIR__ . "/db.php";
$nazwa = "Teczka";
$cena = 8.50;
$stan = 10;

$stmt = mysqli_prepare($db,
    "INSERT INTO produkty (nazwa, cena, stan) VALUES (?, ?, ?)");
mysqli_stmt_bind_param($stmt, "sdi", $nazwa, $cena, $stan);
mysqli_stmt_execute($stmt);

$dodano = mysqli_stmt_affected_rows($stmt);
$noweId = mysqli_insert_id($db);
echo "Dodano rekordów: " . $dodano;
echo "; nowe ID: " . $noweId;

mysqli_stmt_close($stmt);
mysqli_close($db);
```

**sdi** oznacza kolejno tekst nazwy, liczbę ceny i całkowity stan magazynowy. Kolejność typów odpowiada kolejności parametrów i kolumn. Nie wpisujemy id, bo baza nadaje je przez AUTO_INCREMENT.

**mysqli_stmt_affected_rows()** zwraca liczbę rekordów objętych operacją na instrukcji; po pojedynczym udanym INSERT oczekujemy 1. **mysqli_insert_id()** odczytuje identyfikator wygenerowany przez poprzednią odpowiednią operację w tym połączeniu. Pobierz go przed innymi zapisami.

## Co należy sprawdzić przed zapisem?

Nazwa: czy jest tekstem, czy po trim() nie jest pusta i czy mieści się w kolumnie. Cena: czy jest liczbą w dozwolonym zakresie. Stan: czy jest liczbą całkowitą nie mniejszą niż 0. Nie wywołuj execute(), gdy walidacja zgłosiła błąd.

## Odświeżenie po POST

Odświeżenie odpowiedzi na POST może ponowić zapis. Wzorzec **POST → Redirect → GET** kieruje użytkownika po sukcesie na zwykły adres strony. W PHP wykonuje się to przez header("Location: ...", true, 303), a następnie exit. Przekierowanie musi nastąpić przed wysłaniem HTML.

**Zapamiętaj:** Komunikat „Dodano” pokazuj dopiero po udanym wykonaniu instrukcji. Samo przygotowanie zapytania nie zapisuje danych. Po uruchomieniu tego przykładu stan bazy będzie już inny niż początkowe trzy rekordy.

# 27. UPDATE i DELETE

Poniższe operacje zmieniają dane w bazie ćwiczeniowej. W prawdziwym formularzu identyfikator trzeba zweryfikować jako dodatnią liczbę całkowitą. Przed operacją sprawdza się też prawo użytkownika do wskazanego rekordu.

```
require_once __DIR__ . "/db.php";
$id = 1;
$nowyStan = 6;

$stmt = mysqli_prepare($db,
    "UPDATE produkty SET stan = ? WHERE id = ?");
mysqli_stmt_bind_param($stmt, "ii", $nowyStan, $id);
mysqli_stmt_execute($stmt);
echo "Zmieniono: " . mysqli_stmt_affected_rows($stmt);
mysqli_stmt_close($stmt);
```

Pierwszy parametr to nowy stan, drugi to identyfikator. WHERE ogranicza zmianę do wskazanego produktu. W typowej konfiguracji wynik affected_rows równy 0 może oznaczać brak rekordu **albo** wpisanie wartości identycznej z dotychczasową. Nie traktuj zera automatycznie jako błędu.

```
// Dalsza część tego samego skryptu; $db nadal otwarte.
$idDoUsuniecia = 3;
$stmt = mysqli_prepare($db,
    "DELETE FROM produkty WHERE id = ?");
mysqli_stmt_bind_param($stmt, "i", $idDoUsuniecia);
mysqli_stmt_execute($stmt);
echo "Usunięto: " . mysqli_stmt_affected_rows($stmt);
mysqli_stmt_close($stmt);
mysqli_close($db);
```

DELETE usuwa cały rekord, a nie tylko jedną kolumnę. Do zmiany pojedynczego pola służy UPDATE. Gdy nie istnieje rekord id = 3, poprawnie wykonane DELETE zwykle obejmie 0 rekordów.

## Kiedy jedna operacja obejmuje kilka zapytań?

Przy zależnych zmianach można użyć transakcji: **mysqli_begin_transaction($db)**, potem zapytania, na końcu **mysqli_commit($db)**. W obsłudze błędu **mysqli_rollback($db)** wycofuje niezatwierdzone zmiany. Wymaga to tabel obsługujących transakcje, np. InnoDB. Nie każda instrukcja SQL podlega takiemu wycofaniu.

**Zapamiętaj:** W aplikacji internetowej nie realizuj usuwania przez zwykły link GET. Użyj POST, potwierdzenia zamiaru, kontroli uprawnień oraz ochrony CSRF. Ten fragment pokazuje mechanikę SQL i nie stanowi kompletnego panelu administracyjnego.

# 28. Projekt - przygotowanie

**Cel:** strona katalog.php pokazuje produkty z bazy oraz pozwala dodać nowy produkt. Waliduje formularz, zapisuje dane przez parametry i po sukcesie przekierowuje do listy. To lokalna aplikacja ćwiczeniowa bez logowania.

| Element | Co przygotować |
|---|---|
| Baza | inf03_nauka i tabela produkty ze strony 20. |
| db.php | Pełny plik połączenia ze strony 22. |
| katalog.php | Część PHP ze strony 31, a bezpośrednio po niej HTML ze strony 32. Obie części tworzą jeden plik. |
| Uruchomienie | http://localhost/nauka/katalog.php, jeśli pliki umieszczono w katalogu nauka serwera. |

## Dokładne wymagania formularza

**nazwa** - tekst od 1 do 80 znaków po usunięciu skrajnych białych znaków. **cena** - kwota od 0 do 999999,99 z maksymalnie dwoma cyframi po separatorze. **stan** - liczba całkowita od 0 do 100000. Nazwy pól HTML i klucze w PHP muszą być identyczne.

## Dlaczego cena jest w tym projekcie tekstem?

Baza przechowuje cenę jako DECIMAL(8,2). Projekt sprawdza format kwoty i przekazuje ją jako parametr **s**, zachowując jej zapis dziesiętny bez pośredniego liczenia na float. SQL przekształca sprawdzony napis, np. "12.50", do DECIMAL. Do prezentacji użyjemy number_format().

```
$wzorCeny = '/^\d{1,6}(\.\d{1,2})?$/D';
```

To wzorzec wyrażenia regularnego: **^** oznacza początek, **\d{1,6}** od jednej do sześciu cyfr, **\.** dosłowną kropkę, a **(...)?** opcjonalną część dziesiętną. **$** oznacza koniec, a modyfikator D wymusza ścisły koniec napisu. Format odrzuca minus, zapis wykładniczy i więcej niż dwa miejsca po kropce.

**Wymagane rozszerzenia:** mysqli oraz mbstring; to drugie jest użyte do sprawdzania długości nazwy w znakach UTF-8. Plik katalog.php musi zaczynać się od <?php, bez wcześniejszego HTML. Zapisz go bez BOM.

**Zapamiętaj:** Projekt jest przeznaczony do lokalnej nauki. Przed publicznym udostępnieniem funkcji zapisu należałoby dodać uwierzytelnienie, uprawnienia i ochronę CSRF. Na egzaminie dostosuj nazwy plików, bazy i komunikaty do arkusza.

# 29. Projekt - odbiór i zapis

**katalog.php, część 1/2.** Przepisz od początku pliku. Kod HTML z następnej strony dopisz bezpośrednio po końcowym ?>.

```
<?php
require_once __DIR__ . "/db.php";
function e(string $s): string {
    return htmlspecialchars($s, ENT_QUOTES, "UTF-8");
}
$blad = "";
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $nazwa = $_POST["nazwa"] ?? "";
    $cena = $_POST["cena"] ?? "";
    $stan = filter_input(INPUT_POST, "stan",
        FILTER_VALIDATE_INT,
        ["options" => ["min_range" => 0,
                       "max_range" => 100000]]);
    if (!is_string($nazwa) || !is_string($cena)) {
        $blad = "Niepoprawny format danych.";
    } else {
        $nazwa = trim($nazwa);
        $cena = str_replace(",", ".", trim($cena));
        $wzorCeny = '/^\d{1,6}(\.\d{1,2})?$/D';
        if ($nazwa === "" || mb_strlen($nazwa, "UTF-8") > 80 ||
            preg_match($wzorCeny, $cena) !== 1 ||
            $stan === false || $stan === null) {
            $blad = "Sprawdź nazwę, cenę i stan.";
        } else {
            try {
                $stmt = mysqli_prepare($db,
                    "INSERT INTO produkty (nazwa, cena, stan)
                     VALUES (?, ?, ?)");
                mysqli_stmt_bind_param($stmt, "ssi",
                    $nazwa, $cena, $stan);
                mysqli_stmt_execute($stmt);
                mysqli_stmt_close($stmt);
                mysqli_close($db);
                header("Location: katalog.php?ok=1", true, 303);
                exit;
            } catch (mysqli_sql_exception $e) {
                error_log($e->getMessage());
                $blad = "Nie udało się zapisać produktu.";
            }
        }
    }
}
$wynik = mysqli_query($db,
    "SELECT id, nazwa, cena, stan FROM produkty ORDER BY id");
?>
```

# 30. Projekt - formularz i lista

**katalog.php, część 2/2.** Ciąg dalszy poprzedniej strony. Formularz wysyła dane do tego samego skryptu.

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>Katalog produktów</title>
</head>
<body>
<h1>Katalog produktów</h1>
<?php if ($blad !== ""): ?>
    <p><?= e($blad) ?></p>
<?php endif; ?>
<?php if (($_GET["ok"] ?? "") === "1"): ?>
    <p>Produkt dodany.</p>
<?php endif; ?>
<form method="post" action="katalog.php">
    <label>Nazwa:
        <input name="nazwa" maxlength="80" required>
    </label><br>
    <label>Cena:
        <input name="cena" type="number" min="0"
               max="999999.99" step="0.01" required>
    </label><br>
    <label>Stan:
        <input name="stan" type="number" min="0"
               max="100000" step="1" required>
    </label><br>
    <button type="submit">Dodaj</button>
</form>
<table>
    <tr><th>ID</th><th>Nazwa</th><th>Cena</th><th>Stan</th></tr>
    <?php while ($w = mysqli_fetch_assoc($wynik)): ?>
        <tr>
            <td><?= (int) $w["id"] ?></td>
            <td><?= e($w["nazwa"]) ?></td>
            <td><?= number_format(
                (float) $w["cena"], 2, ",", " "
            ) ?> zł</td>
            <td><?= (int) $w["stan"] ?></td>
        </tr>
    <?php endwhile; ?>
</table>
<?php
mysqli_free_result($wynik);
mysqli_close($db);
?>
</body>
</html>
```

# 31. Projekt - analiza i testy

## Jak prześledzić działanie aplikacji?

**Wejście GET:** dołączenie db.php → zdefiniowanie e() → pominięcie obsługi POST → pobranie listy → wygenerowanie HTML.
**Błędny POST:** pobranie danych → wykrycie błędu → ustawienie $blad → pominięcie INSERT → wyświetlenie komunikatu i listy.
**Poprawny POST:** walidacja → przygotowanie INSERT → związanie parametrów → wykonanie → przekierowanie 303 → nowe żądanie GET z listą.

**preg_match()** zwraca 1 przy dopasowaniu, 0 przy jego braku, a false przy błędzie wzorca. Dlatego porównujemy wynik ściśle z 1. Dla stanu 0 walidacja jest poprawna - kod porównuje z false i null, a nie używa empty().

| Próba | Oczekiwany rezultat |
|---|---|
| Pierwsze otwarcie strony | Lista istniejących produktów; brak komunikatu błędu. |
| Nazwa: Teczka; cena: 8.50; stan: 10 | Jeden nowy rekord i przekierowanie na katalog.php?ok=1. |
| Nazwa składająca się ze spacji | Komunikat błędu; brak INSERT. |
| Cena: -1 albo 12.345 | Walidacja HTML blokuje zwykłe wysłanie; PHP również odrzuci dane po ominięciu przeglądarki. |
| Stan: 0 | Poprawny zapis; zero jest dozwolone. |
| Stan: 2.5 albo 100001 | Odrzucenie danych. |
| Nazwa: Notes | Nazwa widoczna jako tekst wraz ze znacznikami, bez ich wykonania. |
| Odświeżenie po sukcesie | Żądanie GET; brak ponowienia tego samego POST. |
| Celowe błędne dane połączenia | Krótki komunikat o niedostępności bazy; szczegóły w logu. |

## Świadome uproszczenia

Po błędnym POST formularz jest pusty; zachowanie wartości można dodać według strony 19. Parametr ok=1 steruje jedynie komunikatem i można go dopisać ręcznie - nie jest dowodem zapisu. Odczyt SELECT nie ma osobnego catch; błąd schematu diagnozuje się w logu środowiska. W rozbudowanej aplikacji objęlibyśmy obsługą błędów również odczyt.

**Zapamiętaj:** Przekierowanie ogranicza ponowienie po odświeżeniu, ale nie gwarantuje ochrony przed każdym podwójnym wysłaniem. Formularz i lista to działający wzorzec dydaktyczny, nie gotowy system sprzedażowy.

# 32. Sesje i ciasteczka

**Sesja** pozwala powiązać kolejne żądania z danymi przechowywanymi po stronie serwera. Przeglądarka zwykle przekazuje identyfikator sesji w ciasteczku. Samo HTTP nie pamięta wcześniejszych wartości zmiennych PHP.

```
<?php
session_start(); // przed wysłaniem HTML
$_SESSION["licznik"] = ($_SESSION["licznik"] ?? 0) + 1;
echo "Liczba odwiedzin: " . (int) $_SESSION["licznik"];
?>
```

**session_start()** rozpoczyna lub wznawia sesję. **$_SESSION** przechowuje dane dostępne przy kolejnych żądaniach z tą samą sesją. Zwykła zmienna $licznik nie zachowałaby wartości po przeładowaniu strony.

## Ciasteczko po stronie przeglądarki

```
<?php
setcookie("motyw", "jasny", [
    "expires" => time() + 3600,
    "path" => "/",
    "httponly" => true,
    "samesite" => "Lax"
]);
$motyw = $_COOKIE["motyw"] ?? "brak";
?>
```

**setcookie()** wysyła nagłówek proszący przeglądarkę o zapis ciasteczka. Nie aktualizuje natychmiast $_COOKIE: nowa wartość zwykle wróci w następnym żądaniu. expires określa czas wygaśnięcia, path zakres ścieżek, httponly ogranicza dostęp JavaScript, a samesite wpływa na wysyłanie między witrynami. Dla HTTPS ustaw również secure na true.

## Hasła - tylko skróty przeznaczone do haseł

```
$hash = password_hash("przykladowe-haslo", PASSWORD_DEFAULT);
$poprawne = password_verify("przykladowe-haslo", $hash);
```

W bazie zapisuj hash, a nie jawne hasło. password_verify() porównuje podane hasło z zapisanym hashem. Dla kolumny hash rozsądnym wyborem jest VARCHAR(255). Nie porównuj dwóch nowych wywołań password_hash(), bo losowa sól powoduje różne wyniki. MD5 i SHA-1 nie są odpowiednimi funkcjami do przechowywania haseł.

**Zapamiętaj:** session_start(), setcookie() oraz header() wywołuj przed HTML, echo i przypadkowymi pustymi znakami. Po udanym logowaniu odnawia się identyfikator sesji; samo zapisanie imienia w sesji nie jest uwierzytelnieniem.

# 33. Pliki i alternatywny zapis

## Odczyt i zapis pliku tekstowego

```
$plik = __DIR__ . "/notatka.txt";
$zapis = file_put_contents($plik, "Pierwsza linia\n");
if ($zapis === false) {
    exit("Nie udało się zapisać pliku.");
}
file_put_contents($plik, "Druga linia\n", FILE_APPEND | LOCK_EX);
$tekst = file_get_contents($plik);
if ($tekst !== false) {
    echo nl2br(htmlspecialchars($tekst, ENT_QUOTES, "UTF-8"));
}
```

**file_put_contents()** bez FILE_APPEND nadpisuje zawartość; zwraca liczbę zapisanych bajtów albo false. **FILE_APPEND** dopisuje, **LOCK_EX** żąda blokady wyłącznej podczas zapisu. **file_get_contents()** czyta cały plik. **nl2br()** dodaje znaczniki końca linii HTML. Serwer musi mieć odpowiednie uprawnienia do pliku / katalogu.

| Funkcja | Do czego służy |
|---|---|
| fopen($plik, "r") | Otwiera do odczytu; "w" zeruje / tworzy plik, "a" otwiera do dopisywania. |
| fgets($uchwyt) | Pobiera linię z otwartego pliku. |
| fwrite($uchwyt, $tekst) | Zapisuje tekst do otwartego pliku. |
| fclose($uchwyt) | Zamyka uchwyt. |
| file_exists($plik) | Sprawdza istnienie ścieżki. |

## MySQLi proceduralne a obiektowe

```
// Te pary ilustrują dwa style tej samej operacji:
$wynik = mysqli_query($db, $sql);
$wynik = $db->query($sql);

$w = mysqli_fetch_assoc($wynik);
$w = $wynik->fetch_assoc();
```

To **alternatywy**, a nie instrukcje do wykonania kolejno - podwójny fetch pobrałby dwa różne rekordy. Wybierz spójny styl. Spotkasz też PDO: inne API z metodami prepare() i execute(). Nie łącz funkcji mysqli_* z obiektem PDO.

**Zapamiętaj:** Przesyłanie plików wymaga formularza POST z enctype="multipart/form-data", pola type="file" oraz $_FILES. Trzeba sprawdzić błędy, rozmiar i rzeczywisty typ pliku. Ten dodatek nie stanowi kompletnej implementacji uploadu.

# 34. Błędy i kontrola przed oddaniem

| Objaw / komunikat | Co sprawdzić |
|---|---|
| Parse error / unexpected token | Średniki, cudzysłowy i domknięcie nawiasów; także poprzednią linię. |
| Undefined variable | Czy zmienna została utworzona wcześniej i w odpowiednim zasięgu? |
| Undefined array key | Czy pole ma name? Czy metoda formularza jest zgodna? Użyj ?? lub sprawdzenia istnienia. |
| Array to string conversion | Czy zamiast tekstu otrzymano tablicę? Nie przekazuj tablicy do echo. |
| Access denied | Użytkownika, hasło i uprawnienia do bazy. |
| Unknown database / table | Nazwę bazy, import tabel oraz wielkość liter w nazwach. |
| Unknown column | Nazwy kolumn i aliasy w zapytaniu SQL. |
| Headers already sent | Czy przed header(), setcookie() lub session_start() wysłano HTML, spacje lub BOM? |
| Biała strona / błąd 500 | Log błędów PHP i serwera; poprawność składni i dostępność rozszerzeń. |
| Brak rekordów bez błędu | Warunek WHERE, parametry, stan bazy oraz num_rows(). |
| Niepoprawne polskie znaki | UTF-8 pliku, meta charset, kodowanie bazy i mysqli_set_charset(). |

## Lokalna diagnostyka

```
error_reporting(E_ALL);
ini_set("display_errors", "1");
// Pomocniczo: var_dump($zmienna);
```

Używaj tego w środowisku nauki, nie do publicznego wyświetlania szczegółów błędów. Błąd składni w tym samym pliku może uniemożliwić wykonanie powyższych ustawień; wtedy sprawdź log / konfigurację. Przy trybie STRICT błędy MySQLi przechwytuj jako wyjątki. mysqli_error($db) jest istotne zwłaszcza w kodzie bez wyjątków.

**Zapamiętaj:** Przed oddaniem: właściwe nazwy plików i pól; poprawny localhost; wykonany import; zgodne SQL; wszystkie rekordy widoczne; brak zapisu przy błędzie; zamknięcie połączenia; brak var_dump(); wymagane w arkuszu zrzuty i pliki zapisane. Nie zakładaj, że poprawny wygląd dowodzi poprawnego działania.

# 35. Ćwiczenia sprawdzające

## 1. Przewidź wynik

```
$x = 2;
for ($i = 1; $i <= 3; $i++) {
    $x += $i;
}
echo $x;
```

Zapisz wartość $x po każdym obiegu. Następnie sprawdź wynik przez uruchomienie kodu.

## 2. Znajdź przyczynę braku danych

```
<form method="post">
    <input id="imie">
    <button type="submit">Wyślij</button>
</form>
```

PHP oczekuje $_POST["imie"]. Co trzeba dopisać do pola i dlaczego samo id nie wystarczy?

## 3. Zero to też wartość

Formularz przesyła stan magazynowy "0". Program używa if (empty($_POST["stan"])) i zgłasza brak danych. Wyjaśnij błąd. Zaproponuj walidację liczby całkowitej od 0 do 100000.

## 4. Lista produktów

Napisz skrypt wyświetlający nazwy produktów ze stanem większym od 5, rosnąco według nazwy. Zastosuj mysqli_fetch_assoc(), listę ul/li i bezpieczne wyświetlanie. Sprawdź wynik na początkowych danych ze strony 20.

## 5. Zapytanie z parametrem

Formularz GET przesyła id produktu. Zweryfikuj dodatnią liczbę całkowitą i przygotuj zapytanie SELECT nazwa, cena FROM produkty WHERE id = ?. Wybierz właściwą literę typu parametru. Jak rozpoznasz brak produktu?

## 6. Rozbuduj projekt

Dodaj kolumnę „Wartość zapasu” obliczaną jako cena × stan oraz sumę wartości całego magazynu. Dla ćwiczenia możesz użyć float i number_format(). Wynik sprawdź dla trzech początkowych produktów. Wskaż miejsce ustawienia sumy na 0 i miejsce dodawania kolejnych wartości.

**Zapamiętaj:** Najpierw wykonaj zadania samodzielnie. Odpowiedzi znajdują się na następnej stronie. Jeżeli wcześniej zmieniano bazę, wyniki liczbowe mogą różnić się od wartości początkowych.

# 36. Odpowiedzi i omówienie

**1.** Początkowo $x = 2. Po kolejnych obiegach: 3, 5, 8. Wynik echo: **8**.
**2.** Dodaj **name="imie"**. Atrybut name wyznacza klucz wysyłany do PHP; id identyfikuje element w dokumencie.
**3.** empty("0") daje true. Użyj filter_input z FILTER_VALIDATE_INT i opcjami min_range = 0, max_range = 100000; błąd rozpoznawaj przez === false lub === null. Wzorzec jest w projekcie.

## 4. Lista - przykładowe rozwiązanie

```
require_once __DIR__ . "/db.php";
$wynik = mysqli_query($db,
    "SELECT nazwa FROM produkty WHERE stan > 5
     ORDER BY nazwa");
echo "<ul>";
while ($w = mysqli_fetch_assoc($wynik)) {
    echo "<li>" . htmlspecialchars(
        $w["nazwa"], ENT_QUOTES, "UTF-8"
    ) . "</li>";
}
echo "</ul>";
mysqli_free_result($wynik);
mysqli_close($db);
```

Na początkowych danych: **Długopis, Notes**. Plecak ma stan 3, więc nie spełnia warunku.

## 5. Parametr identyfikatora

```
$id = filter_input(INPUT_GET, "id", FILTER_VALIDATE_INT,
    ["options" => ["min_range" => 1]]);
if ($id === false || $id === null) {
    exit("Niepoprawne ID.");
}
require_once __DIR__ . "/db.php";
$stmt = mysqli_prepare($db,
    "SELECT nazwa, cena FROM produkty WHERE id = ?");
mysqli_stmt_bind_param($stmt, "i", $id);
mysqli_stmt_execute($stmt);
mysqli_stmt_bind_result($stmt, $nazwa, $cena);
$znaleziono = mysqli_stmt_fetch($stmt);
if ($znaleziono === null) { echo "Brak produktu."; }
mysqli_stmt_close($stmt);
mysqli_close($db);
```

**6.** Notes: 100,00 zł; Długopis: 80,00 zł; Plecak: 299,70 zł. Razem **479,70 zł**. Przed pętlą: $suma = 0; w pętli: $wartosc = (float) $w["cena"] * (int) $w["stan"]; następnie $suma += $wartosc;. Podsumowanie wyświetl po pętli.

# 37. Ściąga - PHP i formularze

| Element | Zapamiętaj |
|---|---|
| echo / print | Wyświetlanie. echo nie wymaga nawiasów. print zwraca 1. |
| $x = ...; | Przypisanie wartości; nie porównanie. |
| . / .= | Łączenie tekstu / dopisywanie do zmiennej. |
| == / === | Równość po możliwej konwersji / zgodność wartości i typu. |
| if / elseif / else | Wybór gałęzi na podstawie warunków. |
| for / while / foreach | Licznik / warunek / przejście po tablicy. |
| isset() / empty() | Istnieje i nie null / wartość uznawana za pustą. |
| ?? | Wartość zastępcza dla braku klucza lub null. |
| $_GET / $_POST | Dane z adresu / dane zwykłego formularza POST. |
| $_SERVER["REQUEST_METHOD"] | Metoda żądania, np. "POST". |
| is_string() / is_array() | Sprawdzenie typu wartości wejściowej. |
| filter_input() / filter_var() | Filtrowanie wejścia / przekazanej wartości. Jawnie wybierz filtr walidacyjny. |
| trim() | Usunięcie skrajnych białych znaków. |
| htmlspecialchars() | Przygotowanie tekstu do HTML. |
| count() / strlen() / mb_strlen() | Elementy tablicy / bajty tekstu / znaki tekstu. |
| strpos() | Pozycja od 0 lub false; porównuj z !== false. |
| explode() / implode() | Podział napisu / połączenie elementów tablicy. |
| str_replace() | Zastąpienie fragmentu tekstu. |
| round() / number_format() | Zaokrąglenie liczby / tekst sformatowanej liczby. |
| date() / time() | Format daty / znacznik czasu Unix. |
| require_once | Dołączenie wymaganego pliku najwyżej raz. |
| header() / exit | Nagłówek odpowiedzi / zakończenie skryptu. |
| var_dump() | Podgląd typu i wartości podczas diagnostyki. |

# 38. Ściąga - MySQLi

| Funkcja | Wynik / rola |
|---|---|
| mysqli_connect(host, user, pass, db) | Połączenie mysqli przy sukcesie; false bez wyjątków przy niepowodzeniu. W STRICT błąd rzuca wyjątek. |
| mysqli_set_charset($db, "utf8mb4") | Ustawia kodowanie komunikacji z bazą. |
| mysqli_select_db($db, $nazwa) | Zmienia / wybiera domyślną bazę dla połączenia. |
| mysqli_query($db, $sql) | Dla SELECT: mysqli_result. Dla zwykłych zmian: true przy sukcesie. Bez wyjątków możliwe false. |
| mysqli_num_rows($wynik) | Liczba wierszy w zestawie wynikowym. |
| mysqli_num_fields($wynik) | Liczba kolumn zestawu wynikowego. |
| mysqli_fetch_assoc($wynik) | Następny rekord jako tablica z nazwami kolumn. |
| mysqli_fetch_row($wynik) | Następny rekord jako tablica z indeksami liczbowymi. |
| mysqli_fetch_array($wynik) | Domyślnie tablica z nazwami i indeksami (MYSQLI_BOTH). |
| mysqli_free_result($wynik) | Zwalnia pamięć wyniku. |
| mysqli_affected_rows($db) | Liczba rekordów objętych poprzednią operacją; interpretacja zależy od rodzaju zapytania. |
| mysqli_insert_id($db) | Ostatnio wygenerowane AUTO_INCREMENT w tym połączeniu. |
| mysqli_error($db) | Tekst ostatniego błędu połączenia; przy STRICT zwykle obsługujesz wyjątek. |
| mysqli_prepare($db, $sql) | Przygotowuje instrukcję z parametrami ?. |
| mysqli_stmt_bind_param(...) | Wiąże zmienne wejściowe; i, d, s, b. |
| mysqli_stmt_execute($stmt) | Wykonuje przygotowaną instrukcję. |
| mysqli_stmt_bind_result(...) | Wiąże kolumny wyniku ze zmiennymi. |
| mysqli_stmt_fetch($stmt) | Pobiera następny wiersz do związanych zmiennych. |
| mysqli_stmt_affected_rows($stmt) | Liczba rekordów objętych operacją instrukcji. |
| mysqli_stmt_close($stmt) | Zamyka przygotowaną instrukcję. |
| mysqli_close($db) | Zamyka połączenie po ostatnim zapytaniu. |

# 39. Źródła i dalsza nauka

Materiał łączy autorskie przykłady dydaktyczne z dokumentacją PHP oraz analizą reprezentatywnego arkusza CKE. Nie jest oficjalnym wykazem wszystkich funkcji obowiązujących na każdym egzaminie. Konkretne wymagania określa treść otrzymanego zadania.

## Arkusz egzaminacyjny

CKE: INF.03-07-24.06-SG, część praktyczna - kopia arkusza w publicznym archiwum. Przykład obejmuje zapytania SQL, generowanie listy i tabeli przez PHP oraz zakończenie połączenia z bazą. To uzasadnia nacisk tego podręcznika na przepływ: zapytanie → rekordy → HTML. Arkusz nie wyczerpuje całego zakresu kwalifikacji.

## Oficjalna dokumentacja PHP - źródła techniczne

Składnia i znaczniki PHP

Obsługa formularzy

htmlspecialchars() i konteksty HTML

filter_input() - wartości zwracane i filtry

MySQLi: przewodnik wprowadzający

MySQLi: wykonywanie zapytań

MySQLi: zapytania przygotowane

## Dalsze tematy do samodzielnego sprawdzenia

W podręczniku PHP wyszukaj nazwy: mysqli_fetch_array, mysqli_report, password_hash, session_start, setcookie, file_put_contents. Czytając dokumentację, zwracaj uwagę na argumenty, zwracany typ, wymagane rozszerzenia i różnice pomiędzy wersjami PHP.

## Jak dopasować wiedzę do polecenia egzaminacyjnego?

Najpierw wypisz: nazwę pliku PHP, bazę i tabelę, parametry formularza, wymagane kolumny, warunki zapytań oraz sposób prezentacji wyników. Zaznacz, które działania mają zachodzić przy otwarciu strony, a które dopiero po wysłaniu formularza. Zbuduj skrypt etapami i sprawdzaj każdy rezultat w przeglądarce oraz w bazie.

**Zapamiętaj:** Nie ucz się wyłącznie nazw funkcji. Umiej wskazać, co w danej chwili przechowują $db, $sql, $wynik, $wiersz i zmienne z formularza. To pozwala dostosować znany schemat do nowego zadania.

Linki są klikalne. Materiały sprawdzono 16.09.2026. Przykłady dotyczą PHP 8.x; ustawienia środowiska szkolnego mogą się różnić.