Програмування та підтримка вебзастосунків

Навчально-методичний посібник

Дорош А.Б., Іліка С.А.

Чернівці
Чернівецький національний університет
імені Юрія Федьковича
2025

УДК 004(075.8)
П-784

Друкується за ухвалою Вченої Ради факультету математики та інформатики Чернівецького національного університету імені Юрія Федьковича, протокол №10 від 23.04.2025 р.

Рецензенти:
І.Д.Скутар, кандидат фіз.-мат. наук, асистент кафедри прикладної математики та інформаційних технологій Факультету математики та інформатики Чернівецького національного університету імені Юрія Федьковича
В.В.Лазорик, кандидат фіз.-мат. наук, доцент кафедри математичних проблем управління і кібернетики Навчально-наукового інституту фізико-технічних та комп'ютерних наук Чернівецького національного університету імені Юрія Федьковича

П-784 Програмування та підтримка вебзастосунків. Навчально-методичний посібник / Укл.: Дорош А.Б., Іліка С.А. Чернівці: Чернівецький нац. ун-т, 2025. 120 с.

Посібник містить теоретичний матеріал, приклади розв'язування типових завдань, завдання для лабораторних занять із курсу бекенд-розробки. Для здобувачів вищої освіти першого (бакалаврського) рівня спеціальностей 122 «Комп'ютерні науки», 124 «Системний аналіз» та 014.09 «Середня освіта (Інформатика)».

Зміст

Розділ 1

PHP – важливий гравець у вебтехнологіях

1.1 Що таке PHP? Звідки походить?

PHP – це мова програмування, створена спеціально для розробки вебдодатків, тобто скриптів, які виконуються на вебсервері.

Абревіатура PHP розшифровується як *Hypertext Preprocessor* (препроцесор гіпертексту).

Свій початок мова PHP бере з продукту під назвою *PHP/FI*, який в 1995 році створив датський програміст *Расмус Лердорф* (Rasmus Lerdorf). Спочатку це був набір Perl-скриптів для відстеження відвідувань його онлайн-резюме. Вебтехнології лише починали свій розвиток, і спеціалізованих засобів для вирішення подібних завдань ще не існувало. Лердорф отримував багато запитань про те, як він зробив підрахунок статистики на своїй сторінці, і він почав безкоштовно поширювати свій інструментарій, назвавши його *Personal Homepages Tools*, тобто, *інструменти для персональних домашніх сторінок.*

Рис. 1.1: Расмус Лердорф

Дуже скоро функціоналу стало замало, і Расмус пише нову, значно більш обширну версію мовою C (PHP/FI – Forms Interpreter). З'явилася можливість працювати з базами даних. Користувачі могли створювати повноцінні вебзастосунки.

Расмус Лердорф виклав вихідний код PHP/FI у загальний доступ, задля виправлення помилок і доповнення. У 1997 році з'являється PHP/FI 2.0 (також написана мовою C).

У той період нею було написано 1% усіх сайтів Інтернету. PHP/FI 2.0 досі залишалась крупним проєктом однієї людини.

PHP 3.0 була першою версією, що нагадує теперішню PHP. У ній з'явилася підтримка об'єктно-орієнтованого

програмування (ООП).

У 1997 році *Енді Гатменс* (Andi Gutmans) та *Зів Сураскі* (Zeev Suraski) переписали код з нуля. Вони вважали код PHP/FI 2.0 непридатним для розробки сайту електронної комерції, над яким працювали для університетського проєкту. Енді, Зів та Расмус об'єдналися і разом почали працювати над PHP 3.0. Розробку PHP/FI було припинено.

Абсолютно нова мова програмування отримала нову назву. Абревіатура PHP тепер означала не Personal Homepage, а *Hypertext Preprocessor*. У 1998 році розробники почали переробляти код PHP задля збільшення продуктивності на складних проєктах та покращення підтримки модульності.

PHP 4 з'явилась у 2000 році.

У 2004 році вийшла PHP 5 із покращеними можливостями ООП та швидкодією.

Певний час велася розробка PHP 6, яка мала підтримувати кодування символів Unicode на рівні ядра. Однак через брак необхідних розробників проєкт було покинуто.

Проте деякі корисні можливості PHP 6 були втілені у PHP 5.3 (простори назв) і PHP 5.4 (трейти).

Хоча PHP 6 так і не було завершено, назва вже була зайнята. І навіть були надруковані книги про PHP 6.

Тому шляхом Інтернет-голосування серед розробників було вирішено наступний реліз назвати PHP 7. Він вийшов у 2014 році. Швидкодія PHP 7 була вдвічі швидшою, порівняно з PHP 5. Що було додано:

- *Покращена продуктивність* – PHP 7 значно швидший за PHP 5 завдяки новому механізму виконання під назвою **Zend Engine 3.0**, що дозволяє збільшити продуктивність до двох разів у більшості вебзастосунків.

- *Скалярні типи аргументів та повернення значень* – PHP 7 дозволяє вказувати типи даних аргументів та значень, що повертаються функціями:

```
function add(int $a, int $b): int {
    return $a + $b;
}
```

- *Оператор об'єднання з null (??)* – новий оператор для встановлення значення за замовчуванням:

```
$username = $_GET['user'] ?? 'гість';
```

- *Оператор космічного корабля (<=>)* – використовується для порівняння:

```
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1
```

- *Анонімні класи (anonymous classes)* – можливість створювати класи без імені:

```
$obj = new class {
    public function sayHello() {
        return "Привіт!";
    }
};
```

- *Покращене оброблення помилок* – запроваджено новий клас `EngineException` і загальну систему `Throwable`, що дозволяє краще обробляти помилки.

- *Видалення застарілих функцій та конструкцій* – багато застарілих елементів було повністю видалено, включаючи `mysql_`, `ereg()`, `call_user_method()` та інші.

- *Покращена підтримка Unicode* та інтернаціоналізації, зокрема в бібліотеках.

- *Генератори з ключами (Generator delegation)* – генератори можуть делегувати іншим генераторам через `yield from`.

- *Return type declarations* – функції можуть явно вказувати тип значення, що повертається.

У 2020 році було випущено PHP 8 зі значними покращеннями продуктивності, безпеки та зручності розробки. Основні нововведення:

- *JIT-компіляція* (Just-In-Time compilation) – значне прискорення виконання коду за рахунок динамічної компіляції байткоду в машинний код.

- Удосконалена типізація – з'явилися *Union Types* (об'єднані типи), що дозволяють вказувати кілька допустимих типів для змінних і параметрів функцій.

- Новий синтаксис атрибутів (анотацій) – дозволяє використовувати метадані у вигляді нативного синтаксису PHP, замість традиційних DocBlock-коментарів.

- *Match Expression* – покращена альтернатива `switch`, яка повертає значення та не потребує `break`.

- *Іменовані аргументи функцій* (Named Arguments) – можливість передавати аргументи у функції за назвою, що підвищує читабельність і зменшує кількість коду.

- *Nullsafe Operator* (`?->`) – спрощений синтаксис для перевірки значення на `null` при виклику методів або зверненні до властивостей.

- *Weak Maps* – покращена робота з об'єктами в кешах і механізмах збору сміття.

- Вбудована валідація та покращена обробка помилок – більше детальних винятків і кращий контроль над помилками.

Версія PHP 8 зробила мову ще більш продуктивною, гнучкою та зручною для розробників, спрощуючи написання чистого та ефективного коду.

На момент написання, актуальна версія PHP – 8.4.

1.2 Особливості PHP

Мова PHP є безкоштовною для використання. Поширюється з відкритим вихідним кодом.

PHP-скрипти можна запускати на різних платформах (Windows, Linux, macOS і т.д.). У PHP використовується інтерпретатор, а не компілятор.

Компільовані мови:

- C
- C++
- Java

- C#
- Pascal
- Go

Переваги: код компілюється тільки 1 раз, і всі подальші рази запускається одразу двійковий код.

Недоліки: після кожного редагування коду треба постійно компілювати проєкт заново.

Інтерпретовані мови:

- PHP
- JavaScript
- Python
- Perl
- Ruby
- Wolfram Mathematica

Переваги: не потрібно після кожної зміни коду компілювати все заново. Достатньо зберегти файл, й інтерпретатор наступного разу запустить змінений код.

Недоліки: при кожному запуску скрипта витрачається час на перетворення кожної команди у двійковий код.

Хоча насправді "за кадром"код PHP перетворюється в проміжний байт-код, який виконується інтерпретатором. Байт-код виконується швидше, ніж вихідний код.

PHP-файли часто називають *скриптами*.

Синтаксис PHP бере початок із мов C, Java і Perl.

Мова PHP дуже проста для вивчення, але разом із тим може задовольнити потреби дуже складних проєктів.

Мовою PHP написані такі відомі вебсайти:

- Facebook
- Wikipedia
- Flickr
- Yahoo!
- Mailchimp
- Wordpress

Також існує чимало фреймворків та платформ, створених мовою PHP: WordPress, Joomla, Drupal, Magento, phpBB, MediaWiki, Laravel, Yii.

Згідно статистики W3Techs.com, 79% усіх вебсайтів світу написано з використанням PHP.

Мова PHP постійно розвивається. Вона ще довго залишатиметься однією з найпопулярніших мов для веброзробки.

1.3 Встановлення вебсервера

PHP-файли (скрипти) мають розширення `.php`. PHP-скрипти виконуються тільки на сервері.

Для їхнього запуску потрібно встановити серверне програмне забезпечення (*вебсервер*). Найпопулярнішими вебсерверами є Apache, nginx, IIS.

Ми будемо розглядати найпопулярніший безкоштовний кросплатформний сервер *Apache* із відкритим вихідним кодом.

Apache на Windows можна встановити окремо, але швидше і зручніше – у складі безкоштовного пакету програм

XAMPP (Apache, MySQL, PHP, Perl). XAMPP на Windows краще встановлювати у корінь диска `D`, а не на диск `C`. Наприклад, у папку `D:\xampp`.

Шлях має складатися тільки з латинських літер і не повинен повинен містити пробілів. Після встановлення і запуску XAMPP відкриється вікно, в якому треба запустити сервер Apache кнопкою Start. PHP-скрипти мають знаходитися в папці `D:\xampp\htdocs`.

У браузері ваш сайт буде доступний за адресою `http://localhost` або `http://127.0.0.1` (тільки з вашого комп'ютера).

По суті, localhost вказує на папку `D:\xampp\htdocs`. Сайтів на сервері може бути багато. Тому краще для кожного сайту створювати окрему підпапку, наприклад:

```
D:\xampp\htdocs\site1
```

Тоді в браузері сайт буде доступний за адресою `http://localhost/site1` або `http://127.0.0.1/site1` (тільки з вашого комп'ютера). Головну сторінку сайту рекомендується називати `index.php`.

На ОС сімейства Linux сервер Apache можна встановити окремо з офіційних репозиторіїв. Наприклад, на Debian та Ubuntu за допомогою консольної команди `sudo apt install apache2`. За замовчуванням, PHP-скрипти потрібно помістити в директорію `/var/www/html`, хоча це можна змінити в налаштуваннях.

1.4 Призначення PHP

Що відбувається, коли ми відкриваємо сайт у браузері?

HTML дає змогу робити тільки фіксований контент. Наприклад, HTML-код сторінки з поточною датою:

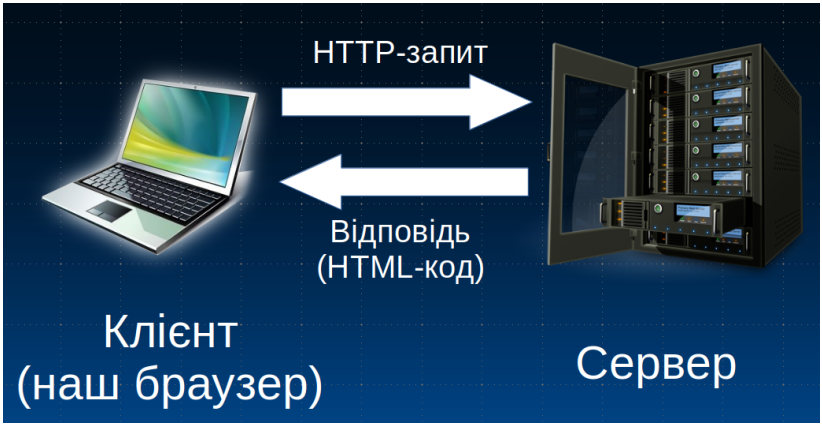

Рис. 1.2: Схема HTTP-запиту

```
<html>
<head>
  <meta charset="UTF-8" />
</head>
<body>
  Сьогодні 17.09.2025
</body>
</html>
```

Але завтра ми побачимо ту саму стару дату. Змінювати її доведеться щодня вручну.

Серверні мови програмування дозволяють генерувати динамічний контент.

HTML-код сторінки із вставкою PHP-коду:

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
  Сьогодні <?php echo date("d.m.Y"); ?>
</body>
```

```
</html>
```

Спочатку сервер порахує і підставить поточну дату, а тоді поверне готовий HTML:

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
  Сьогодні 17.09.2025
</body>
</html>
```

Саме тому PHP означає *препроцесор гіпертексту* (попередня обробка HTML).

За допомогою PHP-коду можна динамічно згенерувати певний HTML-код на сервері, а вже тоді результат повернеться до браузера клієнта.

Звісно, що клієнт у браузері не побачить PHP-коду. Він залишається на сервері. Клієнт побачить лише готовий HTML-код.

Мова PHP має C-подібний синтаксис. Проте, на відміну від мов C та C++, PHP є простішою у вивченні й має низку зручних можливостей.

Як ми вже побачили на прикладі, PHP-код може вставлятися в HTML. Кожен блок PHP-коду починається спеціальним тегом

```
<?php
```

і закінчується іншим тегом

```
?>
```

Є також і скорочений PHP-тег

```
<?
...
?>
```

Але він працює не на всіх серверах. Краще його уникати.

HTML-код за межами PHP-тегів виведеться на екран без змін.

Для виводу на екран усередині PHP-тегів є команда `echo`.

Повернемось до прикладу:

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
  Сьогодні <?php date("d.m.Y"); ?>.
</body>
</html>
```

Результат: `Сьогодні .`

Як бачимо, дата обчислилась сервером, але не вивелась на екран. А тепер із використанням `echo`:

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
  Сьогодні <?php echo date("d.m.Y"); ?>.
</body>
</html>
```

Результат: `Сьогодні 17.09.2025.`

Вставок PHP-коду може бути багато.

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
```

```
  Дата <?php echo date("d.m.Y"); ?>,
  час <?php echo date("H:i:s"); ?>.
</body>
</html>
```

Результат: `Дата 17.09.2025, час 09:30:01.`

Якщо всередині PHP-тегу є тільки єдина команда `echo`

```
<?php echo date("d.m.Y"); ?>
```

то можна використати спеціальний echo-тег:

```
<?= date("d.m.Y"); ?>
```

Отже, є 3 типи PHP-тегів:

- Звичайний `<?php ... ?>`
- Скорочений `<? ... ?>`
- Echo-тег `<?= ... ?>`

У кінці файла закривати PHP-тег не обов'язково.

```
<?php
echo "Hello world";
...
echo "End of file";
```

1.5 Розбиття HTML-коду на окремі файли

Зазвичай, всі сучасні вебсайти мають схожу структуру сторінок:

- Верхня частина сайту (шапка, *header*) – назва сайту, логотип і т.д.

- Меню сайту (ліве, праве, верхнє)
- Вміст сторінки (контент, *content*)
- Нижня частина сайту (підвал, *footer*) – інформація про авторів сайту, контакти тощо.

Типова HTML-структура сторінки:

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
  <div id="header">...</div>
  <div id="left-menu">...</div>
  <div id="content">...</div>
  <div id="footer">...</div>
</body>
</html>
```

Або з використанням тегів HTML5:

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
  <header>...</header>
  <nav>...</nav>
  <main>...</main>
  <footer>...</footer>
</body>
</html>
```

Код кожної частини сторінки прийнято розміщувати в окремому файлі.

```
<html>
  <head><meta charset="UTF-8" /></head>
```

```
<body>
  <div id="header">...</div>
  <!-- ----------------------- -->
  <div id="left-menu">...</div>
  <!-- ----------------------- -->
  <div id="content">...</div>
  <!-- ----------------------- -->
  <div id="footer">...</div>
</body>
</html>
```

Або

```
<html>
  <head><meta charset="UTF-8" /></head>
<body>
  <header>...</header>
  <!-- ----------------------- -->
  <nav>...</nav>
  <!-- ----------------------- -->
  <main>...</main>
  <!-- ----------------------- -->
  <footer>...</footer>
</body>
</html>
```

Для того, щоб вставити HTML-код із файлу, PHP має кілька функцій:

- `include("шлях_до_файлу");`

  відкриває файл і вставляє його вміст у поточний файл; якщо файлу немає, видає попередження (warning) і продовжує виконання далі;

- `require("шлях_до_файлу");`

  те саме, що й `include`; але якщо файлу немає, видає фатальну помилку і припиняє виконання скрипта;

- `include_once("шлях_до_файлу");`

  те саме, що й `include`; але якщо файл із такою назвою вже був підключений, то пропускає його;

- `require_once("шлях_до_файлу");`

  те саме, що й `require`; але якщо файл із такою назвою вже був підключений, то пропускає його.

Якщо вставити один і той же файл кілька разів за допомогою `require_once`

```
require_once("header.php");
require_once("header.php");
```

то файл `header.php` вставиться тільки 1 раз. Помилки при цьому не буде.

Загалом, для підключення HTML-фрагментів сторінки краще використовувати `require_once`.

Якщо у файлі `D:\xampp\htdocs\mysite\index.php` написати

```
require_once("header.php");
```

то в якій папці буде шукатися файл `header.php`?

У тій же папці `D:\xampp\htdocs\mysite`.

Якщо файл у тій же папці, можна також написати

```
require_once("./header.php");
```

Крапка означає поточну папку.

Якщо ж файл header.php знаходиться в підпапці `D:\xampp\htdocs\mysite\layout`, тоді слід писати

```
require_once("layout/header.php");
```

А якщо файл `header.php` знаходиться на одну папку вище, тобто в `D:\xampp\htdocs\`, тоді слід писати дві крапки підряд.

```
require_once("../header.php");
```

На дві папки вище (у `D:\xampp\`):

```
require_once("../../header.php");
```

Це називається *відносний шлях*. Тобто шлях до іншого файла відносно поточного файла. Можна вказувати і повний, тобто, *абсолютий шлях*.

```
require_once("D:\\xampp\\htdocs\\mysite\\
layout\\header.php");
```

Але так робити не рекомендується, оскільки при перенесенні файлів до іншої папки чи сервера треба буде редагувати код.

Після винесення структурних частин вебсторінки файл `index.php` матиме вигляд:

```
<?php
  require_once("./header.php");
  require_once("left_menu.php");
?>
  <div id="content">...</div>
<?php
  require_once("./footer.php");
?>
```

Для перевірки, чи файл існує, можна використовувати функцію

```
file_exists("шлях_до_файлу");
```

Вона повертає `true` або `false`.

Розділ 2

Синтаксис PHP

2.1 Змінні та їх типи

Усі змінні в PHP починаються зі значка `$`.

Наприклад, `$a`, `$b`, `$student_count`.

Назва змінної може містити тільки англійські літери, цифри і знак `_`.

Великі й малі літери у назвах змінних грають роль. `$a` і `$A` – це різні змінні.

А от назви функцій нечутливі до регістру. `Func()` і `func()` – це одна й та ж функція.

Змінні у PHP оголошувати не потрібно. Їм зразу треба присвоювати значення.

```
$a = 5;
```

Тип змінної визначається автоматично.

Основні типи даних у PHP:

- `int`: 5, 7, 0, -2 (4 або 8 байт, тільки знакові)
- `float`: 2.5, 3.14 (8 байт)

- `double`: те саме, що `float`
- `string`: "Hello world"
- `bool`: true або false
- `array`: будь-який масив
- `object`: об'єкт будь-якого класу

Типу даних `char` у PHP немає. Символи належать до типу `string`.

Перетворення типів даних. Щоб перетворити значення з одного типу на інший, треба написати новий тип даних у круглих дужках перед значенням, наприклад:

- `(int)`, `(integer)` перетворює значення до типу `int`
- `(bool)`, `(boolean)` -> `bool`
- `(float)`, `(double)`, `(real)` -> `float`
- `(string)` -> `string`
- `(array)` -> `array`
- `(object)` -> `object`

Наприклад:

```
$a = 5;
$b = (bool)$a;
```

Вивести значення змінної на екран можна за допомогою таких функцій:

```
echo $a; // або echo($a); виведе 5
print $a; // або print($a); виведе 5
var_dump($a); // виведе int(5)
```

Коментарі у PHP бувають:

- однорядкові (`//` або `#`)
- багаторядкові (`/* ... */`)

2.2 Перевірка умов

Оператор `if` працює аналогічно, як і в інших мовах програмування. Якщо умова в дужках справджується (істинна), то виконується блок коду одразу після `if`. Інакше, якщо умова хибна, то спрацьовує блок коду після `else`. Хоча `if` може бути і без `else`.

```
$a = 5;
if($a > 0)
{
  echo "a > 0";
}
else
{
  echo "a <= 0";
}
```

Результат: `a > 0`

Між `if` та `else` можуть бути також перевірки додаткових умов за допомогою `else if`.

```
$a = -3;
if($a > 0) {
  echo "a > 0";
} else if(a < 0) {
  echo "a < 0";
} else {
```

```
  echo "a = 0";
}
```

Результат: `a < 0`

У PHP також є альтернативний синтаксис із використанням `if...endif` замість фігурних дужок:

```
$a = -3;
if($a > 0):
  echo "a > 0";
elseif(a < 0):
  echo "a < 0";
else:
  echo "a = 0";
endif;
```

Перевіримо, чи поточний рік є парним:

```
<?php
if(date("Y")%2 == 0) {
  echo "<div>Парний рік</div>";
} else {
  echo "<div>Непарний рік</div>";
}
?>
```

Результат: `<div>Непарний рік</div>`

Інший запис із винесенням HTML-тегів за межі PHP-коду:

```
<?php if(date("Y")%2 == 0) { ?>
<div>Парний рік</div>
<?php } else { ?>
<div>Непарний рік</div>
<?php } ?>
```

Результат: `<div>Непарний рік</div>`

Існує також оператор `switch`, який у деяких випадках можна використовувати замість `if`:

```
switch(date("Y") % 2) {
  case 0:
    echo "Парний рік";
    break;
  case 1:
    echo "Непарний рік";
    break;
}
```

2.3 Цикли

Цикл for використовується, щоб виконати певні однотипні дії кілька разів. У круглих дужках вказується початкове значення лічильника (ініціалізація), умову виходу з циклу та на скільки збільшується чи зменшується лічильник кожного разу (корекція).

```
for($i=1; $i<=10; $i++)
{
  echo $i;
}
```

Результат: `12345678910`

Щоб кожне число було з нового рядка у вихідному коді вебсторінки, слід дописати символ переходу на новий рядок `\n`.

```
for($i=1; $i<=10; $i++)
{
  echo $i;
```

```
  echo "\n";
}
```

В HTML-коді числа дійсно будуть з нового рядка. Але браузер ігнорує символи `\n` і замінює їх пробілами.

Щоб примусити браузер переходити на новий рядок, слід дописувати HTML-тег переходу на новий рядок `<br>`:

```
for($i=1; $i<=10; $i++)
{
  echo $i;
  echo "<br />";
}
```

Результат: `1<br />2<br />3<br />4<br />5<br />6<br />7<br />8<br />9<br />10<br />`

Можна також поєднати `\n` та `<br>`.

Цикл while є подібним до `for`. Попередній код можна записати у вигляді циклу `while`:

```
$i=1;
while($i<=10)
{
  echo $i;
  echo "<br />";
  $i++;
}
```

Цикл do...while. На відміну від попередніх циклів з передумовою, де умова перевіряється перед виконанням тіла циклу, цикл `do...while` є циклом з післяумовою. Тут тіло циклу завжди виконається принаймні один раз, а вже потім, якщо умова `true`, то повториться.

```
$i=1;
do
{
  echo $i;
  echo "<br />";
  $i++;
}
while($i<=10);
```

2.4 Масиви

Як і змінні, масиви в PHP не потрібно оголошувати. Їм зразу треба задавати значення.

```
$cars[0] = "BMW";
$cars[1] = "Audi";
$cars[2] = "Mercedes-Benz";
$cars[3] = "Toyota";
```

Можна і одним рядком:

```
$cars = array("BMW", "Audi", "Mercedes-Benz",
"Toyota");
```

У PHP >= 5.4 можна й коротше:

```
$cars = ["BMW", "Audi", "Mercedes-Benz",
"Toyota"];
```

Додавання нового елемента в кінець масиву:

```
$cars[] = "Porsche";
```

Вивід на екран окремих елементів масиву:

```
echo $cars[0]; // "BMW"
echo $cars[3]; // "Toyota"
```

Як вивести на екран усі елементи масиву зразу?

У мовах C або C++, довелося б писати цикл `for`. А в PHP для цього є окрема функція:

```
print_r($cars);
```

Результат:

```
Array ( [0] => BMW [1] => Audi
[2] => Mercedes-Benz
[3] => Toyota )
```

Є ще одна функція для виводу на екран як змінних, так і масивів:

```
var_dump($cars);
```

Результат:

```
array(4) { [0]=> string(3) "BMW"
[1]=> string(4) "Audi"
[2]=> string(13) "Mercedes-Benz"
[3]=> string(6) "Toyota" }
```

Як видно, ця функція виводить на екран не лише значення, а й тип даних кожного елемента масиву.

Що буде, якщо використати `echo` для виведення всього масиву зразу?

```
echo $cars;
```

Результат:

```
NOTICE Array to string conversion
Array
```

Тож `echo` доцільно використовувати для виводу окремих елементів масиву. А для виводу всього масиву слід використовувати функції `print_r` або `var_dump`.

На відміну від інших мов програмування, елементи масиву в PHP можуть бути різних типів.

```
$mas = [2, 3.14, "Hello", true];
var_dump($mas);
```

Результат:

```
array(4) { [0]=> int(2)
[1]=> float(3.14)
[2]=> string(5) "Hello"
[3]=> bool(true) }
```

Можна навіть створити масив у масиві.

```
$mas = [2, 3.14, ["Hello", "World"] ];
var_dump($mas);
```

Результат:

```
array(4) { [0]=> int(2)
[1]=> float(3.14)
[2]=> array(2) { [0]=> string(5) "Hello"
[1]=> string(5) "World" } }
```

На відміну від C, C++, у PHP не потрібно зберігати кількість елементів масиву в окремій змінній `n`. Є функція `count`, яка рахує кількість елементів масиву.

```
echo count($cars); // 4
```

Тож якщо потрібно перебрати всі елементи масиву в циклі:

```
$cars = ["BMW", "Audi", "Toyota"];
for($i = 0; $i < count($cars); $i++) {
  echo $cars[$i] . "<br />";
}
```

У PHP для роботи з масивами є ще цикл `foreach`:

```
$cars = ["BMW", "Audi", "Toyota"];
foreach($cars as $el) {
  echo $el . "<br />";
}
```

Тимчасова змінна `$el` міститиме копію кожного елемента. Якщо змінити `$el`, масив не зміниться.

Якщо в циклі треба змінити оригінал масиву, додається символ `&`:

```
$cars = ["BMW", "Audi", "Toyota"];
foreach($cars as &$el) {
  $el = $el . "<br />";
}
```

Тимчасова змінна `$el` міститиме оригінал кожного елемента. Якщо змінити `$el`, масив зміниться.

На відміну від інших мов програмування, в PHP нумерація елементів масиву не обов'язково має починатися з нуля. Можна задати іншу нумерацію.

```
$cars = [5=>"BMW", "Audi", "Toyota"];
print_r($cars);
```

Результат:

```
Array ( [5] => BMW [6] => Audi
[7] => Toyota )
```

Нумерація елементів може навіть іти не по порядку (розривна нумерація).

```
$cars = [
  "BMW",
  "Audi",
  5=>"Mercedes-Benz",
  "Toyota"
];
print_r($cars);
```

Результат:

```
Array ( [0] => BMW [1] => Audi
[5] => Mercedes-Benz
[6] => Toyota )
```

Інший приклад:

```
$cars = [
  3=>"BMW",
  "Audi",
  10=>"Mercedes-Benz",
  "Toyota"
];
print_r($cars);
```

Результат:

```
Array ( [3] => BMW [4] => Audi
[10] => Mercedes-Benz
[11] => Toyota )
```

Можна задавати і від'ємну нумерацію, однак наступні елементи нумеруватимуться все одно від нуля.

```
$cars = [
  -2=>"BMW",
  "Audi",
  "Mercedes-Benz",
  10=>"Toyota"
];
print_r($cars);
```

Результат:

```
Array ( [-2] => BMW [0] => Audi
[1] => Mercedes-Benz
[10] => Toyota )
```

Замість номерів елементів можна навіть підставляти текстові значення (ключі).

```
$person = [
  "first_name"=>"Andriy",
  "last_name"=>"Shevchenko",
  "birth_year"=>1976
];
print_r($person);
```

Результат:

```
Array ( [first_name] => Andriy
[last_name] => Shevchenko
[birth_year] => 1976 )
```

Масив, у якому індексами елементів (ключами) є слова, називається *асоціативним масивом* (або словником).

У масиві також можна комбінувати числові й символьні ключі. У PHP і числові масиви (списки), і асоціативні (словники) належать до типу даних `array`.

```
$person = [
  "first_name"=>"Andriy",
  "last_name"=>"Shevchenko",
  1976
];
print_r($person);
```

Результат:

```
Array ( [first_name] => Andriy
[last_name] => Shevchenko
[0] => 1976 )
```

Можна створювати й двовимірні масиви (матриці).

```
$persons = [
  [
    "first_name"=>"Andriy",
    "last_name"=>"Shevchenko",
    "birth_year"=>1976,
  ],
  [
    "first_name"=>"Andriy",
    "last_name"=>"Pyatov",
    "birth_year"=>1984,
  ],
];
print_r($persons);
```

Результат:

```
Array ( [0] => Array (
  [first_name] => Andriy
  [last_name] => Shevchenko
  [birth_year] => 1976
```

```
) [1] => Array (
  [first_name] => Andriy
  [last_name] => Pyatov
  [birth_year] => 1984
) )
```

Зверніть увагу: після останнього елемента масиву можна ставити кому. Це навіть рекомендується робити на випадок, якщо пізніше в кінці доведеться дописати новий елемент масиву.

Звернення до елементів двовимірного масиву відбувається через дві пари квадратних дужок:

```
echo $persons[0]['last_name'];
```

Результат:

```
Shevchenko
```

Сортування масиву за зростанням відбувається за допомогою функції `sort`.

```
$mas = [9,1,8];
sort($mas);
print_r($mas);
```

Результат:

```
Array ( [0] => 1 [1] => 8 [2] => 9 )
```

Сортування масиву за спаданням відбувається за допомогою функції `rsort`.

```
$mas = [9,1,8];
rsort($mas);
print_r($mas);
```

Результат:

```
Array ( [0] => 9 [1] => 8 [2] => 1 )
```

Але функції `sort` і `rsort` не підходять для асоціативних масивів.

```
$mas = ["nine"=>9,"one"=>1,"eight"=>8];
sort($mas);
print_r($mas);
```

Результат:

```
Array ( [0] => 1 [1] => 8 [2] => 9 )
```

Текстові ключі втрачаються.

Сортування асоціативного масиву за зростанням відбувається за допомогою функції `asort`.

```
$mas = ["nine"=>9,"one"=>1,"eight"=>8];
asort($mas);
print_r($mas);
```

Результат:

```
Array ( [one] => 1 [eight] => 8
[nine] => 9 )
```

Сортування асоціативного масиву за спаданням відбувається за допомогою функції `arsort`.

```
$mas = ["nine"=>9,"one"=>1,"eight"=>8];
arsort($mas);
print_r($mas);
```

Результат:

```
Array ( [nine] => 9 [eight] => 8
[one] => 1 )
```

Сортування асоціативного масиву за ключами за зростанням:

```
$mas = ["nine"=>9,"one"=>1,"eight"=>8];
ksort($mas);
print_r($mas);
```

Результат:

```
Array ( [eight] => 8 [nine] => 9
[one] => 1 )
```

Сортування асоціативного масиву за ключами за спаданням:

```
$mas = ["nine"=>9,"one"=>1,"eight"=>8];
krsort($mas);
print_r($mas);
```

Результат:

```
Array ( [one] => 1 [nine] => 9
[eight] => 8 )
```

Зверніть увагу: всі ці функції змінюють оригінал масиву.

Корисні функції для роботи з масивами:

- `array_merge` – злиття кількох масивів
- `array_diff` – різниця між масивами (елементи першого, яких немає в другому)
- `array_intersect` – перетин масивів (елементи першого, які є в другому)
- `in_array` – перевіряє, чи значення є в масиві

- `array_key_exists` – перевіряє, чи є елемент із заданим ключем у масиві
- `array_unique` – видаляє з масиву повторні значення
- `array_reverse` – обертає масив "задом наперед"
- `array_shift` – повертає перший елемент і видаляє його з масиву
- `array_unshift` – вставляє елемент на початок масиву (нумерація зміщується)
- `shuffle` – перемішує масив випадковим чином

2.5 GET-запити

У мовах C, C++, Pascal вхідні дані можна було вводити в консолі. А як же це зробити в мові PHP? Ми ж відкриваємо сайт у браузері, а не в консолі.

У браузері теж можна задавати вхідні дані. Для цього є кілька способів.

Перший спосіб: додати параметри до назви сторінки в адресному рядку браузера.

```
localhost/mysite/index.php?name=ivan&year=2000
```

Це називається *GET-параметри*. А такий запит називається *GET-запитом*.

При такому запиті в PHP автоматично створиться масив `$_GET` із двома елементами:

```
echo $_GET["name"]; // "ivan"
echo $_GET["year"]; // "2000"
```

Пригадаємо, що наразі наш файл `index.php` має вигляд:

```
<?php
  require_once("header.php");
  require_once("left_menu.php");
?>
Content...
<?php
  require_once("footer.php");
?>
```

Треба позбутися всього HTML-коду в `index.php` і винести контент в окремий файл.

Оскільки `header.php`, `left_menu.php` та `footer.php` однакові на всіх сторінках сайту, їх прийнято виносити в окрему папку `layout`.

Те, що різне на кожній сторінці (контент), прийнято поміщати до папки `views`. Тож контент головної сторінки буде у файлі `views/main.php`.

Тепер `index.php` набуде вигляду:

```
<?php
  require_once("layout/header.php");
  require_once("layout/left_menu.php");
  require_once("views/main.php");
  require_once("layout/footer.php");
?>
```

Наступне завдання – створити сторінку *Про сайт*.

Можна було б створити новий файл під назвою `about.php`:

```
<?php
require_once("layout/header.php");
require_once("layout/left_menu.php");
```

```
require_once("views/about.php");
require_once("layout/footer.php");
?>
```

Але тоді в `index.php` і `about.php` дублюється багато однакового коду.

Прийнято звертатись до всіх сторінок сайту через єдиний файл – `index.php`. Він називається *точкою входу* на сайт (entry point).

Як же розрізняти, яку саме сторінку на сайті хоче відкрити користувач? Зазвичай, це прийнято визначати GET-параметром `action` (хоча його назва може бути довільна):

```
localhost/mysite/index.php?action=main
localhost/mysite/index.php?action=about
```

Тоді в коді `index.php` перевіряємо значення цього параметра:

```
if($_GET["action"] == "main") {
  require_once("views/main.php");
}
else if($_GET["action"] == "about") {
  require_once("views/about.php");
}
```

Але виникає недолік. Якщо параметр action не вказано

```
localhost/mysite/index.php
```

або

```
localhost/mysite/
```

Тоді масив `$_GET` буде порожнім. А при зверненні до неіснуючого елемента масиву

```
if($_GET["action"] == "main")
```

на екрані з'явиться зауваження

```
NOTICE Undefined index: action
```

Щоб уникнути цього, спершу будемо перевіряти, чи існує взагалі в масиві `$_GET` елемент `action`. Для цього знадобиться функція `isset`, яка повертає `true`, якщо змінна чи елемент масиву існує.

```
if(isset($_GET["action"]) &&
  $_GET["action"] == "main") {
    require_once("views/main.php");
}
else if(isset($_GET["action"]) &&
  $_GET["action"] == "about") {
    require_once("views/about.php");
}
```

Але тоді, якщо action не вказано, не виконається жоден `if`. На сторінці не буде контенту.

Додамо в кінці `else`:

```
if(isset($_GET["action"]) &&
  $_GET["action"] == "main") {
    require_once("views/main.php");
}
else if(isset($_GET["action"]) &&
  $_GET["action"] == "about") {
    require_once("views/about.php");
}
else {
    require_once("views/main.php");
}
```

Також цей `else` спрацює тоді, коли користувач відкриє неіснуючу сторінку.

```
localhost/mysite/index.php?action=123
```

Але одержаний код має ще один недолік. Якщо потрібно додати на сайт нову сторінку, наприклад, `action=registration`, треба буде вставити ще один блок `else if`. І так для кожної нової сторінки.

```
if(isset($_GET["action"]) &&
  $_GET["action"] == "main") {
    require_once("views/main.php");
}
else if(isset($_GET["action"]) &&
  $_GET["action"] == "about") {
    require_once("views/about.php");
}
else if(isset($_GET["action"]) &&
  $_GET["action"] == "registration") {
    require_once("views/registration.php");
}
else {
    require_once("views/main.php");
}
```

Такий код можна оптимізувати, щоб він займав лише один блок `if...else`:

```
if(isset($_GET["action"]) && file_exists(...)) {
    require_once(...);
}
else {
    require_once("views/main.php");
}
```

Для цього знадобиться функція `file_exists`, яка перевірятиме, чи існує файл із потрібною назвою в папці `views`.

Тепер пункт меню *Головна* на сайті буде вказувати на сторінку

```
<a href="index.php?action=main">
```

А пункт *Про нас* – на сторінку

```
<a href="index.php?action=about">
```

2.6 POST-запити

Другий спосіб передати вхідні дані для PHP-коду – через HTML-форму.

```
<form action="..." method="...">
  <input type="text" name="login" />
  <input type="password" name="password" />
  <input type="submit" value="Надіслати" />
</form>
```

Атрибут `action` вказує, яка сторінка відкриється, коли користувач натисне кнопку "Надіслати". Якщо зробити значення порожнім `action=""`, то відкриється знову та сама сторінка.

Атрибут `method` вказує, яким способом надіслати на сервер дані, введені користувачем у формі. Метод може бути `GET` або `POST`.

Якщо надсилати форму методом GET, то все, що ввів користувач у формі, приклеїться до адресного рядка браузера.

```
<form action="index.php?action=registration"
  method="GET">
```

згенерує URL-адресу

```
localhost/mysite/index.php?action=registration&
  login=admin&password=123456
```

Так робити категорично не рекомендується, оскільки введений користувачем пароль можуть побачити інші.

Якщо надсилати форму методом POST, то все, що ввів користувач у формі, буде надіслано на сервер невидимо для користувача.

```
<form action="index.php?action=registration"
  method="POST">
```

В адресному рядку браузера не буде змін.

```
localhost/mysite/index.php?action=registration
```

Тому дані з форм майже завжди рекомендується надсилати методом POST. У PHP автоматично створиться масив `$_POST` із даними, що ввів користувач у формі.

```
echo $_POST["login"]; // admin
echo $_POST["password"]; // 123123
```

Одна PHP-сторінка може працювати в кількох режимах.

Наприклад, користувач відкриває її вперше. Йому показується HTML-форма реєстрації. Користувач заповнює поля і натискає кнопку "Надіслати". Якщо

```
<form action="">
```

то відкриється знову та сама сторінка.

Коли сторінка відкривається вдруге, можна перевіряти, чи не порожній масив `$_POST` і що там ввів користувач.

```
<?php if(!empty($_POST)) {
  if(isset($_POST["login"]) &&
    strlen($_POST["login"])<4)) {
      echo "Логін занадто короткий!";
  }
  else if(isset($_POST["password"]) &&
    ...) {
      echo "Пароль занадто слабкий!";
  }
  ...
  if(...) { // якщо помилок не було
      echo "Реєстрацію завершено!";
  }
} ?>
```

Цей код спрацює лише в другому режимі, коли користувач натиснув кнопку "Надіслати"і масив `$_POST` не порожній. А нижче у тому ж файлі може бути HTML-код форми реєстрації.

Проте даний PHP-код має недолік. Після успішної реєстрації користувача все одно спрацює HTML-код знизу. Користувач знову побачить форму реєстрації. У такому випадку після успішної реєстрації можна перенаправити користувача на іншу сторінку функцією

```
header("Location: http://localhost/
  mysite/index.php");
```

Або ж можна перервати виконання скрипту функцією

```
exit;
```

чи її синонімом

```
die;
```

Або оточити HTML-код форми PHP-оператором `if`.

```
<?php if(...) { ?>
<form action="" method="POST">
  ...
</form>
<?php } ?>
```

2.7 Регулярні вирази

Як перевірити, чи в тексті є хоча б одна цифра? Або чи в тексті є 4 букви підряд? Або чи містить текст пробіли? Або що текст є коректною датою?

Можна писати складні цикли і перевіряти кожен символ тексту. А можна використати регулярні вирази.

Регулярні вирази в будь-якій мові програмування використовуються для того, щоб перевірити, чи якийсь текст відповідає заданому шаблону (зразку).

У PHP є функція

```
preg_match(шаблон, текст)
```

яка перевіряє, чи текст відповідає шаблону. Вона повертає `true`/`false`.

Приклад: перевірити, чи в тексті є хоча б одна цифра.

```
if(preg_match("/[0-9]/", $text))
```

Перевірити, чи в тексті є 4 малі латинські літери підряд.

```
if(preg_match("/[a-z]{4}/", $text))
```

Перевірити, чи в тексті є 4 малі або великі латинські літери підряд:

```
if(preg_match("/[a-zA-Z]{4}/", $text))
```

або

```
if(preg_match("/[a-z]{4}/i", $text))
```

Тут `i` означає *case insensitive* (нечутливість до регістру).

Перевірити, чи в тексті є 5 або більше будь-яких символів.

```
if(preg_match("/.{5,}/", $text))
```

Тут крапка означає будь-який символ.

Перевірити, чи в тексті є хоча б одна крапка.

```
if(preg_match("/\./", $text))
```

Символ крапки в регулярних виразах позначається `\.`

Перевірити, чи в тексті є хоча б одна мала українська літера.

```
if(preg_match("/[а-яієїґ]/", $text))
```

Перевірити, чи в тексті є хоча б одна літера будь-якої мови.

```
if(preg_match("/\p{L}/u", $text))
```

Тут `L` означає *letter*, `u` означає *Unicode*.

Квантифікатори (кількість повторень) позначаються в регулярних виразах наступним чином:

- `{4}` – те, що зліва, має повторюватись 4 рази
- `{4,}` – 4 або більше разів
- `{2,4}` – від 2 до 4 разів

- + те саме, що `{1,}`
- * те саме, що `{0,}`
- ? те саме, що `{0,1}`

Попередні регулярні вирази перевіряли, чи є в тексті хоча б 1 або кілька певних символів. А як перевірити, чи весь текст (а не частина) має 4 цифри?

```
if(preg_match("/^[0-9]{4}$/", $text))
```

Символ ^ означає, що лівіше більше нічого не може бути (це початок тексту).

Символ $ означає, що правіше більше нічого не може бути (це кінець тексту).

Приклад:

```
if(preg_match("/[0-9]{4}/", "x1234x"))
// true

if(preg_match("/^[0-9]{4}$/", "x1234x"))
// false
```

Якщо треба перевірити наявність символів ^ або $ у тексті, то пишеться \^ і \$ відповідно.

```
preg_match("/^\$[0-9]+(\.[0-9]+)?$/", '$123.99')
// true
```

Набір символів можна об'єднувати круглими дужками.

Перевіримо, чи в тексті є слово `hello` або `world`.

```
if(preg_match("/hello|world/",
  "hello friend"))
// true
```

Символ | означає "або". Він частіше використовується для пошуку слів, а не поодиноких символів.

Регулярний вираз для перевірки коректності адреси електронної пошти:

```
if(preg_match('/^[a-zA-Z0-9\.\-_]{2,}@
  [a-zA-Z0-9\-_]+\.[a-z]{2,3}$/', $text))
```

Жадібний пошук (greedy search) – якщо використовуються символи `.*`

```
preg_match("/h.*o/", "hello world hello there")
```

Шукає якомога більше (до останньої появи літери о).

Лінивий пошук (lazy search) – якщо використовуються символи `.*?`

```
preg_match("/h.*?o/", "hello world hello there")
```

Шукає якомога менше (до першої появи літери о).

Корисні сайти для тестування регулярних виразів:

- https://regex101.com
- https://regexr.com

2.8 Робота з базами даних

Як зберігати список зареєстрованих на сайті користувачів?

Можна зберігати їх у текстовому файлі. Кожен новий користувач буде дописуватися в кінець файлу.

А якщо користувачів мільйон? Десять мільйонів? Як знайти потрібного користувача?

Просканувати весь файл від початку до кінця.

Виникає проблема: що більший файл, то довше триватиме пошук.

Як вирішується проблема пошуку в книжках? У книзі створюється зміст або алфавітний покажчик. Тоді легше знайти ту чи іншу інформацію. Достатньо знайти її у змісті і одразу відкрити відповідну сторінку.

Із текстовим файлом можна зробити так само. Створити окремо індекси (зміст), щоб можна легко і швидко знайти потрібну інформацію. Можна навіть створити кілька індексів. У книзі зміст подає інформацію, згруповану за темами, а алфавітний покажчик – за алфавітом. Так можна створити й індекси для пошуку користувача за датою реєстрації, алфавітом, місцем проживання тощо. Для цього існують бази даних.

По суті, база даних – це набір таблиць (які можуть бути пов'язані між собою) та їхніх індексів (для швидкого пошуку).

Навіщо ж потрібні бази даних, якщо є таблиці Microsoft Excel?

	A	B	C	D	E
1	**Логін**	**Пароль**	**Дата реєстрації**	**Електронна пошта**	
2	user1	111111	01.10.2019	user1@gmail.com	
3	user2	123321	08.10.2019	user2@gmail.com	
4	banana	@#$^&*()	14.10.2019	banana@example.com	
5					
6					
7					
8					
9					
10					
11					
12					

Рис. 2.1: Приклад таблиці Microsoft Excel зі списком користувачів

PHP запросто може працювати з Excel-документами. Однак, як і у випадку з текстовими файлами, доведеться

вручну програмувати операції пошуку, вставки і видалення рядків.

При роботі з базами даних цього програмувати не потрібно.

Є багато програм із уже готовими командами для роботи з БД. Програми для роботи з БД називаються *системами керування базами даних* (СКБД) (англ. Database Management System – DBMS).

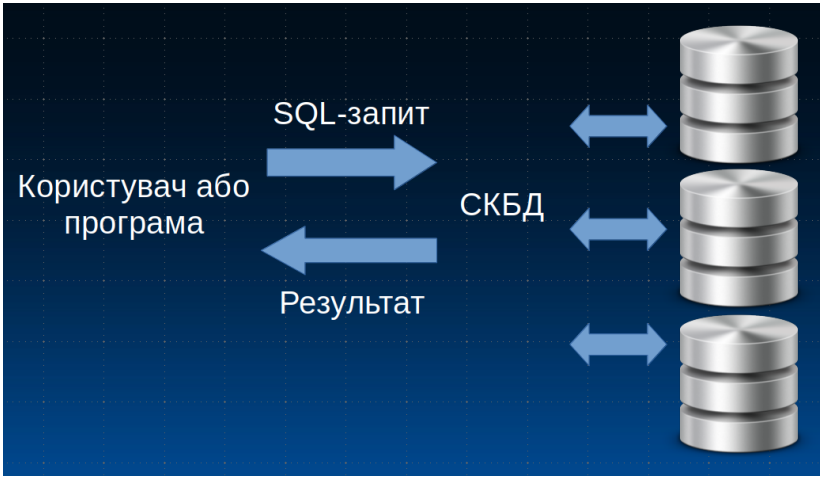

Рис. 2.2: Схема роботи з СКБД

Наразі існує багато СКБД. Найвідоміші серед них:

- MySQL
- PostgreSQL
- SQLite
- MS SQL Server
- MS Access

Є також повністю безкоштовний різновид MySQL під назвою MariaDB.

Більшість СКБД зберігають таблиці та їхні індекси в окремих файлах. SQLite і MS Access зберігають цілу БД в одному файлі.

Більшість СКБД мають майже однаковий набір команд для пошуку, додавання та видалення даних з БД. Ці команди утворюють мову *SQL* (Structured Query Language).

SQL не є окремою незалежною мовою програмування. Це скоріше мова взаємодії користувача з БД.

Майже кожна мова програмування дозволяє працювати з БД за допомогою *SQL-запитів* (SQL queries). Користувач також може працювати напряму з БД через консоль і SQL-запити (рис. 2.3).

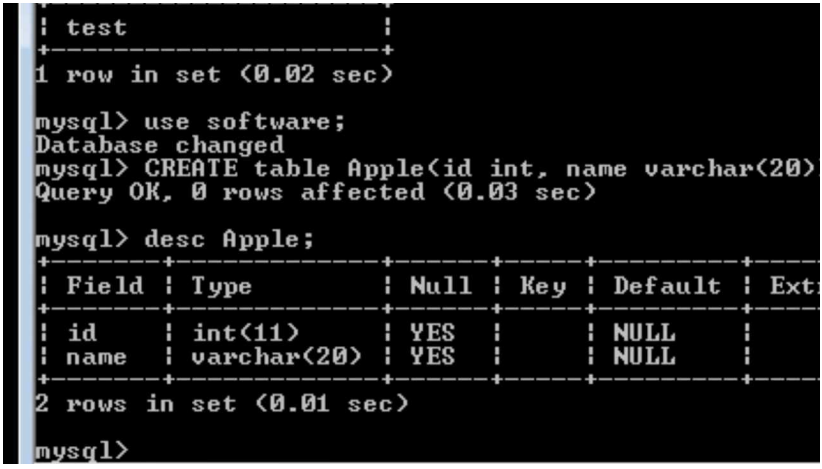

Рис. 2.3: Робота з MySQL через консоль

Кожна БД складається з таблиць. Таблиці прийнято називати малими латинськими літерами без пробілів. Наприклад, `users`, `posts`, `comments`.

Таблиці прийнято називати іменниками у множині,

оскільки таблиця містить інформацію не про один об'єкт, а багато.

Кожна таблиця складається з рядків і стовпців (полів). Поля також прийнято називати малими латинськими літерами без пробілів. Наприклад, `username`, `login`, `password`, `email`, `reg_date`.

Наприклад, структура (список полів) таблиці `users`:

- `username` (текст)
- `login` (текст)
- `password` (текст)
- `email` (текст)
- `age` (число)
- `reg_date` (дата)

Припустимо, в таблиці `users` є 2 користувачі з однаковим іменем `username`. Як їх розрізняти?

Прийнято кожному рядку давати унікальний номер (`id` – ідентифікатор).

Щоб розрізняти схожі дані, прийнято, що кожна таблиця повинна мати поле, значення якого різне для кожного рядка. Таке поле, яке має унікальне значення для кожного рядка, називається *первинним ключем* (primary key). Зазвичай його називають `id`.

Нова структура таблиці users:

- `id` (число)
- `username` (текст)
- `login` (текст)

id	username	login	password	email	reg_date
1	Bob	bob	111111	bob@example.com	2019-10-01
2	Alice	alice	123456	alice@meta.ua	2019-10-08
3	Bob	bob	password	bob74@gmail.com	2019-10-14

Рис. 2.4: Структура таблиці `users`

- `password` (текст)
- `email` (текст)
- `age` (число)
- `reg_date` (дата)

У багатьох СКБД є можливість зробити так, щоб поле `id` збільшувалось автоматично для кожного нового рядка таблиці. Зазвичай, така властивість називається `AUTO_INCREMENT`.

Надалі ми будемо використовувати СКБД MySQL (або MariaDB). Ця СКБД постачається у складі пакету програм *XAMPP*.

Щоб не працювати з MySQL через консоль, є безкоштовний зручний вебдодаток *phpMyAdmin*, який також

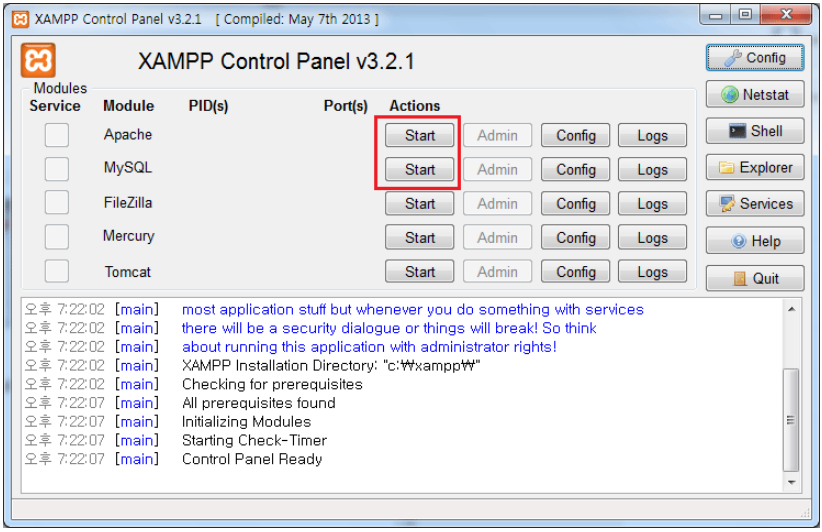

Рис. 2.5: Як запустити сервер Apache у вікні *XAMPP*

постачається у складі XAMPP. Його можна відкрити у браузері, ввівши адресу

`localhost/phpmyadmin`

Для роботи з БД MySQL потрібно ввести окремий логін і пароль (рис. 2.6). За замовчуванням, XAMPP у Windows встановлює логін `root`, пароль порожній.

Створити нову БД можна в консолі за допомогою SQL-запитів. Але для початку це зручно зробити візуальним способом у phpMyAdmin. Відкриємо вкладку *Databases* (рис. 2.7).

Потрібно ввести назву нової БД у текстовому полі (без пробілів) і задати кодування символів. Рекомендується `utf8mb4_unicode_ci` (рис. 2.8).

Новостворена база даних поки що не має таблиць (рис. 2.9).

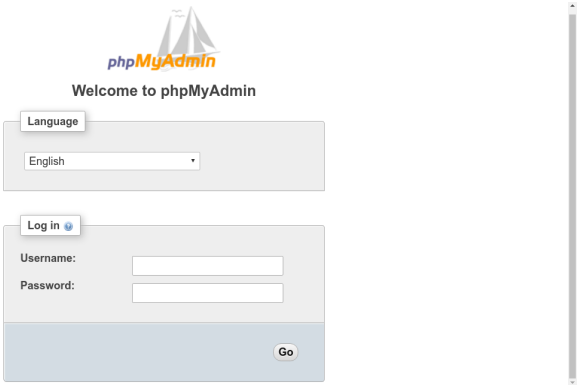

Рис. 2.6: Вебінтерфейс *phpMyAdmin*, сторінка авторизації

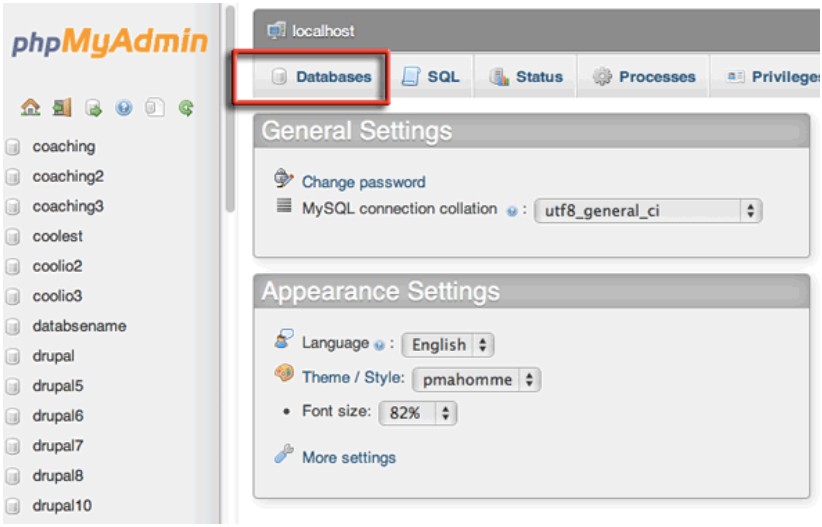

Рис. 2.7: Вкладка *Databases*

Введемо назву нової таблиці `users`. І кількість полів у ній (рис. 2.10).

Далі заповнюємо назву і тип кожного поля таблиці. Для цілих чисел залишаємо `INT` (рис. 2.11).

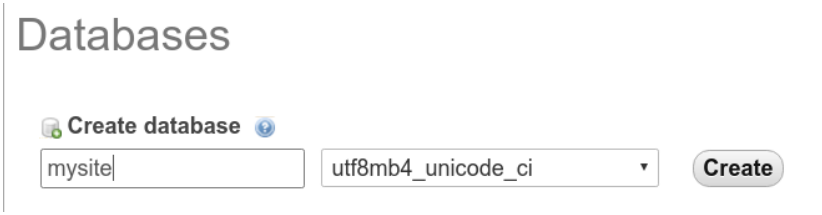

Рис. 2.8: Створення нової бази даних

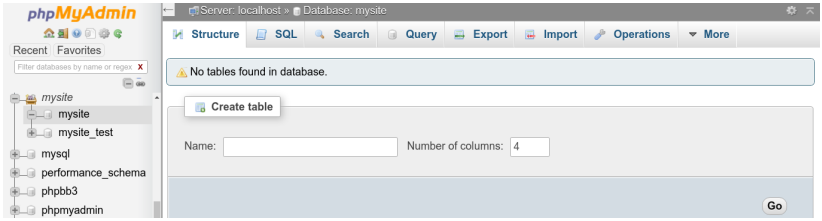

Рис. 2.9: Новостворена база даних поки що не має таблиць

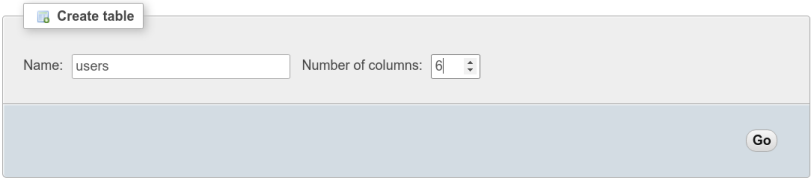

Рис. 2.10: Створення нової таблиці

Для поля `id` ставимо галочку `A_I` (`AUTO_INCREMENT`). Для короткого тексту (логін, пароль, e-mail) обираємо `VARCHAR` і вписуємо довжину (length) 255.

У MySQL є також тип даних `TEXT`, але він, як правило, використовується для довгого тексту. Наприклад, текст статті чи коментаря.

Для дати обираємо тип `TIMESTAMP` і обираємо в default `CURRENT_TIMESTAMP`. Тоді поточна дата й час підставлятимуться автоматично при додаванні нового рядка таблиці.

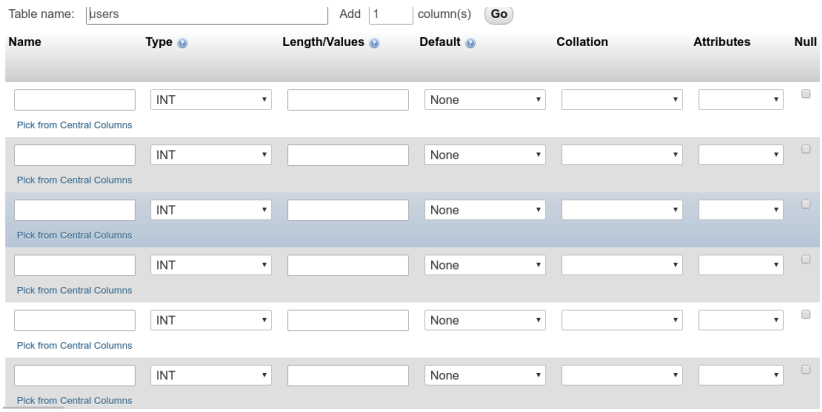

Рис. 2.11: Створення полів таблиці та вказання їхніх типів

У MySQL є також типи даних `DATE` і `DATETIME`, але вони не враховують часовий пояс користувача.

У phpMyAdmin можна також вказати тип поля `BOOLEAN`, однак у MySQL немає такого типу даних. Насправді це поле збережеться як `TINYINT(1)`, тобто ціле одноцифрове число.

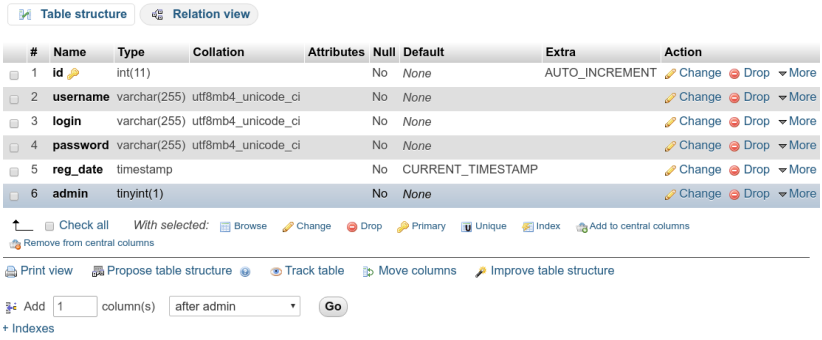

Рис. 2.12: Таблиця з полями

На вкладці *Browse* буде видно, що в новоствореній таблиці немає рядків (рис. 2.13).

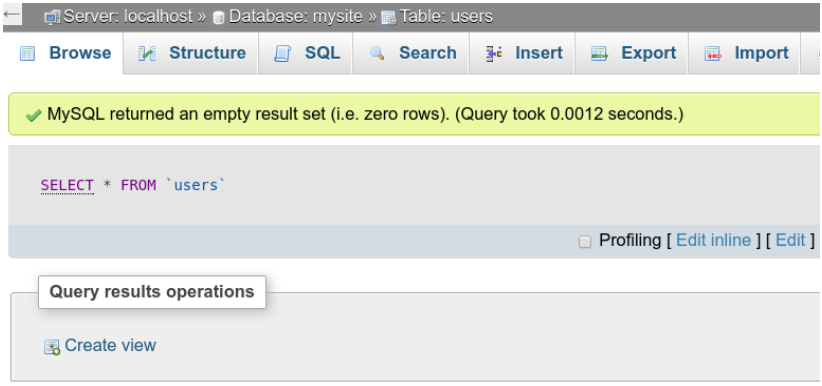

Рис. 2.13: Таблиця без рядків

Новий рядок до таблиці можна вставити за допомогою SQL-запиту або ж візуальним способом у phpMyAdmin. Для цього треба відкрити вкладку *Insert* (рис. 2.14).

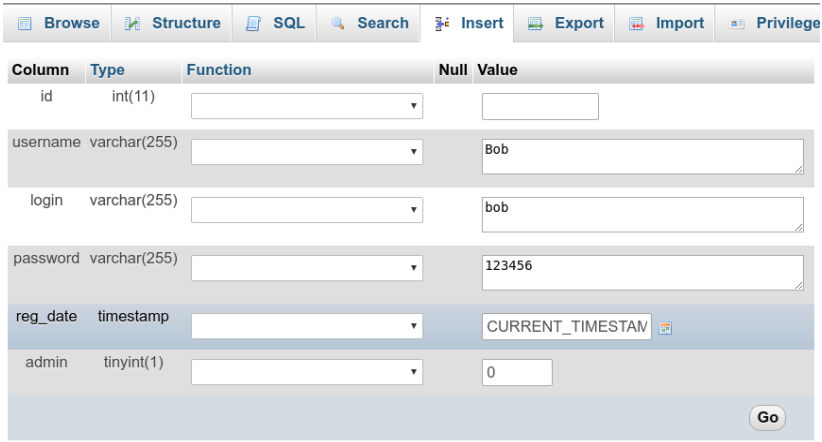

Рис. 2.14: Вкладка *Insert*

Значення поля `id` слід залишити порожнім. Воно заповниться автоматично завдяки властивості `AUTO_INCRE-`

`MENT`. Перший рядок матиме id = 1, наступний 2 і т.д.

Якщо є поле дата (`TIMESTAMP`), його теж залишаємо порожнім. Воно заповниться автоматично завдяки властивості `CURRENT_TIMESTAMP`.

Зберігаємо новий рядок.

На вкладці *Browse* тепер видно, що таблиця має 1 рядок (рис. 2.15).

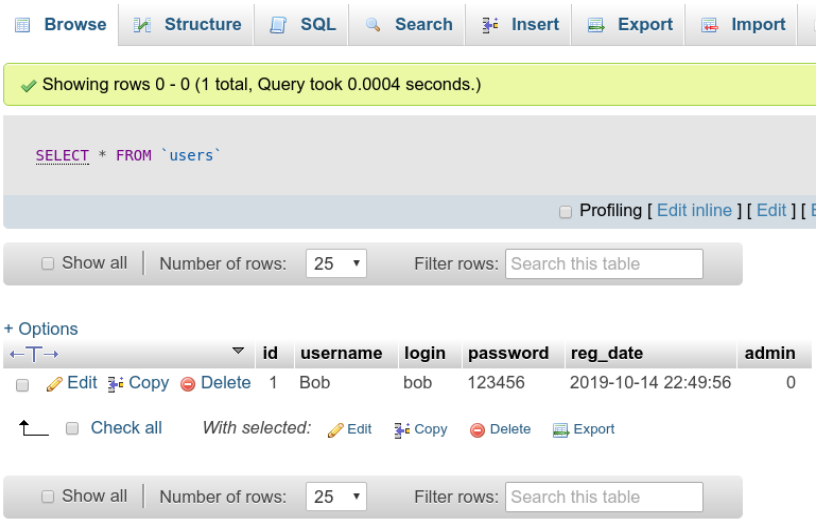

Рис. 2.15: Новостворений рядок на вкладці *Browse*

Якщо придивитися, у верхній частині вікна phpMyAdmin написано, який SQL-запит було виконано, щоб показати даний результат (рис. 2.16).

По суті, при кожному натисканні мишею за кадром виконується певний SQL-запит. phpMyAdmin надає зручний інтерфейс і дозволяє не задумуватись про SQL.

Але програмістам слід уміти працювати з БД за допомогою SQL-запитів.

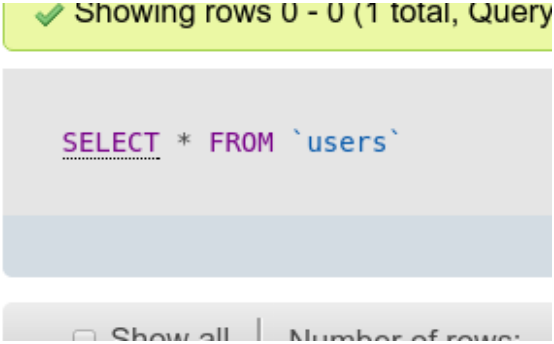

Рис. 2.16: Виконаний SQL-запит

У phpMyAdmin можна також вводити SQL-запити напряму на вкладці `SQL` (рис. 2.17).

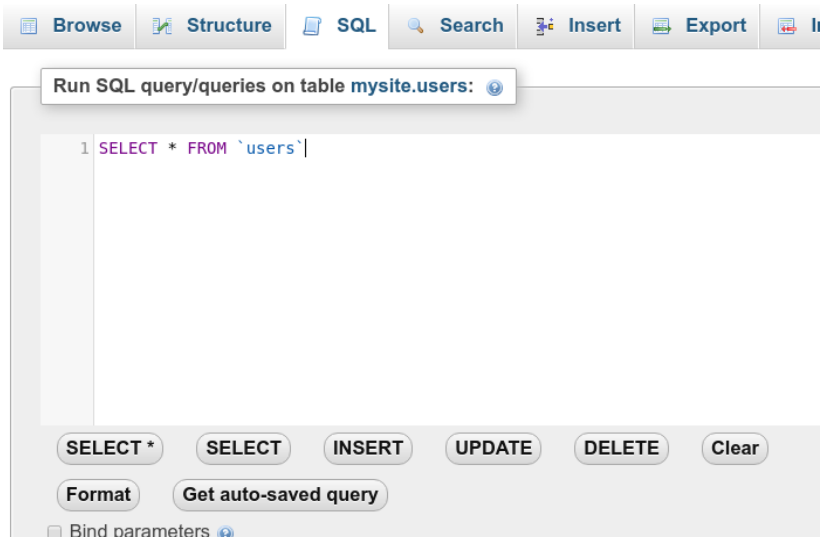

Рис. 2.17: Вкладка *SQL*

Існують 4 основі типи операцій з БД.
1. Пошук інформації в БД.

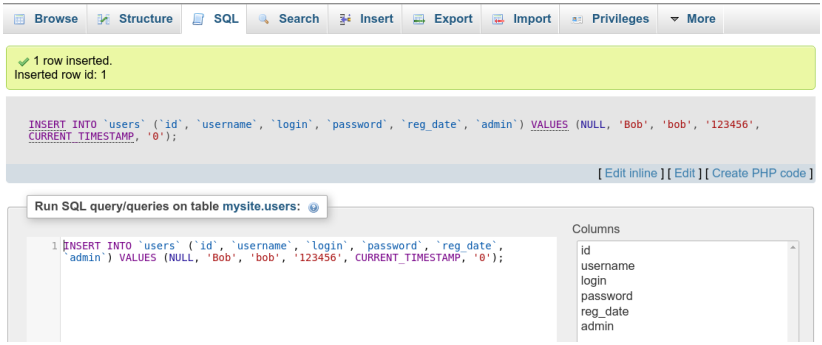

Рис. 2.18: Приклад SQL-запиту для додавання нового рядка в таблицю

```
SELECT * FROM users;
```

Тут * означає витягти всі поля таблиці. Крапка з комою в кінці запиту необов'язкова.

```
SELECT login, password FROM users
```

Витягти не всі поля таблиці, а лише поля `login` і `password`. Тобто лише логіни й паролі користувачів.

```
SELECT login, password FROM users WHERE id=2
```

Витягти логін і пароль лише користувача, в якого id=2.

```
SELECT login, password FROM users
  WHERE id=1 OR id=2
```

Витягти логін і пароль користувача, в якого id=1 або id=2. Тобто результат матиме інформацію про обох користувачів (2 рядки), якщо вони існують.

Цей запит можна записати коротшим способом за допомогою оператора `IN`.

```
SELECT login, password FROM users
  WHERE id IN (1,2)
```

2. Додавання інформації до БД.

```
INSERT INTO users (login,password,email)
  VALUES ('alice', '123456', 'alice@meta.ua')
```

Текст може бути як в одинарних, так і в подвійних лапках.

3. Редагування інформації в БД.

```
UPDATE users SET password='hello',
  email='bobbb@gmail.com'
  WHERE id=1
```

Вказувати `WHERE` обов'язково, інакше будуть відредаговані ВСІ рядки!

4. Видалення інформації з БД.

```
DELETE FROM users WHERE id=2
```

Вказувати `WHERE` обов'язково, інакше будуть видалені ВСІ рядки!

Запит `SELECT` повертає результат. Інші типи запитів – ні.

Примітка: назви таблиць та полів дозволяється брати в кутові лапки ‘, але тільки в кутові.

```
SELECT ‘login‘, ‘password‘ FROM ‘users‘ WHERE id=2
```

Кутові лапки знаходяться зліва від цифри 1 на клавіатурі (англійська розкладка).

Як виконувати SQL-запити у PHP-коді?

Для цього є 2 способи: процедурний і об'єктно-орієнтований. Працюють вони ідентично.

1. Процедурний стиль.

```
<?php
$link = mysqli_connect("localhost", "логін_БД",
  "пароль_БД", "назва_БД");
if (mysqli_connect_errno() != 0) {
    die(mysqli_connect_error());
}
```

Змінна `$link` міститиме вказівник на з'єднання з БД.

Функція `mysqli_connect_errno()` повертає 0, якщо вдалося успішно підключитися до БД. Якщо вона повернула не 0, то слід перевірити, чи правильно вказано логін, пароль або назву БД.

Функція `mysqli_connect_error()` містить словесний опис помилки, якщо вона виникла.

```
$query = "SELECT * FROM users";
$result = mysqli_query($link, $query);
if(mysqli_errno($link) != 0) {
    die(mysqli_error($link));
}
while($row = mysqli_fetch_assoc($result)) {
    echo $row["username"];
    echo $row["login"];
    echo $row["email"];
}
```

Будь-які запити до БД у PHP виконуються за допомогою функції `mysqli_query`. Перший параметр – до якої БД звертаємось (може бути відкрито кілька БД одночасно). Другий параметр – текст SQL-запиту.

Змінна `$result` міститиме "запакований"результат з БД, усі рядки одразу. Зчитувати треба їх звідти в циклі по одному рядку за допомогою функції

```
mysqli_fetch_assoc($result)
```

Коли рядків більше немає, функція поверне `NULL`, і цикл припиниться.

У кінці можна закрити з'єднання з БД функцією

```
mysqli_close($link)
```

Але це не обов'язково. Усі з'єднання з БД закриються автоматично після завершення PHP-скрипта.

2. Об'єктно-орієнтований стиль.

```
<?php
$mysqli = new mysqli("localhost", "логін_БД",
  "пароль_БД", "назва_БД");
if ($mysqli->connect_errno != 0) {
    die($mysqli->connect_error);
}
```

Тут змінна `$mysqli` – це об'єкт класу `mysqli`.

```
$query = "SELECT * FROM users";
$result = $mysqli->query($query);
if($mysqli->errno != 0) {
    die($mysqli->error);
}
while($row = $result->fetch_assoc( )) {
    echo $row["username"];
    echo $row["login"];
    echo $row["email"];
}
```

Тут змінна `$result` – це об'єкт класу `mysqli_result`.

У кінці можна закрити з'єднання з БД функцією

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

Але це не обов'язково.

2.9 Файли cookies

У PHP всі змінні очищаються після завершення виконання скрипта. Тобто коли сторінка повністю завантажилась у браузері.

При наступному завантаженні сторінки всі змінні створюються заново.

Бувають випадки, коли треба зберегти якусь однакову інформацію в браузері для всіх сторінок певного сайту. Наприклад, ознаку того, що користувач авторизувався на сайті. Інакше доведеться вводити заново логін і пароль на кожній сторінці сайту.

У вебпрограмуванні існує механізм зберігання інформації у браузері клієнта для всіх або деяких сторінок певного сайту. Це файли *cookies*.

По суті, це текстові файли, які зберігаються в певній папці браузера.

Файли cookies встановлюються для кожного вебсайту окремо (рис. 2.19).

Якщо браузер має файли cookies для певного сайту, то вміст цього файлу надсилається на сервер при відкритті кожної сторінки даного сайту (рис. 2.20).

Наприклад, користувач ввів логін і пароль до електронної пошти.

Сервер відповів, що дані введено вірно і сказав браузерові зберегти дані про те, що користувач авторизувався, у файлі cookie.

Користувач відкриває папку "Вхідні листи".

Браузер надсилає серверові запит відкрити відповідну сторінку, а також файл cookie, де сказано, що користувач уже авторизований.

Сервер перевіряє у файлі cookie, що користувачеві не треба заново вводити логін і пароль. Тож він має доступ

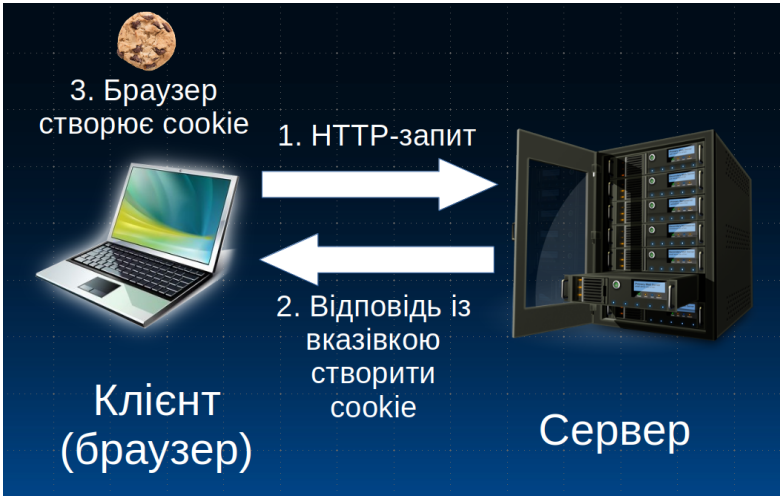

Рис. 2.19: Після першого запиту створюються файли *cookie*

Рис. 2.20: Наступні запити відправлятимуться з файлами *cookie*

до своїх листів.

Сервер показує браузеру список вхідних листів користувача.

Такий механізм повторюється на всіх сторінках одного сайту.

З метою безпеки файли cookies, створені для одного сайту, недоступні для інших.

Файли cookies, можна створювати як на стороні сервера (PHP), так і клієнта (JS). Але зберігатись вони будуть лише на стороні клієнта (у браузері).

Кожен файл cookie має назву і вміст. А також термін придатності. Тобто час, після якого браузер автоматично видалить цей файл cookie.

Як вказати у PHP-коді, що браузер користувача має створити новий файл cookie?

```
setcookie("auth", 1);
```

Створиться новий файл cookie з назвою `auth` і вмістом `1`. Цей файл самознищиться, коли користувач закриє браузер.

```
setcookie("auth", 1, time() + 3600, "/php/");
```

Четвертий параметр означає, що цей файл cookie буде діяти не на всіх сторінках даного сайту, а тільки на тих, що в папці `php` на сервері.

Слід зазначити, що файл cookie буде видимий для сервера лише після перезапуску сторінки.

Як прочитати вміст файлу cookie, якщо браузер прислав його на сервер?

Уся інформація з файлів cookies буде знаходитись у суперглобальному масиві `$_COOKIE`.

```
echo $_COOKIE["auth"];
```

Зверніть увагу: `$_COOKIE`, а не `$_COOKIES` (в однині, без S у кінці).

Як видалити файл cookie у PHP-коді?

1. Він видалиться автоматично, коли закінчиться термін його дії.

2. `unset($_COOKIE["auth"]);`

3. `setcookie("auth , time()-1);` (встановити минулий час)

Як переглянути файли cookie для даного сайту в браузері?

Якщо це Google Chrome, то відкрити консоль розробника (F12) (рис. 2.21).

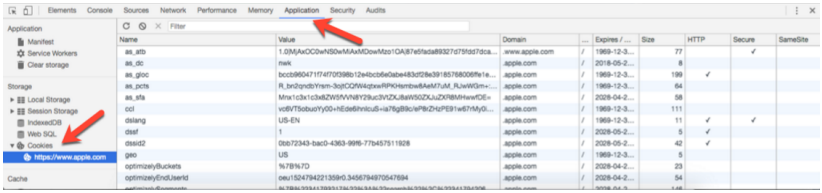

Рис. 2.21: Перевірка файлів *cookie* у браузері *Chrome*

У 2009 р. Євросоюз прийняв постанову, згідно з якою сайти повинні питати дозволу користувача на створення файлів cookies із приватною інформацією. Навіть якщо сервер із сайтом фізично знаходиться не в ЄС, відвідувачі сайту все одно можуть бути з ЄС. Використання cookies без дозволу користувача може бути підставою для судового позову.

Після цього на багатьох сайтах з'явилися набридливі сповіщення про використання cookies.

Щоправда, у 2015 р. було внесено зміни до закону, згідно якого сайтам не потрібно питати дозволу на використання аналітичних cookies (використовуються для статистики відвідувань сайту).

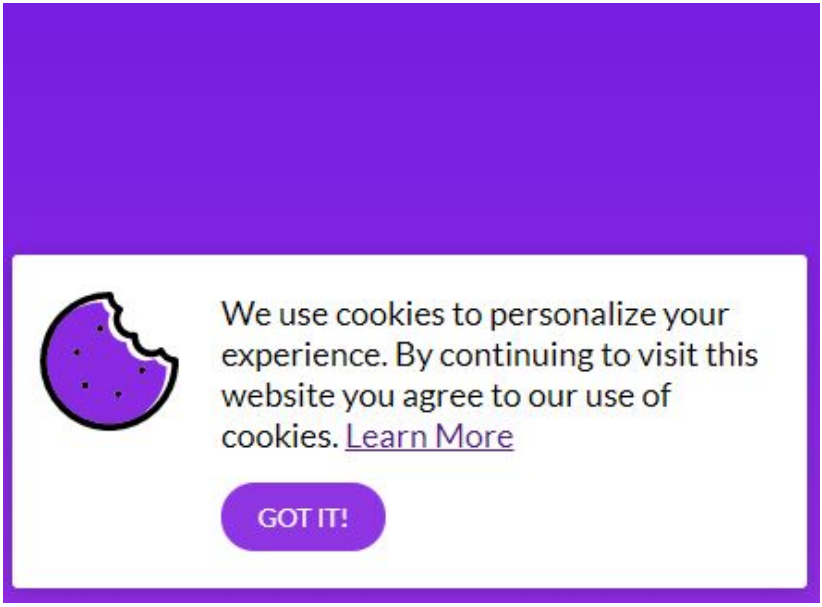

Рис. 2.22: Типове попередження про використання файлів *cookie*

2.10 Сесія користувача

Ми вже знаємо, що можна зберігати довготривалу інформацію для сайту в файлах cookies на боці клієнта (браузера).

Але ці файли легко підробити вручну. Тобто, користувач може відкрити і написати у файлі, що він авторизований на сайті, навіть не маючи паролю. Це небезпечно.

Інший спосіб зберігати важливу інформацію – на сервері.

Такий механізм називається *сесією користувача* (session).

По суті, сесія – це теж текстові файли з інформацією, але зберігаються вони на боці сервера.

Для кожного браузера, з якого заходять на сайт, створюється окремий файл сесії.

Як працювати з сесією в PHP-коді?

Для початку треба увімкнути механізм сесій.

```
session_start();
```

Зазвичай, цю функцію пишуть на початку файлу `index.php`. Після цього всі дані, які потрібно зберегти на довгий час, можна записувати в суперглобальний масив `$_SESSION`.

Наприклад, ознаку того, що користувач уже авторизований на сайті:

```
$_SESSION["auth"] = true;
```

Ця змінна збережеться в масиві `$_SESSION` на всіх сторінках сайту для даного користувача (браузера).

Як завершити сесію користувача, коли він виходить зі свого акаунту?

Просто очистити масив `$_SESSION`. Для цього є зручна функція

```
session_destroy();
```

Але звідки сервер знає, якому браузеру належить який файл із сесією?

Кожен файл сесії має унікальну назву.

Крім того, сервер створює на стороні клієнта файл cookie під назвою `PHPSESSID`, в якому міститься назва файлу сесії на сервері. Тож, по суті, сесія також використовує файли cookies.

2.11 Створення функцій

У PHP є сотні вбудованих функцій на різні випадки життя.

Але запросто можна створювати і власні функції.

```
// опис функції
function printHello()
{
    echo "Hello";
}
printHello(); // виклик функції
```

Назви функцій у PHP нечутливі до регістру. `Func()` і `func()` – це одна функція.

А назви змінних чутливі до регістру. `$a` і `$A` – це різні змінні.

Функція може отримувати параметри. Вказується лише їхня назва, без типу.

```
function printSum($a, $b)
{
    echo $a + $b;
}
printSum(3,5); // виведе 8
```

Щоправда, у PHP 7 додано можливість явно вказувати типи параметрів.

```
function printSum(int $a, int $b)
{
    echo $a + $b;
}
```

Значення `$a` і `$b` буде автоматично перетворено в `int`.

Параметри можуть мати значення за замовчуванням.

```
function printSum($a=2, $b=3)
{
    echo $a + $b;
}
```

Тоді `printSum();` надрукує `5`.

Параметри за замовчуванням завжди повинні бути в кінці.

```
function printSum($a=2, $b)
// невірно, бо printSum(4); спричинить помилку,
// що $b не вказано

function printSum($a, $b=3)
// вірно, printSum(4); підставить $a=4, а $b=3
```

Параметри у функцію передаються як копія.

```
function inc($a)
{
    $a++;
}
$x = 0; inc($x); echo $x;
// виведе 0
```

Щоб передати параметри як оригінал, треба написати знак `&`.

```
function inc(&$a)
{
    $a++;
}
$x = 0; inc($x); echo $x;
// виведе 1
```

Функція може повертати значення будь-якого типу за допомогою ключового слова `return`.

```
function sum($a, $b)
{
    return $a + $b;
}
```

У PHP 7 додано можливість явно вказувати, який тип повертає функція.

```
function sum($a, $b) : int
{
    return $a + $b;
}
```

Якщо вказується тип результату, то краще вже вказувати і тип вхідних параметрів.

```
function sum(int $a, int $b) : int
{
    return $a + $b;
}
```

Увага! Якщо функція не має `return`, то вона все одно повертає `null`.

До речі, `null` можна писати в PHP як великими, так і малими літерами. Зазвичай, пишуть малими.

На відміну від інших мов програмування, функції у PHP не бачать глобальних змінних.

```
$a = 5;
function f() {
    echo $a; // неіснуюча змінна
}
f(); // NOTICE Undefined variable: a
```

Треба явно вказати функції, які глобальні змінні вона повинна бачити.

```
$a = 5;
function f() {
    global $a;
    echo $a; // 5
}
f();
```

Це не стосується суперглобальних масивів PHP `$_GET`, `$_POST`, `$_COOKIE`, `$_SESSION`. Вони завжди видимі у всіх функціях.

Розділ 3

PHP та ООП

3.1 Класи та об'єкти

PHP дозволяє писати код як із класами, так і без них.

Класи дозволяють описати певний тип об'єктів один раз, а потім створювати скільки завгодно об'єктів даного типу.

Наприклад, клас, що описує коло на площині:

```
class Circle
{
    public $x, $y, $r;
    public function length()
    {
        return 2 * 3.14 * $this->r;
    }
}
```

Назви класів прийнято писати з великої літери.

Тіло класу пишеться у фігурних дужках `{}`. На відміну від C++, після закриття класу `}` крапка з комою НЕ ставиться.

Змінні всередині класу називаються *полями класу*. Функції всередині класу називаються *методами класу*.

Якщо поле класу є *публічним* (public), то воно буде доступним за межами класу.

```
class Circle
{
    public $x, $y, $r;
}
$c = new Circle;
echo $c->r; // працює
```

Якщо поле класу є *приватним* (private), то доступ до нього будуть мати лише функції даного класу (методи).

```
class Circle
{
    private $x, $y, $r;
}
$c = new Circle;
echo $c->r; // FATAL ERROR
```

Аналогічно, публічні методи класу доступні за межами класу.

```
class Circle
{
    public $x, $y, $r;
    public function length() {
        return 2 * 3.14 * $this->r;
    }
}
$c = new Circle;
echo $c->length(); // працює
```

Приватні методи класу доступні лише всередині класу.

```
class Circle
{
    public $x, $y, $r;
    private function length() {
        return 2 * 3.14 * $this->r;
    }
}
$c = new Circle;
echo $c->length(); //FATAL ERROR
```

Але до приватних полів та методів можна звертатись за допомогою публічних методів-посередників.

```
class Circle {
    public $x, $y, $r;
    private function pi() {
        return 3.14;
    }
    public function length() {
        return 2 * $this->pi() * $this->r;
    }
}
$c = new Circle;
echo $c->length(); // OK
```

Якщо ми всередині класу, то до полів та методів звертаємось через `$this->...`

У попередньому прикладі:

```
public function length() {
    return 2 * $this->pi() * $this->r;
}
```

Коли клас описаний, можна створювати об'єкти даного класу. У PHP для цього завжди використовується оператор `new`.

```
$c = new Circle;
```

Звернення до публічних полів та методів створеного об'єкта у PHP відбувається завжди через оператор `->` (бо крапка означає конкатенацію, тобто "склеювання"твох текстових рядків).

```
echo $c->r; // якщо поле публічне
echo $c->length(); // якщо метод публічний
```

Зверніть увагу, що перед назвою поля знак `$` не пишеться!

```
echo $c->$r; // невірно
echo $c->r; // вірно
```

3.1.1 Конструктор класу

Чому дорівнюють значення полів класу в новоствореного об'єкта?

Якщо їх ніде не задано, то всі поля мають значення `NULL`.

```
class Circle
{
    public $x; // NULL
    public $y; // NULL
    public $r; // NULL
}
$c = new Circle;
var_dump($c->r); // NULL
```

Полям можна задати значення за замовчуванням.

```
class Circle
{
    public $x = 0;
    public $y = 0;
    public $r = 1;
}
$c = new Circle;
var_dump($c->r); // 1
```

Можна також створити *конструктор* (функцію, яка сама заповнить усі поля при створенні об'єкта). Конструктор класу в PHP завжди має назву `__construct`.

```
class Circle {
    public $x, $y, $r;
    public function __construct( ) {
        $this->x = 0;
        $this->y = 0;
        $this->r = 1;
    }
}
$c = new Circle; // виклик конструктора
var_dump($c->r); // 1
```

Це був *конструктор за замовчуванням*, який завжди заповнює поля однаковими значеннями.

Конструкторові можна передавати інші значення для заповнення. Тоді він називається *конструктором із параметрами.*

```
class Circle {
    public $x, $y, $r;
    public function __construct($x,$y,$r) {
```

```
        $this->x = $x;
        $this->y = $y;
        $this->r = $r;
    }
}
$c = new Circle(2, -3, 8);
var_dump($c->r); // 8
```

Конструктор не може повертати значення (`return` у ньому не пишеться).

На відміну від інших мов, у PHP кожен клас може мати лише один конструктор. Але це обмеження легко обійти, використовуючи параметри за замовчуванням.

```
class Circle {
    public $x, $y, $r;
    public function __construct($x=0, $y=0, $r=1)
    {
        $this->x = $x;
        $this->y = $y;
        $this->r = $r;
    }
}
$c=new Circle; //конструктор за замовчуванням
$z=new Circle(5,4,7); //конструктор із параметрами
```

Конструктор за замовчуванням можна викликати з дужками або без:

```
$c = new Circle;
$c = new Circle();
```

3.1.2 Деструктор класу

У класі також може бути *деструктор* – функція із назвою `__destruct`, яка викликається автоматично, коли

об'єкт знищується (пам'ять очищається від нього). Деструктор у класі може бути лише один.

```
class Circle {
    public $x, $y, $r;
    public function __destruct() {
        echo "Destructor";
    }
}
$c = new Circle;
// деструктор спрацює автоматично в кінці скрипту
```

Деструктор не може отримувати параметрів і повертати значення (`return` у ньому не пишеться).

3.1.3 Статичні поля і методи

Бувають випадки, коли метод класу працює абсолютно однаково для всіх об'єктів даного класу. Наприклад, метод, який повертає число π. Це число завжди однакове, незалежно від конкретного кола.

```
class Circle {
    public $x, $y, $r;
    public function pi() {
        return 3.14;
    }
    ...
}
$c = new Circle;
echo $c->pi(); // 3.14
```

Незручно щоразу створювати нове коло тільки для того, щоб дізнатися число π. Було б зручніше викликати метод `pi()` напряму, без створення кола.

Для цього існують *статичні методи* класу.

```
class Circle {
    public $x, $y, $r;
    public static function pi() {
        return 3.14;
    }
    ...
}
echo Circle::pi();
// не потрібно створювати коло
```

Статичні методи можна викликати без створення об'єкта класу. Перед статичними методами пишеться ключове слово `static`.

Викликається такий метод наступним чином:

```
назва_класу::назва_методу();
```

Ще зручніший спосіб для зберігання числа π в класі – оголосити його як *статичне поле*. До статичних полів, як і методів, можна звертатися без створення об'єкта класу. Значення статичного поля однакове для всіх об'єктів класу.

```
class Circle {
    public $x, $y, $r;
    public static $pi = 3.14;
    ...
}
echo Circle::$pi;
// зверніть увагу, перед назвою статичного поля
// пишеться $
```

Щоб звернутись до статичного поля чи методу всередині класу, пишеться ключове слово `self`.

```
class Circle {
    public $x, $y, $r;
    public static function pi() {
        return 3.14;
    }
    public function length() {
        return 2 * self::pi() * $this->r;
    }
}

class Circle {
    public $x, $y, $r;
    public static $pi = 3.14;
    public function length() {
        return 2 * self::$pi * $this->r;
    }
}
```

Але значення статичного поля можна змінити.

```
Circle::$pi = 3;
```

Для зберігання числа π краще оголосити його як *константу класу*.

```
class Circle {
    public $x, $y, $r;
    const PI = 3.14;
    ...
}
echo Circle::PI; // 3.14
```

Назви констант прийнято писати ВЕЛИКИМИ літерами.

Значення констант не можна змінювати.

Константи всередині класу оголошуються за допомогою ключового слова `const`.

Не плутайте з константами за межами класу, які оголошуються за допомогою функції `define`.

3.2 Чарівні методи

Поля класу прийнято робити приватними (інкапсуляція). Як же зчитувати/записувати туди дані за межами класу?

Для цього створюють спеціальні методи класу, які служать посередниками між "зовнішнім світом"та приватними полями класу.

Методи, які зчитують значення приватних полів, називають *ґеттерами* (getters).

Методи, які записують значення в приватні поля, називають *сеттерами* (setters).

```
class Circle
{
    private $x, $y, $r;
    public function getX() {
        return $this->x;
    }
    public function setX($value) {
        $this->x = $value;
    }
}

$c = new Circle;
$c->setX(10);
echo $c->getX();
```

Якщо полів у класі багато, незручно створювати для кожного з них ґеттер і сеттер.

На щастя, PHP дає змогу написати один універсальний ґеттер і один універсальний сеттер для всіх полів. У PHP вони також називаються *"чарівними"методами*. Розглянемо їх детальніше.

3.2.1 "Чарівний"ґеттер

```
class Circle {
    private $x, $y, $r;
    public function __get($field) {
        return $this->$field;
    }
}
$c = new Circle;
echo $c->x; // спрацює __get('x')
```

"Чарівний"ґеттер обов'язково отримує 1 параметр. Його можна назвати довільно, наприклад, `$field`. Це параметр типу `string` – назва поля, значення якого потрібно повернути.

Зверніть увагу, що в рядку

```
$this->$field
```

пишеться знак `$` перед `field`. Це означає, що замість `$field` підставиться `x`:

```
$this->x
```

А якщо користувач звертається до неіснуючого поля класу?

```
echo $c->a; // помилка
```

Тому варто спершу перевіряти, чи існує в класі поле з такою назвою.

```
class Circle {
    private $x, $y, $r;
    public function __get($field) {
        if(property_exists($this, $field))
        {
            return $this->$field;
        }
    }
}
```

Функція `property_exists` перевіряє, чи існує в класі поле із вказаною назвою. Повертає `true/false`.

3.2.2 "Чарівний"сеттер

```
class Circle {
    private $x, $y, $r;
    public function __set($field, $value) {
        $this->$field = $value;
    }
}
$c = new Circle;
$c->y = 5; // спрацює __set('y',5)
```

"Чарівний"сеттер обов'язково отримує 2 параметри. Їх можна назвати довільно, наприклад, `$field` і `$value`. Перший параметр – назва поля, другий – значення, яке потрібно туди записати.

Якщо записати значення в неіснуюче поле, воно автоматично створиться і заповниться.

```
$c = new Circle;
$c->a = 5; // створиться нове поле
echo $c->a; // 5
```

3.3 Успадкування класів

Пригадаємо клас, що описує коло на площині:

```
class Circle
{
    public $x, $y, $r;
    public function S()
    {
        return 3.14 * $this->r * $this->r;
    }
}
```

А тепер створимо клас, який описує циліндр. Циліндр характеризується центром та радіусом основи, а також висотою.

```
class Cylinder
{
    public $x, $y, $r, $h;
    public function S()
    {
        return 2*3.14*$this->r*$this->r +
          2*3.14*$this->r*$this->h;
    }
}
```

Нескладно помітити, що клас Циліндр має дещо спільне з класом Коло. Зокрема, основою циліндра є коло. Але, на додачу до кола, у циліндра є нова характеристика – висота.

А центр кола (`$x`, `$y`) і радіус `$r` є в обох класах.

Було б зручніше не писати однакові дані у двох класах, а створити один клас, а потім на основі нього інший.

На щастя, у PHP така можливість передбачена. Вона називається *успадкуванням класів* (inheritance).

Основний клас називають *базовим* або *батьківським*.

```
class Circle {
    public $x, $y, $r;
    public function __construct($x, $y, $r) {
        $this->x = $x;
        $this->y = $y;
        $this->r = $r;
    }
}
```

Клас на основі базового називають *похідним* або *дочірнім*.

```
class Cylinder extends Circle {
    public $h; // $x,$y,$r уже є
    public function __construct($x, $y, $r, $h) {
        $this->x = $x;
        $this->y = $y;
        $this->r = $r;
        $this->h = $h;
    }
}
```

Але навіщо заповнювати всі 4 поля в конструкторі дочірнього класу? Адже перші 3 поля вміє заповнювати конструктор батьківського класу `Circle`. Можна просто викликати батьківський конструктор з дочірнього. Звернутись до батьківського класу можна за допомогою ключового слова `parent`.

```
class Cylinder extends Circle {
    public $h;
```

```
    public function __construct($x, $y, $r, $h) {
        parent::__construct($x, $y, $r);
        $this->h = $h;
    }
}
```

Якщо у батьківському класі є приватні поля чи методи, вони будуть недоступними в дочірньому класі.

```
class Circle {
    private $x, $y, $r;
    ...
}

class Cylinder extends Circle {
    // $x, $y, $r недоступні тут
}
```

Як вихід, можна залишати всі поля батьківського класу публічними. А що, як потрібно приховати поле класу від інших класів, окрім дочірнього? Для цього є ключове слово `protected`.

```
class Circle {
    protected $x, $y, $r;
    ...
}

class Cylinder extends Circle {
    // $x, $y, $r уже доступні тут
}
```

Отже, модифікатори доступу бувають:

- public – поле або метод видимі скрізь

- private – видимі лише всередині поточного класу
- protected – лише всередині поточного і дочірніх класів

Звернімо увагу, як обчислює площу метод `S` у класі `Circle`:

```
class Circle {
    public $x, $y, $r;
    public function S()
    {
        return 3.14 * $this->r * $this->r;
    }
}
```

Очевидно, що площа поверхні циліндра обчислюється інакше. Тож метод `S` у класі `Cylinder`, що успадкований від класу `Circle`, доведеться замінити на інший.

```
class Cylinder extends Circle {
    public $h;
    public function S()
    {
        return 2*3.14*$this->r*$this->r +
          2*3.14*$this->r*$this->h;
    }
}
```

Це називається *поліморфізмом*.

У дочірніх класах можна змінювати поведінку (тіло) успадкованих методів. Або, як кажуть, *перевизначити* метод.

Що ще можна вдосконалити у класі `Cylinder`? Звернімо ще раз увагу на метод `S`, який обчислює площу циліндра.

```
class Cylinder extends Circle {
    public $h;
    public function S()
    {
        return 2*3.14*$this->r*$this->r +
          2*3.14*$this->r*$this->h;
    }
}
```

Площа поверхні циліндра – це площа двох основ плюс площа бічної поверхні. Площа основи – це площа круга. А площу круга вміє обчислювати метод `S` у класі `Circle`. Можна просто його викликати.

```
class Cylinder extends Circle {
    public $h;
    public function S()
    {
        return 2*parent::S() +
          2*3.14*$this->r*$this->h;
    }
}
```

У PHP, як і в інших мовах програмування, клас може мати багатьох нащадків. Але, на відміну від C++, клас може мати лише одного батька. У PHP немає *множинного успадкування* (від кількох класів).

Якщо ми НЕ хочемо, щоб якийсь метод був замінений у дочірніх класах, то перед ним пишемо ключове слово `final`.

```
class A {
    final public function f() { ... }
}
```

```
class B extends A {
    public function f() { ... }
    // fatal error
}
```

Фінальні методи не можна перевизначати.

Бувають також *фінальні класи*. Вони не можуть мати нащадків.

```
final class A {
     ...
}

class B extends A { // fatal error
    ...
}
```

Як гадаєте, при описі яких явищ чи об'єктів реального світу можна застосувати успадкування класів? Фінальні класи?

3.4 Абстрактні класи та інтерфейси

Уявімо комп'ютерну гру жанру стратегія. Гру, в якій гравець керує військами різних типів. Наприклад, танкові війська, авіація, морський флот.

Програмний код, який описує кожен тип війська, знаходиться в окремих класах. Нехай кожен із цих класів створює інший програміст. Програмісти знають, що кожен тип війська повинен уміти робити три речі:

1. атакувати

2. оборонятися

3. відступати

Але, припустимо, кожен програміст називає методи класу по-своєму:

```
class Tanks {
    public function ataka() {...}
    public function oborona() {...}
    public function vidstup() {...}
}
class Aviation {
    public function f1() {...}
    public function f2() {...}
    public function f3() {...}
}
```

Як виглядатиме використання цих класів?

```
$tank = new Tanks();
$avia = new Aviation();
$tank->ataka();
$avia->f1(); // це ж треба запам'ятати?
```

Чому б не назвати методи всіх цих класів однаково, щоб не плутатися?

Наприклад:

- attack
- defend
- escape

Але хто має за цим слідкувати? А якщо хтось помилково назве метод `atack` замість `attack`?

Це може виявитись надто пізно, коли керівник проєкту отримає код від усіх програмістів і спробує його поєднати.

```
$tank = new Tanks();
$avia = new Aviation();
$tank->attack();
$avia->attack(); // помилка: неіснуючий метод
```

Хотілося б, щоб сам інтерпретатор примушував кожного програміста називати методи як слід.

У цьому допоможуть *абстрактні класи*.

Можна створити один базовий клас (наприклад, `Army`), який описує поведінку будь-якого війська в загальному. А всі інші класи будуть його успадковувати.

```
class Army { ... }

class Tanks extends Army { ... }

class Aviation extends Army { ... }

class Navy extends Army { ... }
```

А до чого тут абстрактні класи? І що це таке?

Військо – це досить абстрактне поняття. Кожен тип війська має різну поведінку. Танки атакують по-своєму, зі своєю фізикою, анімацією і звуком, авіація по-своєму і флот по-своєму. Як же описати метод `attack` для абстрактного війська?

Можна просто написати метод без тіла, лише заголовок із ключовим словом `abstract`. Якщо метод класу не має тіла (а лише заголовок), то такий метод називається

абстрактним. Його призначення – нічого не робити, а просто вказати, що дочірні класи зобов'язані описати тіло цього методу.

Клас називається *абстрактним*, якщо він має хоча б один абстрактний метод. Деякі методи абстрактного класу все одно можуть мати тіло.

Перед абстрактними класами у PHP також пишеться `abstract`.

```
abstract class Army {
    abstract public function attack();
    abstract public function defend();
    abstract public function escape();
}

class Tanks extends Army {
    public function attack() {/*тіло*/}
    public function defend() {/*тіло*/}
    public function escape() {/*тіло*/}
}
```

В абстрактному класі `Army` сказано, що всі його нащадки зобов'язані вміти атакувати, оборонятися й відступати. Дочірні класи повинні реалізувати всі успадковані абстрактні методи (описати тіло). Якщо дочірній клас не має тіла хоча б для одного абстрактного методу, то цей клас теж вважається абстрактним. А створювати об'єкти абстрактного класу не можна. Виникає Fatal Error.

```
abstract class Army {
    abstract public function attack();
    abstract public function defend();
    abstract public function escape();
}
```

```
class Tanks extends Army {
    public function atack() {/*помилка в назві*/}
    public function defend() {/*тіло*/}
    public function escape() {/*тіло*/}
}
$t = new Tanks(); // fatal error
```

Дочірній клас `Tanks` повинен містити методи `attack`, `defend` та `escape` разом із тілами. Якщо програміст помилково напише `atack` замість `attack`, це вважається іншим методом. А успадкований метод `attack` досі не має тіла, тож клас `Tanks` також вважається абстрактним. При спробі створити об'єкт абстрактного класу виникає фатальна помилка. У цьому випадку програміст сам зверне увагу, що він невірно назвав якийсь метод. Інші учасники проєкту не повинні будуть витрачати зайвий час на виправлення чужих помилок.

Тож абстрактні класи використовуються лише для успадкування. Наприклад, щоб змусити програмістів створювати в дочірніх класах лише методи із вказаними назвами.

Очевидно, що поля і методи в абстрактному класі можуть бути лише `public` або `protected`. Адже приватні (`private`) методи не успадковуються. Їх не буде видно в дочірніх класах. Доступу до них ніде не буде. Писати їх немає змісту.

Нагадаємо, що в абстрактному класі можуть бути методи з тілом. Наприклад, якщо є метод, який працює ідентично для класів `Tanks`, `Aviation`, `Navy`, його можна винести в батьківський клас `Army`. Усі класи успадкують його автоматично.

Наприклад, усі типи військ належать одній країні:

```
abstract class Army {
```

```
    abstract public function attack();
    abstract public function defend();
    abstract public function escape();
    public function getCountry() {
        return "Ukraine";
    }
}
```

Також нагадаємо, що в PHP, як і в інших мовах програмування, клас може мати багатьох нащадків. Але, на відміну від C++, клас може мати лише одного батька. У PHP немає множинного успадкування (від кількох класів).

У PHP, як і в деяких інших мовах програмування, також є *інтерфейси*. По суті, це те саме, що й абстрактні класи. Але, на відміну від абстрактних класів, там жоден метод не може мати тіла. Тільки заголовки.

```
interface Army {
    public function attack();
    public function defend();
    public function escape();
}
class Tanks implements Army {
    public function attack() {/*тіло*/}
    public function defend() {/*тіло*/}
    public function escape() {/*тіло*/}
}
```

Призначення інтерфейсів таке ж, як і абстрактних класів.

Якщо програміст забуде написати в класі якийсь метод з інтерфейсу, виникне фатальна помилка. Він неодмінно зверне увагу на свою помилку.

Клас може втілювати кілька інтерфейсів одночасно.

```
class Tanks implements Army, Vehicles { ... }
```

Клас може успадковувати інший клас, а також одночасно втілювати один або кілька інтерфейсів.

```
class Tanks extends Army
  implements Vehicles, GroundTroops { ... }
```

Розділ 4

Лабораторні роботи

4.1 Лабораторна робота №1. Створення базової файлової структури сайту

Теми, які необхідно знати для виконання даної роботи:

- Вставлення PHP-коду на вебсторінку
- Підключення зовнішніх файлів

1. Обрати чітку тематику для свого сайту.
2. Встановити вебсервер Apache. Його можна завантажити окремо або у складі пакетів програм XAMPP, WAMPP, Denwer чи OpenServer.
3. Створити вебсторінку засобами HTML та CSS із чіткою структурою - заголовочна частина (хедер), бокове меню (зліва, справа чи обидва) та нижня частина (футер). У меню мають бути два пункти: *Головна* і *Про сайт*.

4. "Розрізати"вебсторінку на окремі файли: `header.php`, `left_menu.php`, `right_menu.php` (якщо є), `footer.php`. Помістити у відповідні файли відповідні частини HTML-коду. У файлі `index.php` має залишитись лише контент (вміст) сторінки. Крім того, у файлі `index.php` має бути PHP-код, який підключає вміст інших файлів. Таким чином, ціла сторінка має "збиратись"по частинах в одне ціле.

4.2 Лабораторна робота №2. Створення єдиної точки входу

Теми, які необхідно знати для виконання даної роботи:

- Оператор if
- Суперглобальний масив $_GET

1. У кореневій директорії створити підпапки `js`, `css` та `img`, куди будуть поміщатись відповідні файли.

2. Створити підкаталог `layout`, помістити в нього усі PHP-скрипти, створені раніше, окрім `index.php`. Створити підкаталог `views` і в ньому два нові файли: `main.php` та `about.php`. У перший файл помістити контент головної сторінки сайту, який раніше знаходився у файлі `index.php`. Під контентом мається на увазі лише HTML-код із текстом сторінки. А PHP-код, який відповідає за підключення хедера та футера, має залишитись у файлі `index.php`. У файл `about.php` помістити HTML-код із коротким описом сайту та його автора.

3. Відтепер доступ до всіх сторінок сайту здійснюється через точку входу – файл `index.php`, залежно від того, чому дорівнює параметр action у суперглобальному масиві `$_GET`. Якщо цей параметр відсутній або невірний, то показувати сторінку `main.php`. Наприклад, за адресою `index.php?action=about` повинна відображатися сторінка `about.php`, за адресою `index.php` – сторінка `main.php`, а за неіснуючою адресою `index.php?action=123` – також сторінка `main.php`. У меню сайту пункти *Головна* і *Про сайт* тепер повинні вказувати на відповідні сторінки.

4. Перевірка параметра action у масиві `$_GET` не повинна виглядати як перелік усіх наявних на сайті сторінок і має займати лише один блок `if`.

4.3 Лабораторна робота №3. Створення сторінки реєстрації користувача

Теми, які необхідно знати для виконання даної роботи:

- Цикли
- Масиви
- Функції для обробки текстових даних
- Суперглобальний масив `$_POST`
- Регулярні вирази

1. У директорії `views` створити файл `registration.php` (на сайті вона буде відкриватися через `index.php?action=registration`) із HTML-кодом форми реєстрації нового користувача. Там мають бути наступні поля:

   - Логін
   - Пароль
   - Повторіть пароль
   - Електронна пошта
   - Ще одне поле, згідно свого варіанту завдання
   - Кнопка "Зареєструватися"

   Кожне поле повинне мати унікальний атрибут `name`. При створенні форми використати теги `<label>`.

2. Коли користувач натисне кнопку *Зареєструватися*, скрипт має здійснити перевірку, чи правильно заповнені усі поля. Обов'язково використати регулярні вирази. Зокрема:

   - Логін – не менше 4 літер, може містити лише латинські та кириличні літери (великі та малі), цифри, нижнє підкреслення та дефіс.
   - Пароль – не менше 7 літер, обов'язково має містити великі та малі літери, а також цифри.
   - Повторіть пароль – значення має співпадати з полем Пароль.
   - Електронна пошта – має бути коректною Email-адресою, наприклад, містити символ @.
   - Власне поле – згідно варіанту завдання.

Якщо якесь із полів заповнене невірно, має відкритися ця сама сторінка, але з повідомленням про те, які саме поля були невірно заповнені і чому. Якщо ж усі поля заповнені вірно, користувач має побачити нову сторінку (наприклад, `index.php?action=registration_successful`) із повідомленням про те, що реєстрація пройшла успішно (форму реєстрації при цьому вже не показувати), а нижче – посилання *На головну*, яке відкриває головну сторінку сайту.

Варіанти завдань:

1. Багаторядкове текстове поле *Про себе*, необов'язкове для заповнення. Якщо заповнене, то повинно фільтруватись на наявність HTML-тегів, наприклад, `<div>Привіт</div>` має стати просто `Привіт`, не виводячи помилки.

2. Текстове поле *Ім'я*, може бути порожнім. Якщо не порожнє, то не має перевищувати 255 символів. Не може містити цифр та розділових знаків, окрім дефісу та апострофа.

3. Дві радіокнопки для вибору статі користувача. Допустимі значення: 0 - жіноча, 1 - чоловіча.

4. Текстове поле *Телефон*, може бути порожнім. Якщо не порожнє, то не має перевищувати 30 символів. Може містити лише цифри, круглі дужки, пробіли, дефіс та символ +. Допустимі формати номерів: +380123456789, (012) 3456789, (012) 34-56-789, +38 (012) 34 56 789.

5. Текстове поле *ПІБ* (прізвище, ім'я, по-батькові), може бути порожнім. Якщо не порожнє, то не має перевищувати 255 символів. Не може містити цифр та

розділових знаків, окрім пробілу, дефісу, крапки та апострофа. Допустимі формати ПІБ: Прізвище Ім'я По-батькові, Ім'я По-батькові Прізвище, Прізвище І.Б., І.Б. Прізвище.

6. Випадний список *Країна*, обов'язковий для заповнення. Перший елемент списку – текст "оберіть країну". Список повинен містити перелік усіх країн світу в алфавітному порядку. Елементи списку повинні зчитуватись циклом із зовнішнього файлу довільного формату. Кожен елемент списку повинен мати вигляд: `<option value="дволітерний_код_країни» Назва_країни</option>`. Перелік дволітерних кодів країн ISO 3166-1 alpha-2 можна знайти в Інтернеті. Допустиме значення поля – дві великі латинські літери.

7. Випадний список *Мова*, обов'язковий для заповнення. Перший елемент списку – текст "оберіть мову". Список повинен містити перелік усіх мов світу в алфавітному порядку. Елементи списку повинні зчитуватись циклом із зовнішнього файлу довільного формату. Кожен елемент списку повинен мати вигляд: `<option value="дволітерний_код_мови» Назва_мови</option>`. Перелік дволітерних кодів мов ISO 639-1 можна знайти в Інтернеті. Допустиме значення поля – дві малі латинські літери.

8. Текстове поле *Адреса*, може бути порожнім. Якщо не порожнє, то не має перевищувати 255 символів. Може містити лише латинські та кириличні літери (великі й малі), цифри, пробіли, крапки, дефіс, дріб, апостроф. Спочатку має вказуватись назва вулиці, а потім після коми номер будинку та номер кварти-

ри через дріб (якщо є). Наприклад:

- вул. Університетська, 28
- просп. Незалежності, 28/18
- пров. Індустріальний, 28А

9. Випадний список *Область*, обов'язковий для заповнення. Перший елемент списку – текст "оберіть область". Список повинен містити перелік усіх областей України в алфавітному порядку. Елементи списку повинні зчитуватись циклом із зовнішнього файлу довільного формату. Кожен елемент списку повинен мати вигляд: `<option value="код_області»Назва_області</option>`. Перелік числових кодів кожної області можна знайти в Інтернеті. Допустиме значення поля – невід'ємне ціле число.

10. Текстове поле *Моя вебсторінка*, може бути порожнім. Якщо не порожнє, то не має перевищувати 255 символів. Допустимі формати:

- http://example.com
- http://www.e-xam_ple.com
- https://example.com.ua
- http://example.com?id=123
- http://example.com?param1=abc&param2=def

11. Текстове поле `Нікнейм` у соціальних мережах чи месенджерах, може бути порожнім. Якщо не порожнє, то не має перевищувати 255 символів. Може містити лише латинські літери (великі та малі), цифри, крапки, дефіс та нижнє підкреслення.

12. Текстове поле `Дата народження`, обов'язкове для заповнення. Не дозволяється використання `<input type="date»` або jQuery UI-віджета datepicker. Допустимі формати дати:

    - 26.10.1985
    - 1985-10-26
    - 10/26/1985

13. Текстове поле *Капча* (Captcha). Значення цього поля не потрібно зберігати у базі даних у наступній лабораторній роботі. Над текстовим полем має бути напис: "Введіть символи, зображені на малюнку а також зображення з літерами і/або цифрами. Зображення повинне містити різні символи при кожному наступному відкритті сторінки. Спосіб формування цього зображення обрати на власний розсуд. Дозволяється використання для цього сторонніх скриптів.

4.4 Лабораторна робота №4. Підключення до бази даних MySQL

Теми, які необхідно знати для виконання даної роботи:

- Функції для роботи з базою даних MySQL
- Функції хешування паролів

1. Створити базу даних MySQL. Її назва має відображати тематику сайту. Це можна зробити у графічному вебсередовищі phpMyAdmin. Воно постачається у складі пакету XAMPP, але за потреби можна завантажити окремо.

2. У новоствореній базі створити таблицю `users`. У ній має бути обов'язкове поле `id` (або `user_id`, `id_user`) – первинний ключ, а також усі інші поля, описані у відповідному варіанті попередньої лабораторної роботи. Типи цих полів повинні відповідати типу значень, яке вони зберігатимуть (ціле число, текст тощо).

3. Вставити у скрипті реєстрації код, який додає нового користувача до таблиці `users`, якщо всі поля заповнені вірно.

4. Пароль користувача має зберігатись у базі даних у захешованому вигляді за допомогою алгоритму Blowfish. Для цього потрібно використати функцію `password_hash`.

4.5 Лабораторна робота №5. Авторизація користувачів

Теми, які необхідно знати для виконання даної роботи:

- Робота з файлами cookies
- Робота із сесією користувача

1. Додати до таблиці `users` у базі даних поле `admin` типу `boolean` (хоча такий тип є у списку, насправді він зберігатиметься як `tinyint(1)`, оскільки в MySQL немає окремого типу `boolean`, це лише псевдонім). Створити двох користувачів: адміністратора (вручну задати полю `admin` значення 1 у БД) та звичайного користувача (зі значенням 0 у полі `admin`).

2. У директорії `views` створити файл `login.php` із HTML-кодом форми авторизації користувача. Там мають бути наступні поля:

   - Логін
   - Пароль
   - Кнопка *Увійти*

3. Коли користувач натисне кнопку *Увійти*, скрипт має здійснити перевірку, чи є користувач із вказаним логіном у базі даних. Якщо є, то чи співпадає хеш введеного паролю із тим хешем, що у базі (для цього використати функцію `password_verify`). Якщо не виконується будь-яка з вищевказаних умов, вивести знову форму авторизації із повідомленням "Невірний логін або пароль". Якщо ж усі дані введені вірно, записати до сесії логін, ID користувача та ознаку, чи він є адміністратором, і перенаправити на головну сторінку сайту (або просто показати посилання на головну).

4. Додати до меню сайту два нові пункти:

   - Увійти – цей пункт мають бачити лише неавторизовані користувачі (гості).
   - Вийти – цей пункт мають бачити лише авторизовані користувачі. Це посилання має вести на нову сторінку `logout`. Ця сторінка видаляє із сесії всі дані про поточного користувача і перенаправлює на головну сторінку (або показує посилання на головну).

4.6 Лабораторна робота №6. Створення CRUD-сторінок

Теми, які необхідно знати для виконання даної роботи:

- Робота з HTML-формами
- Суперглобальні масиви `$_GET` і `$_POST`
- Робота із сесією користувача
- Функції для роботи з базами даних

CRUD – це скорочення від *create, read, update, delete* ("створення, перегляд, редагування, видалення"). У даній лабораторній роботі потрібно запрограмувати ці всі операції згідно тематики свого вебсайту. Наприклад, якщо тематика сайту – автомобілі, то потрібно зробити наступне:

1. Створити нову таблицю у базі даних і назвати її згідно тематики свого сайту (наприклад, `automobiles`). Там обов'язково мають бути такі поля:
    - `visible` (або `published`) типу `boolean`. Це поле міститиме значення 1, якщо інформація про автомобіль опублікована на сайті, інакше 0;
    - `date` типу `timestamp`, яке вказує, коли було додано інформацію про автомобіль. Перевага типу `timestamp` у MySQL в тому, що його заповнювати не потрібно. Воно само підставить поточну дату й час. Для цього при створенні таблиці треба вказати параметр `CURRENT_TIMESTAMP` для поля date у стовпці "За замовчуванням" ("Default");

- `author_id` типу `int`. Воно має бути зовнішнім ключем, яке вказує на поле `id` (чи `user_id`) у таблиці users. Це поле міститиме ID користувача (а не логін), який додав інформацію про цей автомобіль;
- інші поля, які характеризують автомобіль. Наприклад, марка, модель, рік випуску тощо.

2. Створити сторінку з формою додавання інформації про новий автомобіль (наприклад, `index.php?action=create_automobile`). Здійснювати перевірку, чи поля заповнені вірно. Якщо так, то додати запис до таблиці `automobiles`. Додати до меню сайту пункт *Додати автомобіль*. Цей пункт можуть бачити лише авторизовані користувачі. Якщо автомобіль додає адміністратор сайту, то новий запис у таблиці матиме значення поля `visible` 1. Якщо ж не адміністратор, то 0.

3. Створити сторінку зі списком автомобілів (`index.php?action=automobiles`), а також додати до меню пункт *Автомобілі* (видимий для всіх користувачів). На цій сторінці буде показано інформацію про всі автомобілі з таблиці `automobiles`, у яких значення поля `visible` дорівнює 1 (але адміністратор повинен бачити всі автомобілі, разом із неопублікованими). Новіші записи мають бути на початку. Список можна оформити довільним чином – у вигляді таблиці або ж окремих абзаців.

4. Створити сторінку редагування існуючого автомобіля (`index.php?action=update_automobile&id=5`), доступну лише для адміністратора. Скрипт повинен зчитувати параметр `id` з адресного рядка браузера.

Наприклад, id=5 вказує, що потрібно витягнути з БД із таблиці `automobiles` запис, у якого значення поля `id` (або `id_automobile`) дорівнює 5. Якщо у базі даних не існує автомобіля з таким ID, показувати повідомлення, що такої сторінки не існує. Дана сторінка може мати такий же вигляд, як і сторінка додавання нового автомобіля, аналогічно перевіряючи, чи коректно заповнені усі поля форми. Єдина відмінність – коли користувач відкриває сторінку редагування 5-го автомобіля, текстові поля уже мають бути заповнені інформацією про цей автомобіль. Крім того, дати можливість обирати, чи видимий він на сайті (редагування поля `visible` із бази даних).

5. Створити сторінку перегляду існуючого автомобіля (`index.php?action=view_automobile&id=5`), доступну для всіх користувачів. Відмінність від пункту 3 полягає в тому, що на даній сторінці буде лише інформація про один автомобіль, а не всі зразу. Якщо у базі даних не існує автомобіля з таким ID, показати повідомлення, що такої сторінки не існує.

6. Створити сторінку видалення існуючого автомобіля (`index.php?action=delete_automobile&id=5`), доступну лише для адміністратора. Якщо у базі даних не існує автомобіля з таким ID, показати повідомлення, що такої сторінки не існує. Якщо ж існує, то видалити вказаний автомобіль і показати повідомлення, що автомобіль успішно видалено.

7. Сторінки, створені у пунктах 4-6, поки що ніяк не можна переглянути, адже на сайті ніде немає посилання на них. Для цього на сторінці зі списком ав-

томобілів біля кожного автомобіля слід додати кнопки: *Перегляд*, *Редагувати*, *Видалити*. Останні дві буде бачити лише адміністратор. Ці кнопки відкриватимуть відповідні сторінки, але з різним параметром `id`. Наприклад, якщо у базі є два автомобілі з ID 1 та 8, то кнопка *Перегляд* біля першого автомобіля відкриватиме сторінку `index.php?action=view_automobile&id=1`, а біля другого – сторінку `index.php?action=view_automobile&id=8`. При натисненні кнопки *Видалити* спочатку запитувати, чи дійсно користувач хоче видалити інформацію про вказаний автомобіль. Для цього можна використати JavaScript-метод `confirm`.

Зауваження. На сторінках перегляду, редагування та видалення інформації про автомобіль при зчитуванні параметру ID з масиву `$_GET` варто перетворювати його одразу в ціле число. Інакше, якщо користувач вручну введе в адресному рядку браузера замість числа якийсь текст, є ризик виникнення SQL-ін'єкції. Наприклад, якщо у нас є такий SQL-запит

```
$sql = "SELECT * FROM users WHERE id=".$_GET['id'];
mysqli_query($link, $sql);
```

і користувач введе у браузері адресу

```
index.php?action=view&id=105; DROP TABLE users
```

то запит до БД матиме вигляд

```
SELECT * FROM users WHERE id=105; DROP TABLE users
```

А отже, після пошуку користувача з ID=105 виконається ще один запит, який повністю видаляє з БД таблицю `users`.

Якщо ж перетворювати ID одразу в ціле число

```
$id = (int)$_GET['id'];
$sql = "SELECT * FROM users WHERE id=" . $id;
mysqli_query($link, $sql);
```

то у вказаному вище прикладі буде просто 105, а всі решта символи ігноруватимуться. Навіть якщо користувач вкаже в ID не число, а текст, після перетворення в `int` буде число 0. Таким чином усувається загроза вставлення (ін'єкції) зловмисного SQL-запиту.

Указаний вище метод має недолік – він допомагає фільтрувати лише числові дані. А якщо потрібно зчитати з масиву `$_POST` текстові дані, скажімо, марку автомобіля, яку ввів користувач у формі? Там також може бути шкідливий код. Для запобігання цьому в PHP є функція `mysqli_real_escape_string`.

```
$marka = mysqli_real_escape_string($con,
  $_POST['marka']);
$sql = "INSERT INTO automobiles (marka)
  VALUES (" . $marka . ")";
mysqli_query($link, $sql);
```

4.7 Лабораторна робота №7. Додатковий функціонал на сайті

Варіанти завдань:

1. Система коментарів до статей/записів. Додавати коментарі можуть усі зареєстровані користувачі. Редагувати та видаляти їх може лише адміністратор. Створити таблицю `comments` у БД. Список коментарів до кожної статті/запису показується на сторінці її перегляду. Новіші коментарі мають іти зверху.

2. Можливість додавання зображень до статей/записів. Зображення повинні зберігатись у папці `uploads` у кореневому каталозі сайту, а в БД додати поле `image`, у якому міститься назва файлу із зображенням. Здійснювати перевірку, чи завантажений файл є коректним графічним файлом.

3. Кешування записів. Створити підкаталог `cache`. Коли користувач уперше заходить на сайт, то з БД завантажується список статей/записів і зберігається у текстовому файлі у форматі JSON. Коли користувач знову відкриває сайт, то список статей/записів уже зчитується з цього текстового файлу, а не з БД. Таким чином, якщо на сайт заходять тисячі користувачів, ми зменшуємо навантаження на БД, звертаючись до неї тільки раз. Зчитування з файлу відбувається швидше, ніж із БД. Логічно, що при додаванні нової статті/запису, редагуванні чи видаленні існуючої потрібно видаляти текстовий файл із кешем, оскільки користувачі не побачать змін на сайті. І коли користувач знову відкриє сторінку, система згенерує новий кеш-файл на основі оновлених даних із БД.

4. Система голосування за статті/записи ("лайки"). Під кожною статтею/записом має бути кнопка *Подобається*, якщо користувач іще не голосував за цю статтю/запис. Після натискання на неї повинна відкритись ця ж сторінка, а напис кнопки змінитись на *Не подобається*. Праворуч показувати кількість користувачів, які проголосували ("лайків"). Для цього створити у БД таблицю `likes`. Крім первинного ключа, у ній мають бути поля `user_id`, а також `automobile_id` (або інша назва, відповідно до те-

матики сайту). Коли користувач натискає *Подобається*, до цієї таблиці додається інформація про те, який користувач проголосував за яку статтю/запис. Коли він натискає *Не подобається*, то цей запис видаляється із таблиці.

5. Категорії статей/записів. Їх може додавати, змінювати та видаляти лише адміністратор. Список усіх наявних категорій має відображатись у боковому меню сайту (для всіх користувачів). При натисканні на якусь із них будуть показані лише статті/записи із вказаної категорії. При додаванні та редагуванні статті/запису має відображатись випадний список, у якому користувач може вибрати категорію для даної статті/запису. Для цього створити у БД таблицю `categories`, а також додати поле `category_id` до таблиці `automobiles` (чи як вона називається – згідно тематики сайту). Це поле буде зв'язано зовнішнім ключем із таблицею `categories`.

6. Створити сторінки перегляду (`index.php?action=profile`) та редагування свого профілю (`action=edit_profile`), а також відповідний пункт меню на сайті, який бачать лише зареєстровані користувачі. Додати до таблиці users у БД поля `first_name` (ім'я, `varchar 255`), `last_name` (прізвище `varchar 255`) та `birthdate` (дата народження, типу `date`). Ці ж поля мають бути і на сторінці реєстрації. На сторінці редагування свого профілю користувач може змінювати свій пароль. Для цього мають бути три текстові поля: *Старий пароль*, *Новий пароль*, *Повторіть пароль*. Якщо старий пароль невірний, то вивести повідомлення про помилку і не змінювати пароль. Якщо новий пароль порожній, то просто не

змінювати пароль. Логічно, що на сторінці перегляду профілю користувача не можна показувати його пароль.

7. Посторінкова навігація статей/записів. Показувати по 20 на сторінці. Скажімо, у БД є 48 автомобілів. Тоді на першій сторінці будуть перші 20. Знизу будуть кнопки *Перша*, *1*, *2*, *3*, *Остання*. Причому *Перша* і *1* не будуть клікабельними, оскільки відкрита перша сторінка. Коли користувач натисне *2*, то відкриються наступні 20 записів і кнопка *2* не буде клікабельною. На третій сторінці будуть останні 8 автомобілів. Номер сторінки передавати через додатковий параметр `page` в адресі сторінки, наприклад, `index.php?action=automobiles&page=2`. Підказка: щоб витягати лише по 20 записів із БД, потрібно в SQL-запиті використати ключове слово `LIMIT`.

8. Те саме, що у варіанті 7, але завдання виконати із використанням технології AJAX. Нові статті/записи підвантажувати динамічно, без перезавантаження цілої сторінки. При натисканні на сторінку *2* виконується асинхронний запит до сервера. Коли він поверне результат, то JavaScript повинен замінити контент сторінки на новий і зробити кнопку *2* не клікабельною. Створити файл `ajax_pagination.php`, який витягує з БД кожні 20 статей/записів і повертає їх. Порада: для асинхронних запитів зручно використовувати функцію `load` із бібліотеки `jQuery`.

4.8 Лабораторна робота №8. Розгортання сайту на робочому сервері

Процес перенесення коду з локального комп'ютера на робочий сервер та його налаштування називається *розгортанням* проєкту на сервері (deployment).

На сьогоднішній день існує багато компаній, що пропонують послуги з надання серверів для розміщення вебсайтів. Ця послуга називається *хостингом* (hosting), а самі компанії – *хостерами* (hoster). Ці сервери цілодобово під'єднані до Інтернету і мають фіксовану IP-адресу. Залежно від потужності сервера та дискового простору, ціни на хостинг можуть бути різними. Зокрема, є і безкоштовні хостинги. Зазвичай, вони характеризуються невеликом обсягом дискового простору та наявністю реклами після встановлення власного сайту. Проте для виконання даної лабораторної роботи підійде і безкоштовний хостинг.

1. Усі файли проєкту, зроблені в ході попередніх лабораторних робіт, потрібно заархівувати і закинути на сервер через файловий менеджер у браузері. Логін та пароль до сервера надаються після реєстрації. За бажанням, можна використовувати FTP.

2. Базу даних потрібно окремо експортувати з локального комп'ютера. Для цього у середовищі phpMyAdmin слід вибрати свою базу даних і натиснути вкладку *Експортувати*. Потім зберегти її у файл із розширенням `.sql`.

3. У панелі керування хостингу також має бути середовище phpMyAdmin, у якому треба імпортувати

збережений sql-файл. На деяких хостингах нову базу даних потрібно спершу створити не в phpMyAdmin, а через окремий пункт меню в панелі керування.

4. Після реєстрації компанія, що надає хостинг, повинна надати також логін та пароль до MySQL (не плутати з FTP-логіном та паролем). У кожному PHP-файлі, де є підключення до бази даних, потрібно вписати цей логін та пароль. Не зручно, правда? Тому краще винести логін та пароль до БД в окремий файл і підключати його в потрібних місцях.

Це був лише короткий екскурс у те, що може мова програмування PHP. Звісно, зараз використовуються вже готові бібліотеки кодів, класів та методів, які називаються *фреймворками* (англ. framework – каркас, структура). Вони значно спрощують і пришвидшують процес створення вебсайтів, якщо добре розібратися з їхніми можливостями.

Фреймворків для PHP, як і для інших мов програмування (C#, Java, Ruby, Python), є чимало. Усі вони сповна використовують ООП-можливості PHP. Як і в інших мовах, наразі вже рідко хто пише код без використання класів. Ми ж намагалися розглянути основи PHP, зрозуміти принципи роботи з нею. А розуміючи їх, можна потім буде краще осягнути принципи дії фреймворків.