РОЗДІЛ 1
ВСТУП ДО ДИНАМІЧНОГО ВЕБУ

1.1. Динамічний зміст HTML-сторінки

На початку 90-х років минулого століття Тімом Бернерс-Лі запропоновано концепцію HTTP (протокол передачі гіперпосилань – Hyper Text Transfer Protocol), що лежить в основі WWW. Концепція являє собою взаємодією Web-браузера та Web-сервера. Завдання сервера полягає в прийнятті запиту від клієнта і відправці Web-сторінки за цим запитом. Партнером, який взаємодіє з сервером, є клієнт, тому дане поняття застосовується як до браузера, так і до комп'ютера, на якому він працює.

Між клієнтом і сервером може розташовуватися ряд інших програмно-апаратних пристроїв: маршрутизатори, модулі доступу, шлюзи і т.д.

У кожної машини, підключеної до Інтернету, є своя IP-адреса. Браузер звертається до допоміжної Інтернет-служби, так званої служби доменних імен (Domain Name Service (DNS)), для того, щоб знайти пов'язану з сервером IP-адресу, а потім скористатися нею для зв'язку з комп'ютером.

Загальна схема роботи концепції Hyper Text Transfer Protocol (рис. 1.1):

1. Вводиться http://server.com в адресний рядок браузера.
2. Браузер шукає IP-адресу, що відповідає доменному імені server.com.
3. Браузер надсилає запит на головну сторінку server.com.
4. Запит проходить по Інтернету і надходить на Web-сервер server.com.

5. Web-сервер, який отримав запит, шукає Web-сторінку на своєму твердому диску.

6. Сервер витягає Web-сторінку і відправляє її по зворотному маршруту на адресу браузера.

7. Браузер відображає Web-сторінку.

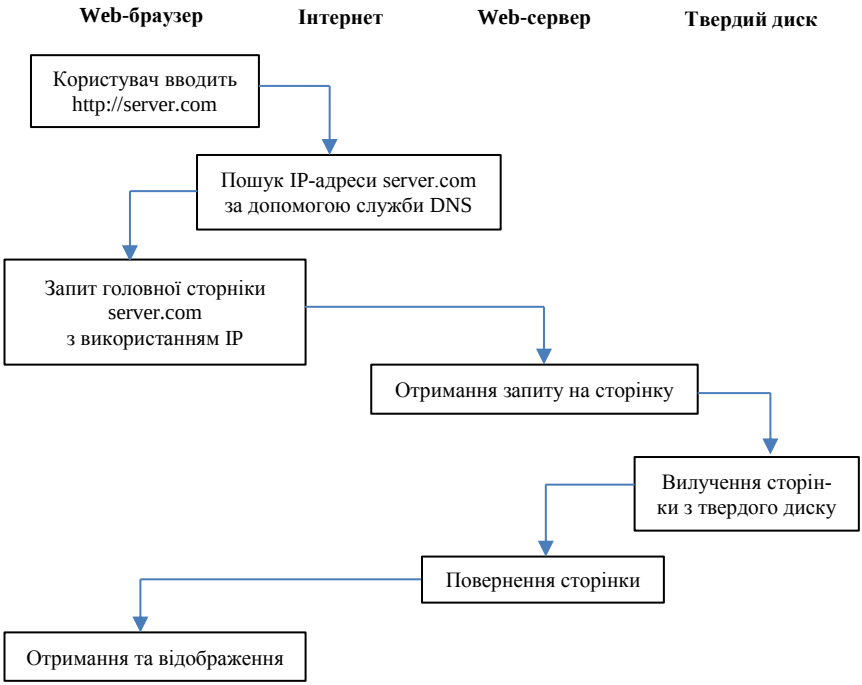

Рисунок 1.1 – Схема роботи концепції HTTP для статичних сторінок

Ця концепція була призначена для відображення статичних Web-сторінок.

При передачі динамічних Web-сторінок процедура складається з більшої кількості дій (рис. 1.2).

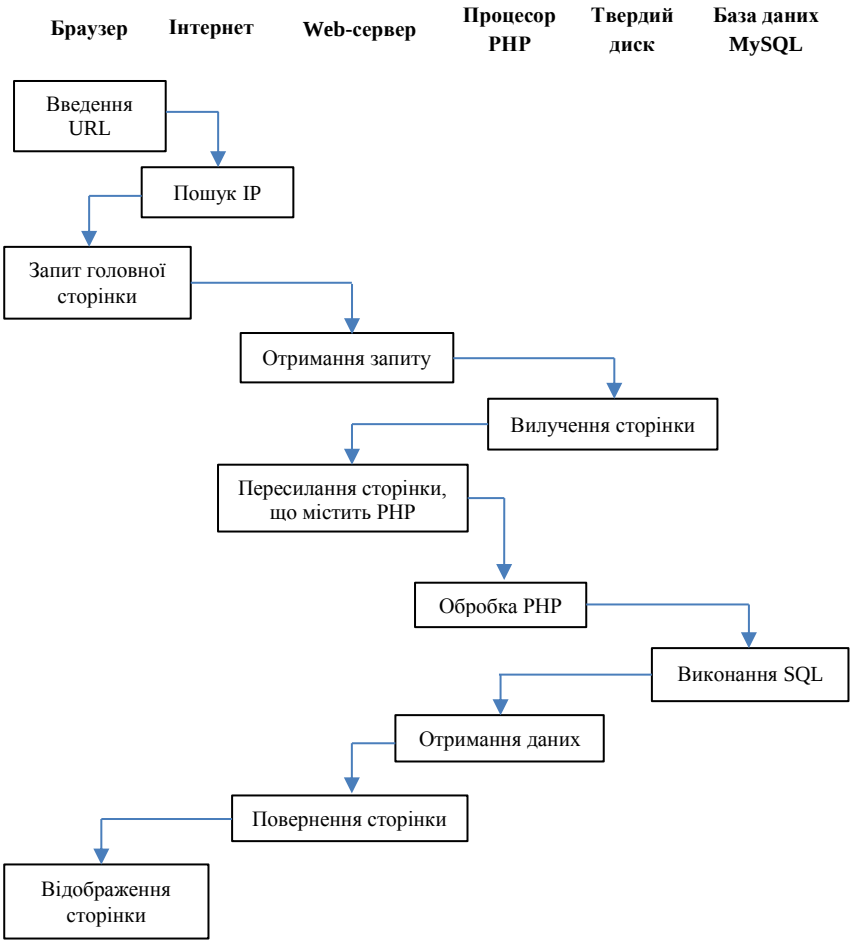

Рисунок 1.2 – Схема роботи концепції HTTP для динамічних сторінок

1. Вводиться http://server.com в адресний рядок браузера.

2. Браузер шукає IP-адресу, що відповідає доменному імені server.com.

3. Браузер надсилає запит на головну сторінку server.com.

4. Запит проходить по Мережі і надходить на Web-сервер server.com.

5. Сервер, який отримав запит, шукає Web-сторінку на своєму твердому диску.

6. Після того як головна сторінка, що включає в себе PHP-сценарії, розміщена в його пам'яті, Web-сервер передає сторінку інтерпретатору PHP.

7. Інтерпретатор PHP виконує PHP-код.

8. Інтерпретатор PHP передає фрагменти коду PHP, що містять MySQL-інструкції, процесору бази даних MySQL.

9. База даних MySQL повертає результати виконання інструкції інтерпретатора PHP.

10. Інтерпретатор PHP повертає серверу результати виконання коду PHP, а також результати, отримані від бази даних MySQL.

11. Web-сервер повертає сторінку за запитом, який відображає цю сторінку на екрані.

Динамічний сайт означає, що, читаючи його, користувач взаємодіє з сайтом, і він змінюється. Відображення динамічних Web-сторінок включає в себе три основні найбільш популярні технології:

– мова створення Web-сценаріїв, які виконуються на стороні сервера, PHP;

– система керування базами даних MySQL;

– JavaScript і каскадні таблиці стилів (Cascading Style Sheets (CSS)).

Так само поряд з PHP, MySQL і JavaScript в динамічній Web-технології присутній Web-сервер, наприклад Apache.

Таким чином, динамічні сторінки об'єднують інформацію, отриману одночасно з декількох джерел, включаючи: Apache, PHP, MySQL, каскадні таблиці стилів і JavaScript.

Переваги використання PHP і MySQL для створення динамічних елементів Web-сайта:

– PHP і MySQL добре взаємодіють разом, з PHP організовується доступ до MySQL;

– PHP і MySQL – програмні продукти з відкритим вихідним кодом, вони можуть бути використані без будь-яких обмежень і без ліцензії.

Для того щоб Web-сервер сприймав сторінку як сторінку з PHP-сценарієм, вона повинна мати розширення .php. Для того щоб фрагмент коду був переданий процесору PHP, він повинен бути у вигляді фрагмента коду:

```
<? php
…
?>
```

Все, що за межами даного тегу, буде передано Web-сторінкою як простий HTML-код.

1.2. Установлювання сервера

Сервер – набір програм, що забезпечують роботу сайта. Щоб сайт запрацював, необхідно всі файли завантажити на вінчестер віддаленого комп'ютера (сервера) і використати спеціальну програму – сервер.

Для того щоб створити Web-застосунок, зазвичай його створюють і налагоджують на локальному комп'ютері, а не на віддаленому Web-сервері, а після цього відправляють його на загальнодоступний Web-сервер. Для цього на локальному комп'ютері необхідно встановити: сервер, PHP і MySQL.

PHP – мова програмування, а MySQL – база даних. Без підтримання Web-сервера у користувачів мережі немає можливості отримати сторінки, що містять програмний код PHP.

Після того як встановити Web-сервер на локальний комп'ютер, можна перетворити свій комп'ютер у Web-сервер:

– можна налагоджувати сторінки, не передаючи їх в Інтернет;

– можна здійснювати доступ до свого комп'ютера іншим користувачам (отримавши виділену IP-адресу і запис DNS).

У теперішній час на ринку представлено більше десятка серверів, які керують роботою сайтів, як відкритих, так і закритих. Наприклад, закритий сервер компанії Microsoft – Internet Information Services (IIS) і відкритий Apache.

Переваги сервера Apache:

– безкоштовний;

– високої якості;

– входить до складу багатьох операційних систем: Windows, Linux, Mac OS X та ін.;

– легко розширюється, має відкритий вихідний код.

Таким чином, для налагодження динамічних Web-сторінок на своєму комп'ютері необхідно встановити: Apache, PHP і MySQL.

Можна встановлювати окремо кожен компонент, а можна установити відразу спрощеним інтегрованим пакетом. Існує декілька інтегрованих пакетів, що дозволяють встановити всі ці програмні продукти в єдиний каталог на локальному комп'ютері. Наприклад, XAMPP (X Apache, MySQL, PHP and Perl) *http://www.apachefriends*.org (рис. 1.3) або Денвер *http://denwer.ru*.

Сервер працює не як застосунок користувача, він не має головного вікна, кнопок меню і т.д., він функціонує цілком у фоновому режимі, а його сервісами користуються інші програми і застосунки – браузери, бази даних, інтерпретатор PHP і т.д. (рис. 1.4).

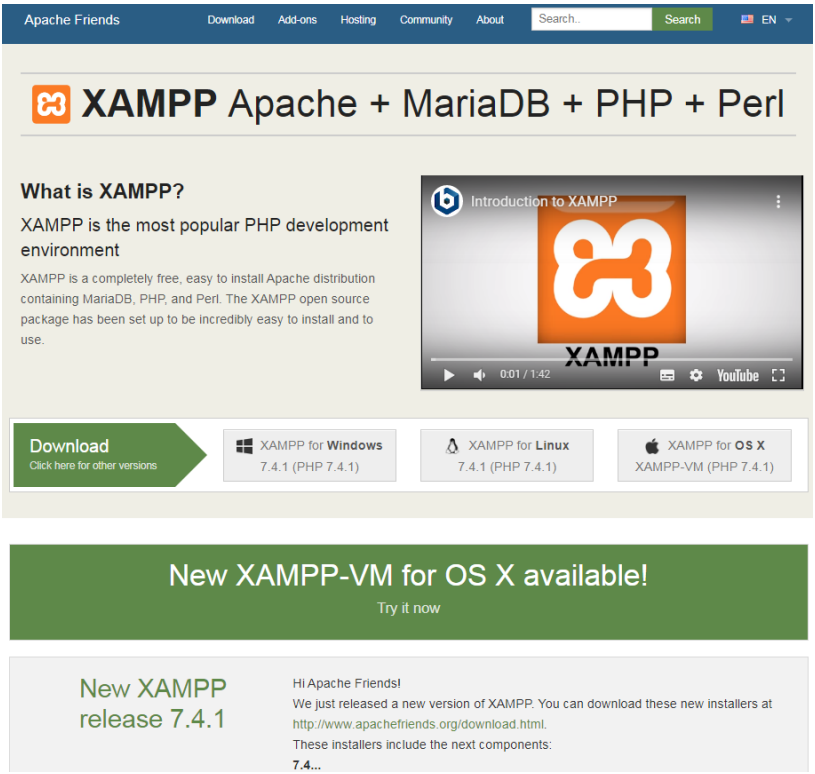

Рисунок 1.3 – Сервер XAMPP

Після закінчення установлювання необхідно перевірити його коректність, ввівши в адресному рядку браузера: *http://localhost/127.0.0.1*, який є адресою локального комп'ютера.

Увесь пакет буде встановлено в директорію *c:\xampp*, а створювані WWW-документи (файли у форматах HTML і PHP, зображення, файли каскадної таблиці стилів) слід зберігати в папці *c:\xampp\htdocs* або в дочірніх папках цієї папки. Доступ до них буде здійснюватися в браузерах за адресами: *http://localhost/test.php* або *http://localhost/lab/test.html.*

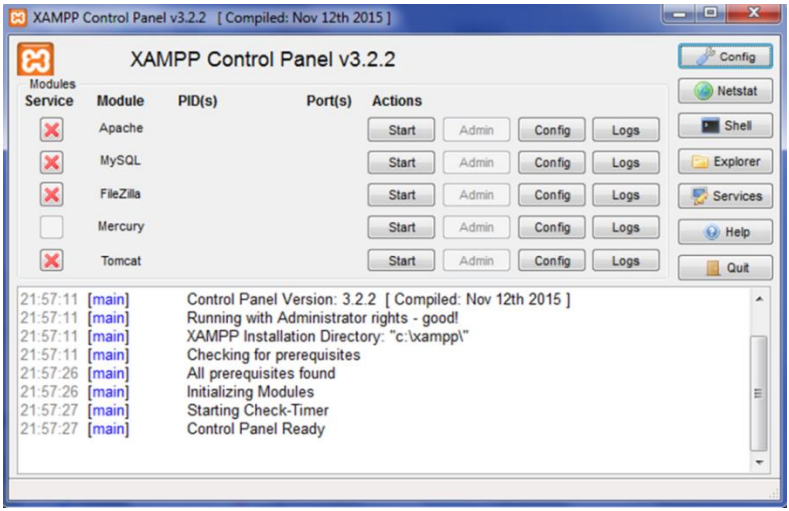

Рисунок 1.4 – Панель керування

Хоча для редагування HTML, PHP і JavaScript підходить будь-який редактор неформатованого тексту, існують спеціальні редактори, призначені для редагування тексту програм, що мають, наприклад, колірне підсвічування синтаксису.

Для PHP-коду можна використовувати редактор Editra (*http://editra.org*). Editra виділяє синтаксис, використовуючи відповідні кольори. Можна помістити курсор за квадратними або фігурними дужками, і Editra підсвітить відповідну парну дужку, що дає можливість визначити зайві або відсутні дужки.

Крім того, можна використовувати інтегровані середовища розробки IDE (Integrated Development Environment). Наприклад, популярними інтегрованими середовищами розробки PHP є phpDesiner або безкоштовне Eclipse PDT (*http://eclipse.org/downloads*).

Контрольні запитання

1. У чому полягає концепція HTTP?

2. Яким чином проводиться обробка динамічної HTML-сторінки?

3. Назвіть основні технології відображення динамічних Web-сторінок.

4. Яке призначення Web-сервера?

5. Назвіть види Web-серверів.

6. Яким чином перевіряється коректність установлювання сервера Apache?

7. В якій директорії слід зберігати створювані WWW-документи?

8. Назвіть переваги використання спеціальних редакторів для редагування PHP-коду.

РОЗДІЛ 2
ОСНОВИ МОВИ PHP

2.1. Можливості мови PHP.

2.2. Основи синтаксису мови PHP.

2.3. Виведення даних.

2.1. Можливості мови PHP

PHP (Hypertext Preprocessor, спочатку Personal Home Page Tools) – скриптова мова програмування, яка застосовується для розробки Web-застосунків (для створення динамічних Web-сайтів).

Особливості мови:

– проста у вивченні;

– призначена для написання скриптів, що виконуються на стороні сервера;

– підтримується майже на всіх відомих платформах, операційних системах і серверах;

– поєднує дві парадигми програмування – об’єктну і процедурну;

– дозволяє створювати зображення, PDF-файли, флеш-ролики, підтримує сучасні бази даних, працює з будь-якими текстовими форматами, включаючи XML; включає ряд корисних функцій для створення застосунків електронної комерції.

Для вбудовування PHP-скрипту в HTML-код можна скористатися тегом, що відкривається, ***<? php*** і тегом, що закривається, ***?>***. Такого виду спеціальні теги дозволяють перемикатися між режимами HTML і PHP.

Коли PHP обробляє файл, він просто передає його текст, поки не зустріне спеціальний тег, який повідомляє йому про необхідність почати інтерпретацію тексту як коду PHP. Потім він виконує весь знайдений код до тегу, який закривається, та говорить інтерпретатору, що далі знову йде просто текст.

2.2. Основи синтаксису мови PHP

У PHP передбачено набір синтаксичних правил. Спрощення синтаксису PHP, наприклад, не вимагає:

– оголошення змінних;

– прототипу функцій.

Синтаксис PHP застосовується тільки до фрагментів коду PHP. Код PHP не змішується з кодом HTML тих документів, в які він упроваджений.

У синтаксисі PHP не має значення кількість розташованих підряд пробільних символів (пробіли, табуляції, символи переходу на новий рядок), тобто один символ пробілу рівнозначний декільком подібним пропускам. Оператор не обов'язково повинен міститися на одному рядку.

PHP іноді враховує регістр символів. Імена змінних є чутливими до регістру. Не враховується регістр в іменах функцій та основних мовних конструкціях (if, then, else, while і т.д.).

До неподільних лексем мови PHP (найменших складових блоків PHP, які відділяються один від одного пробільними символами, круглими і фігурними дужками) належать: числа, рядки, змінні, константи, ключові слова. Вираз – будь-яка комбінація лексем, яка має значення.

Оператором у мові PHP є будь-який вираз, за яким йде крапка з комою (;).

Якщо необхідно використовувати більше одного оператора в тому місці, де повинен бути використаний один оператор, то використовується блок операторів {}.

Мова PHP підтримує синтаксис коментарів мов C++, Java, Perl і сценарії командного інтерпретатора UNIX:

1) /* багаторядковий
коментар */;

2) # коментар у кінці конкретного рядка;

3) // коментар у кінці конкретного рядка.

Змінні і константи в PHP. Всі змінні в мові PHP позначаються префіксом у вигляді знака долара ($), інші символи повинні складатися з букв (великих і малих), цифр і символів підкреслення, першим символом після знака $ не повинна бути цифра.

Значенням змінної є значення, присвоєне змінній в останній за часом операції присвоєння, що здійснюється за допомогою =, +=, -=, *=, /=, .=.

Змінні можуть бути оголошені перед присвоєнням їм значень, але така вимога не є обов'язковою. У мові PHP типи пов'язані зі значеннями, а не зі змінними, тому таке оголошення не потрібне. Першим кроком у використанні змінної є присвоєння їй значення.

Змінні, що використовуються до того, як їм буде присвоєно значеня, мають значення, задані початково. Оскільки змінні є нетипізованими, то початковий тип значення визначається залежно від контексту, в якому використовується ця змінна. Якщо змінна в контексті розглядається як число, то початкове значення у неї 0. Якщо змінна розглядається як рядок, то початкове значення є порожній рядок.

Будь-яка змінна PHP, що не задана у функції, має глобальну область визначення в одному файлі. Після кожного виклику на виконання будь-якої сторінки PHP відбувається присвоєння значень змінним, передбачене в коді сторінки, а після завершення формування сторінки ці змінні знищуються.

Для того щоб зберегти інформацію про значення, змінній при переході від сторінки до сторінки:

– передають інформацію зі сторінки на сторінку за допомогою змінних GET і POST;

– забезпечують постійне зберігання інформації в базі даних;

– зберігають інформацію на твердому диску користувача за допомогою cookie-файлів;

– пов'язують інформацію з сеансом користувача, застосовуючи механізм сеансів PHP.

Змінні, яким присвоюється значення у функції, є локальними по відношенню до цієї функції, а функція не має доступу до глобальних змінних, які визначені за межами цієї функції, навіть, якщо ці змінні визначені в тому ж файлі, що і сама функція. Щоб мати можливість звертатися в функції до глобальних змінних, необхідно застосувати спеціальне оголошення.

Константи PHP мають імена, що не містять префікса $, які складаються зазвичай з великих літер. Константи можуть містити тільки скалярні значення (числа і рядки). Константи мають глобальні області визначення, тому після свого визначення доступні в будь-якому місці сценарію, навіть всередині функції. Після визначення констант їхні значення встановлюються для всієї іншої програми і не можуть бути змінені.

Існують вбудовані константи (наприклад, NULL) й константи користувача. При оголошенні константи використовується функція **define()**, яка вимагає задати ім’я константи і значення цієї константи.

Приклад 2.1

```
define(“STRING_CONSTANT”, “This is my string.”);
define(“NUMERIC_CONSTANT”, 5);
echo STRING_CONST;
echo NUMERIC_CONSTANT;
```

Система типів мови PHP. В усіх мовах програмування передбачена та чи інша система типів, які відображають формат подання даних у комп’ютері на рівні бітів. У PHP:

– тип змінної не потрібно оголошувати заздалегідь;

– вимоги до типів виразів, які використовуються в поєднанні з операціями і функціями, не є занадто жорсткими. У мові передбачені зручні способи автоматичного перетворення типів у разі необхідності;

– у математичних виразах зі змішаними числовими типами здійснюється перетворення до більш загального типу;

– автоматично перетворюються типи параметрів функції.

Наприклад, функція substr(12345, 2, 2) виділяє підрядок символів з рядка, починаючи з заданого символу.

Приклад 2.2

```
$sub=substr(12345, 2, 2);
print ($sub);
// відбувається автоматичне перетворення типів, результат – 34
```

Приклад 2.3

```
$a=2+2* “nonsense”+TRUE;
// результат – числове значення 3
```

У мові PHP передбачено всього вісім типів:

– прості типи: цілі числа (як позитивні, так і від’ємні), числа з рухомою крапкою подвійної точності (записуються з крапкою), логічні значення (TRUE і FALSE), NULL-значення, рядкові значення (послідовності символів в одинарних або подвійних лапках необмеженої довжини);

– складові типи: масиви (іменовані та індексовані колекції інших значень), об’єкти (екземпляри класів);

– ресурси (спеціальні змінні, які зберігають посилання на ресурси, зовнішні по відношенню до інтерпретатора PHP, такі, як з’єднання з базою даних).

Рядкові типи даних:

1. Рядки в одинарних лапках залишаються практично незмінюваними, з двома винятками (\\, \’):

– лапки іншого типу не призводять до розриву рядка, що розміщується в лапках;

– щоб використовувати в рядку апостроф, необхідно перед ним поставити переключення на інший режим обробки, як і перед косою рискою.

Приклад 2.4

```
$str=‘Видавництво\‘Нова книга\’’;
print($str);
// Видавництво ‘Нова книга’
```

Приклад 2.5

```
$path=‘C:\\TEMP\\PHP\\’
print(“Windows path: $path’’);
// Windows path: C:\TEMP\PHP\
```

2. Рядки в подвійних лапках, піддаються попередній обробці інтерпретатором PHP:

– символьні послідовності, що починаються зі зворотної косої риски, замінюються спеціальними символами (до ескейп послідовності додається \$);

– імена змінних (починаються зі знака $) замінюються рядковими поданнями їхніх значень. Цей процес називається інтерполяцією – заміна змінної в рядку її вмістом.

2.3. Виведення даних

Більшість конструкцій PHP виконуються, не виводячи жодних повідомлень на зовнішній пристрій. Для виведення PHP-кодом інформації в програму браузера використовуються ключові слова ***echo*** і ***print*** – основні конструкції мови, які можуть використовуватися з круглими дужками і без дужок:

– використання декількох фактичних параметрів:

Приклад 2.6

```
echo “рядок виведення 1”, “рядок введення 2”;
```

– echo з дужками допускає один фактичний параметр:

Приклад 2.7

```
echo (“рядок виведення”);
```

– у команді echo можна використовувати оператор точки для з’єднання рядків і змінних:

Приклад 2.8

```
<?php
$fname = “Іванов”;
$lname = “Петро”;
echo “Ім’я корисувача” . $fname . “ ”. $lname;
?>
```

– команда print в будь-якому форматі приймає тільки один фактичний параметр. На відміну від команди echo команда print повертає значення, яке дозволяє дізнатися, чи успішно виконаний оператор. Якщо значення, що повертається оператором print, дорівнює 1, то виведення даних виконано успішно, якщо 0, то виведення даних не вдалось;

– оператори echo і print не створюють автоматичні переноси рядків, необхідно використовувати тег <p> або
 для створення абзаців або переносів рядків.

Контрольні запитання

1. Яким чином вбудовується PHP-скрипт у HTML-код?
2. Як визначити змінні і константи в PHP?
3. Назвіть основні особливості синтаксису PHP.
4. Які типи даних застосовуються в мові PHP?
5. Які конструкції в PHP використовуються для виведення даних?

РОЗДІЛ 3
РОБОТА З МАСИВАМИ В PHP

3.1. Два типи масивів у PHP

Змінну типу масиву можна використовувати для зберігання множини або послідовності значень. PHP підтримує два види масивів: масиви з числовими індексами та асоціативні масиви.

При роботі з масивами слід виділяти два поняття: елементи та індекси. Елементи – це значення, що зберігаються в масиві. До кожного елемента можна звернутися за його унікальним індексом. Значенням індексу може бути число (масив з числовим індексом) або рядок (асоціативні масиви), але це значення має бути унікальним. Масив можна уявити як таблицю, яка містить два стовпці. Перший стовпець унікально ідентифікує рядок таблиці (тобто є ключем), а в другому зберігаються значення.

Елементами масивів можуть бути будь-які значення: рядки, числа або масиви.

Поле ключа має бути скалярним (числом, рядком, TRUE або FALSE) та унікальним для кожного елемента. Якщо призначати новому елементу ключ, який вже визначений для іншого елемента, то нове значення просто замінить старе. Чим коротше строкове значення ключів масиву, тим швидше і простіше буде працювати програма.

Щоб створити масив, потрібно визначити значення його елементів та індексів.

При додаванні нового елемента в масив з числовими індексами PHP автоматично призначить йому перше вільне число, починаючи з 0.

При додаванні елементів в асоціативний масив їм необхідно призначати унікальний строковий індекс. Асоціативні масиви допускають застосування числових і рядкових значень індексу.

3.2. Створення масиву

У мові PHP записати значення в масив можна двома способами.
І спосіб основано на ідентифікаторах масиву.

Приклад 3.1

```
<?php
$weekdays[0] = ‘Monday’;
$weekdays[1] = ‘Tuesday’;
?>
```

Якщо при призначенні індексів елементів пропустити один індекс, то з точки зору PHP це не буде помилкою, але може бути відсутнім одне значення.

Якщо в квадратних дужках не вказувати індекс, то PHP вибере найменший незайнятий числовий індекс.

Приклад 3.2

```
<?php
$weekdays[ ] = ‘Monday’;
$weekdays[ ] = ‘Tuesday’;
?>
```

ІІ спосіб основано на використанні функції array().

Функція ***array()*** дозволяє створювати масиви з одночасним присвоєнням множини значень елементів.

Параметрами функції є ключ / значення. Значення, що повертається, – масив, який можна присвоїти змінній.

Створення масиву з числовими індексами:

Приклад 3.3

```
<?php
$weekdays = array(‘Monday’, ‘Tuesday’, ‘Wednesday’, ‘Thursday’, ‘Friday’, ‘Satuday’, ‘Sunday’);
?>
```

При створенні асоціативного масиву між індексом і значенням встановлюється знак (=>), який є знаком рівності, за яким відразу йде символ більше.

Приклад 3.4

```
<?php
$shapedays = array(‘Phone book’ => ‘Rectangle’,
                   ‘Apple’ => ‘Ball’,
                   ‘Orange’ => ‘Ball’,
                   ‘Notebook’ => ‘Rectangle’);
?>
```

Особливість синтаксису масивів. PHP дозволяє створювати масиви більш ніж з одним виміром. Ця особливість полягає в можливості включати цілий масив до складу іншого масиву.

Масиви, ключі яких подані рядковими значеннями, що містять пробільні символи і знаки пунктуації, повинні бути поміщені у фігурні дужки ({}).

Функція ***is_array()*** дозволяє перевірити, чи є змінна масивом.

Приклад 3.5

```
<?php
$yes=array(‘це’, ‘–’, ‘масив’);
echo is_array($yes) ? ‘Масив’ : ‘Не масив’;
$no=‘це – рядок’;
echo is_array($no) ? ‘Масив’ : ‘Не масив’;
?>
```

3.3. Витяг значень з масиву

Масив у PHP є впорядкованим відображенням. Відображення є типом, який відображає значення в ключі. Змінні масивів складаються з двох частин – індексу і елемента. Індекс масиву, іноді званий ключем масиву, є значенням, що застосовується для ідентифікації або доступу до елементів масиву. Індекс масиву поміщається в квадратні дужки. Більшість масивів використовують числові індекси, які зазвичай починаються з 0 або 1. В PHP асоціативні масиви можуть використовувати рядкові індекси.

Приклад 3.6

```
<?php
$my_array = array(‘red’, ‘green’, ‘blue’);
echo “Перше значення масиву –” . $my_array[0];
echo “Друге значення масиву –” . $my_array[1];
echo “Третє значення масиву –” . $my_array[2];
?>
<HTML>
Перше значення масиву – red
Друге значення масиву – green
Третє значення масиву – blue
</HTML>
```

Приклад 3.7

```
<?php
$members = array(‘First Name’ => ‘John’,
                 ‘LName’ => ‘Smith’,
                 ‘Age’ => 50);
echo “The user\’s first name is” . {$members[‘First Name’]};
echo “The user\’s last name is” . $members[‘LName’];
echo “The user\’s age is” . $members[‘Age’];
?>
<HTML>
The user’s first name is John
The user’s last name is Smith
The user’s age is 50
</HTML>
```

Масив – тип даних, з якими повинні бути визначені операції. Масиви можна складати і порівнювати.

Складають масиви за допомогою стандартного оператора “+”, який об’єднує масиви. Якщо є два масиви, $a і $b, то результатом їх складання (об’єднання) буде масив $c, що складається з елементів $a, до яких праворуч дописані елементи масиву $b.

Якщо зустрічаються ключі, що збігаються, то в масив з результатом включається елемент ключів, що співпадають, з першого масиву, тобто з $a.

Приклад 3.8

```
<?php
$a = array(“і” => “Інформатика”, “м” => “Математика”);
$b = array(“і” => “Історія”, “м” => “Біологія”, “ф” => “Фізика”);
$c = $a + $b;
$d = $b + $a;
print_r ($c);
```

```
/* Отримаємо: array([і] => Інформатика [м] => Математика [ф] => Фізика) */
print_r($d);
// Отримаємо: array([і] => Історія [м] => Біологія [ф] => Фізика)
?>
```

Порівнювати масиви можна, перевіряючи їхню рівність чи нерівність або еквівалентність чи нееквівалентність. Рівність масивів – це збіг усіх пар ключ-значення елементів масивів. Еквівалентність – це рівність значень і ключів елементів і запис елементів обох масивів в одному і тому ж порядку. Рівність значень в PHP позначається символом «= =», а еквівалентність – символом «= = =».

Приклад 3.9

```
<?php
$a = array(“і” => “Інформатика”, “м” => “Математика”);
$b = array(“м” => “Математика”, “і” => “Інформатика”);
if($a = = $b) echo “Масиви рівні та”;
else
echo “Масиви НЕ рівні та”;
if($a = = = $b)
echo “еквівалентні”;
else
echo “НЕ еквівалентні”;
// отримаємо “Масиви рівні та НЕ еквівалентні”
?>
```

Масив, елементами якого є масиви, називається багатовимірним. Кожен набір ключів і значень являє собою вимір. У багатовимірних масивах для кожного виміру є свій набір ключів і значень.

Щоб отримати доступ до другого виміру, використовується друга пара квадратних дужок, в яких вказується другий ключ. Якщо у масиві більше двох вимірів, вказуються ключі для кожного з них.

PHP допускає в асоціативних масивах скорочену форму запису, в якій елемент масиву присвоюється змінній, ім'я якої збігається зі значенням ключа. При цьому імена ключів повинні бути записані латинськими літерами, без пробілів, що відповідає вимогам, які висуваються до імен змінних. Для цього використовується функція ***extract()***, єдиним параметром якої є масив.

Якщо значення ключа збіглося з ім'ям якоїсь змінної, то значення цієї змінної буде заміщено значенням елемента масиву. Щоб цього не сталося, функція extract() може автоматично додавати символ підкреслення в початок імен змінних, запобігаючи можливому заміщенню змінної, що вже використовувалася. Символом підкреслення в створюваних іменах змінних автоматично відокремлюються ім'я ключа і префікс:

extract($ім'я_масиву, EXTR_PREFIX_ALL, "префікс").

Функція ***compact()*** діє протилежно функції extract(). Вона приймає у вигляді окремих параметрів змінні, масиви або їхню комбінацію і створює асоціативний масив, ключами якого слугують імена змінних, а значеннями елементів – значення змінних.

Для налагодження програми, яка працює з масивами, часто використовується вбудована функція ***var_dump($ім'я_масиву***), яка дозволяє вивести весь масив за одне звернення.

Конструкція ***list()*** використовується для того, щоб присвоїти списку змінних значення за одну операцію.

Приклад 3.10

```
<?php
list($day, $month, $year) = full_age("07", "08", "1974");
echo "Вам $year років, $month місяців і $day днів";
?>
```

Для присвоєння змінним значень елементів масиву конструкція list() використовується в такий спосіб.

Приклад 3.11

```
<?php
$arr = array(“first”, “second”);
list($a, $b) = $arr;
// змінній $a присвоюється перше значення масиву, $b – друге
echo $a, “ ”, $b;
// виводимо рядок “first second”
?>
```

3.4. Функції для роботи з масивами

Крім функції array(), система PHP включає безліч інших функцій для роботи з масивами. Великий список доступний на Web-сайті PHP. Деякі з функцій, що найчастіше використовуються:

– count() – функція використовується для підрахунку числа елементів в масиві;

– sort() – функція використовується для сортування елементів існуючого масиву. Функція має такий синтаксис: ***sort(масив[, прапори])*** і сортує масив за зростанням. Ця функція видаляє всі існуючі в масиві ключі, замінюючи їх числовими індексами, що відповідають новому порядку елементів. У разі успішного завершення роботи вона повертає true, інакше – false.

Як додатковий аргумент може використовуватися одна з таких констант:

- SORT_REGULAR – порівнювати елементи масиву звичайним чином;
- SORT_NUMERIC – порівнювати елементи масиву як числа;
- SORT_STRING – порівнювати елементи масиву як рядки.

Якщо потрібно зберігати індекси елементів масиву після сортування, то потрібно використовувати функцію ***asort(масив[, прапори])***.

Якщо необхідно впорядкувати масив в зворотному порядку, тобто від найбільшого значення до найменшого, то можна задіяти функцію ***rsort(масив[, прапори])***.

Якщо при цьому потрібно ще й зберегти значення ключів, то слід використовувати функцію ***arsort(масив[, прапори])***.

Синтаксис у цих функцій абсолютно такий самий, як у функції sort(). Відповідно і значення прапорів можуть бути такими самими, як у sort(): SORT_REGULAR, SORT_NUMERIC, SORT_STRING.

Приклад 3.12

```
<?php
$books = array(“Пушкін” => “Руслан і Людмила”,
                “Толстой” => “Війна і мир”,
                “Лермонтов” => “Герой нашого часу”);
asort($books);
// сортуємо масив, зберігаючи значення ключів
print_r($books);
rsort($books);
// сортуємо масив у зворотному порядку, ключі будуть замінені
print_r($books);
?>
```

Функції ***ksort()*** і ***krsort()*** сортують масив значень ключів. Синтаксис цих функцій аналогічний синтаксису функції sort();

– shuffle() – функція використовується для випадкового перемішування елементів у заданому масиві;

– sizeof() – функція є синонімом (аліасом) функції count();

– array_slice($array_name, offset, length) – функція використовується для вилучення частини існуючого масиву. *$array_name* є ім'ям масиву, що розрізається, *offset* вказує позицію, де буде починатися розріз, *length* вказує число елементів, яке буде вирізано з масиву;

– array_merge($array_name, $array_name) – функція використовується для об'єднання або злиття двох або більшої кількості існуючих масивів. Імена масивів розділяються комами.

Наступний код показує, як застосовується кожна з функцій для роботи з масивами.

Приклад 3.13

```
<HTML>
<HEAD>
<TITLE>Сторінка Web </TITLE>
</HEAD>
<BODY>
<?php
$numbers = array(50, 20, 18, 30, 10, 7);
$colors = array('red', 'blue', 'green');
//створено два масиви
$array_size = sizeof($numbers);
// визначається розмір масиву $numbers – 6
sort($numbers);
/* сортуються елементи масиву $numbers – повертає array(7, 10, 18, 20, 30, 50) */
shuffle($numbers);
// випадковим чином перемішуються елементи масиву $numbers
$merged_array = array_merge($numbers, $colors);
/* $merged_array повертає array(7, 10, 18, 20, 30, 50, 'red', 'blue', 'green') */
```

```
$slice = array_slice($numbers, 2, 2);
// вирізаються номери 18 і 20 з сортованого масиву $numbers
// $slice містить array(18, 20)
?>
</BODY>
</HTML>
```

Контрольні запитання

1. Охарактеризуйте два типи масивів у PHP.
2. Які два способи застосовуються для створення масивів?
3. Укажіть особливості синтаксису масивів, ключі яких подані рядковими значеннями.
4. Назвіть функцію, що дозволяє перевірити, чи є змінна масивом.
5. Як вилучити значення з масиву?
6. Яким чином отримати доступ до другого виміру в багатовимірному масиві?
7. Які операції можна проводити з масивами?
8. Для чого застосовуються функції extract() та compact()?
9. Укажіть основні функції роботи з масивами.
10. Які функції використовуються для сортування ключів та елементів масиву?

РОЗДІЛ 4
ВИКОРИСТАННЯ ФУНКЦІЙ PHP

4.1. Поняття функції

Функція – блок програмного коду, який приймає деякі значення, обробляє їх і виконує певні дії. Для того щоб отримати можливість виклику функції, її потрібно визначити, тобто присвоїти фрагменту коду ім'я. Після чого цей фрагмент коду можна використовувати, звертаючись до нього за ім'ям. Імена функцій не чутливі до регістрів букв.

Виклик функції в PHP здійснюється таким чином: *function_name(вираз_1, вираз_2, …, вираз_n),* вирази в дужках називаються фактичними параметрами, їхня кількість може бути від нуля і вище.

При виявленні виклику функції інтерпретатором PHP спочатку здійснюється обчислення кожного виразу, заданого як фактичний параметр, а потім отримані значення використовуються як вхідні дані для функції.

Приклади допустимих викликів вбудованих функцій:

– sqrt(9) – корінь квадратний;

– rand(10, 10 + 10) – випадкове число від 10 до 20.

Результат значення, що повертається функцією, може належати до будь-якого типу, тому якщо необхідно повернути з функції відразу декілька значень, то зазвичай використовують як значення, що повертається, масив.

Вбудовані функції PHP можуть викликатися з будь-якого сценарію PHP. Мова PHP містить сотні вбудованих функцій. Повний список вбудованих функцій міститься на сайті *http://www.php.net*.

4.2. Визначення власних функцій PHP

Для того щоб створити (визначити) власну функцію, використовують ключове слово *function*:

function ім'я_функції ([параметри функції])

{програмний код return}.

Працюючи з функціями PHP:

– типи даних для параметрів не визначаються;

– параметри функції – це змінні мови, перед назвами кожного повинен стояти знак $;

– не вказується, чи повинна функція повертати якесь значення, а також тип цього значення;

– визначення функції допускається в будь-якому місці програми:

Приклад 4.1

```
<?php
function fact($n)
{
if($n==0) return 1;
else return $n*fact($n–1);
}
echo fact(4). “<br>”;
echo fact(50);
?>
```

– після ключового слова *return* має йти коректний php-вираз, але може і не бути значення, що повертається:

Приклад 4.2

```
<?php
function mass($nn)
{
$mm=$nn*$nn;
$yy=$nn*$nn*$nn;
return array($nn, $mm, $yy);
}
$array1=mass(4);
echo “Перший елемент $array1[0],
другий елемент= $array1[1],
третій елемент = $array1[2]”;
?>
```

До версії PHP4 викликати функцію було можна тільки після її визначення, починаючи з версії PHP4, можна викликати функцію до її визначення (винятком є умовні функції, які визначаються всередині умовних операторів або інших функцій).

Приклад 4.3

```
<?php
$make= true;
Save_info(“Василь”, “Іванов”, “вибрав курс з PHP”);
if($make)
{
function Make_even()
{
echo “Хочу вивчати C++ <br>”;
//умовне оголошення функції
}
}
```

```
Make_even();
function Save_info($first, $last, $message)
{
echo “$message <br>”;
echo “Имя: ”. $first. “ ”. $last. “<br>”;
}
Save_info(“Федір”, “Федіров”, “А я вибрав Python”);
?>
```

Починаючи з PHP4, можна створювати функції зі змінним числом аргументів, тобто створюючи функцію, невідомо зі скількома аргументами її викличуть.

Для написання такої функції використовуються спеціальні вбудовані функції.

Функція ***func_num_args()*** повертає число аргументів, що передано в поточну функцію. Ця функція може використовуватися тільки всередині визначення функції користувача. Якщо вона з’явиться за межами функції, то інтерпретатор видасть попередження:

Приклад 4.4

```
<?php
function DataCheck()
{
$n=func_num_args();
echo “Число аргументів функції $n <br>”;
}
DataCheck();
DataCheck(1, 2, 3);
?>
```

Функція ***func_get_arg(int номер_аргументу)*** повертає аргумент зі списку переданих у функцію аргументів, порядковий номер якого заданий параметром *номер_аргументу*. Аргументи функції лічаться, починаючи з нуля. Як і func_num_args(), ця функція може використовуватися тільки всередині визначення будь-якої функції. *Номер_аргументу* не може перевищувати число аргументів, переданих у функцію. Інакше буде згенеровано попередження, і функція func_get_arg() поверне false:

Приклад 4.5

```
echo “Третій агрумент =”. func_get_arg(2);
```

Функція ***func_get_args()*** повертає масив, що складається зі списку аргументів, які передано функції. Кожен елемент масиву відповідає аргументу, який передано функції. Якщо функція використовується за межами визначення функції користувача, то генерується попередження.

Приклад 4.6

```
<?php
function DataCheck()
{
$n=func_num_args();
$args=func_get_args();
for ($i=0; $i<$n; $i++)
{
echo “Значення аргументів функції DataCheck(): <br>”;
echo “Номер аргументу i = $i” . “Аргумент [i] =$args[$i]”;
}
}
DataCheck(10, 22, 55);
?>
```

4.3. Область визначення функції

Функція повинна бути визначена один і тільки один раз у тому сценарії, в якому вона використовується. Область визначення імен функції розглядається як глобальна, тому функція, яка визначена в сценарії, доступна в будь-якому місці цього сценарію. Але рекомендується визначати функцію до того, як вона буде викликана в коді.

Дуже часто виникає необхідність використовувати один і той же набір функцій у всьому наборі сторінок Web-сайта. У цьому випадку функції зазвичай поміщають в окремий файл. Можливість підключення файлів у PHP забезпечують чотири функції:

– include;

– require;

– include_once;

– require_once.

Всі ці функції можуть приймати як параметр ім'я локального файла. Файлам, що підключаються, можна присвоювати будь-які імена, але завжди з розширенням .php.

Дужки після конструкцій include, reduire, include_once, re-quire_once є необов'язковими. Дані конструкції вставляють вміст зазначених файлів у ту точку програми, де стоїть дана конструкція.

Функція *include* дозволяє підключати до сценарію інші сценарії. Приклад підключення до сценарію файла add.php:

Приклад 4.7

```
<?php
include('add.php');
echo add(2, 2);
?>
```

Файл add.php, що знаходиться в тому ж каталозі:

```
<?php
function add($x, $y){
```

```
return $x+$y;
}
?>
```

Функція *include_once* працює так само, як і функція include, тільки вона не включає файл з тим же ім'ям, якщо цей файл вже включено.

Функція *require* працює аналогічно функції include, тільки якщо не вдалося знайти потрібний файл, конструкція include викликає виведення попереджуючого повідомлення, але обробка сценарію триває, а конструкція require в разі відсутності файла викликає непоправну помилку і припиняє виконання сценарію.

Глобальні і статичні змінні. Щоб мати можливість звертатися у функції до глобальних змінних (всі змінні, які оголошені за межами функції, є глобальними), необхідно всередині функції оголосити їх як глобальні. Для цього слід перелічити їх після ключового слова *global*. При цьому користувач повідомляє інтерпретатору PHP інформацію про те, що ім'я змінної у функції позначає те саме, що воно означає в контексті за межами функції.

Приклад 4.8

```
<?php
$a=1;
function Test_g(){
global $a;
$a = $a*2;
echo ‘в результаті роботи функції $a=’. $a;
}
echo ‘за межами функції $a=’. $a; // $a=1
Test_g(); // $a=2
echo ‘за межами функції $a=’. $a; // $a=2
Test_g(); // $a=4
?>
```

Щоб використовувати змінні тільки всередині функції, при цьому зберігаючи їхні значення і після виходу з функції, потрібно оголосити ці змінні як статичні. Статичні змінні видно тільки всередині функції, і вони не втрачають свого значення, якщо виконання програми виходить за межі функції. Тобто статичні змінні зберігають своє значення від одного виклику однієї і тієї ж функції до іншого. Оголошення таких змінних проводиться за допомогою ключового слова *static*.

Приклад 4.9

```
<?php
function Test_s()
{
static $a = 1;
$a = $a*2;
echo $a;
}
Test_s();
// виводить 2
echo $a;
// нічого не виведе, оскільки $a доступна тільки всередині функції
Test_s();
/* всередині функції $a = 2, тому результатом роботи функції буде число 4*/
?>
```

4.4. Функції для роботи з рядками

Рядок у PHP може бути визначено 3 різними способами:

– за допомогою одинарних лапок;

– за допомогою подвійних лапок;

– heredoc-синтаксисом.

Для визначення рядка за допомогою *heredoc-синтаксису* рядок повинен починатися з символу <<<, після якого йде ідентифікатор, далі рядок, який закінчується тим же ідентифікатором. Ідентифікатор, яким закінчується рядок, повинен починатися в першому стовпці рядка. Ідентифікатор повинен відповідати тим же правилам іменування, що і всі інші мітки в PHP: містити тільки буквені і цифрові символи, знак підкреслення і починатися з цифри або знака підкреслення.

Heredoc-текст поводиться так само, як і рядок у подвійних лапках, тому непотрібно їх обробляти за допомогою esc-послідовності. Змінні всередині heredoc теж обробляються.

Приклад 4.10

```
<?php
$name = ‘Василь’;
echo <<<EOD
Мене звати “$name”.
EOD;
// виводить: Мене звати “Василь”.
?>
```

Для того щоб визначити, чи входить підрядок до складу рядка, використовується функція ***strpos(вихідний_рядок, підрядок_для_пошуку[, з_якого_символу_шукати])***.

Функція повертає позицію появи шуканого підрядка у вихідному рядку або повертає логічне false, якщо входження не знайдено. Додатковий аргумент дозволяє задавати символ, починаючи з якого буде проводитися пошук. Крім логічного false, ця функція може повертати також інші значення, які приводяться до false (наприклад, 0 або “ ”).

Приклад 4.11

```
<?php
$str = “Ідея наносити дані”;
$pos = strpos($str, “дані”);
if ($pos !== false)
echo “Шуканий рядок зустрічається в позиції з номером $pos”;
else
echo “Шуканий рядок не знайдено”;
?>
```

Рекомендується для порівняння використовувати оператор еквівалентності (= = =), який порівнює не тільки значення, але і типи. Оператор еквівалентності = = = здійснює неявне перетворення типів.

Якщо один з операндів логічного оператора може трактуватися як число, то обидва операнди трактуються як числа:

Приклад 4.12

```
$a = 10;
$b = “10”;
if ($a == $b)
echo “a і b рівні”;
// виводить: a і b рівні
```

Якщо значення параметра *підрядок_для_пошуку* не є рядком, то воно перетвориться у цілий тип і розглядається як ASCII-код символу:

Приклад 4.13

```
$pos = strpos($str, 228);
// інтерпретатор буде вважати, що шукається символ “д”
```

Щоб отримати ASCII-код будь-якого символу в PHP, можна скористатися функцією ***int ord(string)***.

Функція, зворотна за змістом ord, ***string chr(код_символу)*** повертає рядок з одного символу, код якого заданий аргументом.

Функція ***strpos()*** шукає першу появу рядка в заданому рядку. Зворотна їй функція ***strrpos(вихідний_рядок, символ_для_пошуку)*** дозволяє знайти позицію останньої появи в рядку символу для пошуку.

Функція ***strstr(вихідний_рядок, підрядок_для_пошуку)*** знаходить першу появу шуканого підрядка і повертає підрядок, починаючи з цього шуканого рядка до кінця заданого рядка. Якщо *підрядок_для_пошуку* не знайдено, то функція поверне false. Якщо *підрядок_для_пошуку* не належить рядковому типу даних, то він переводиться в ціле число і розглядається як код символу. Функція чутлива до регістру.

Функція strstr() повністю ідентична функції strchr().

Функція ***substr(вихідний_рядок, позиція_початкового_символу[, довжина])*** повертає частину початкового рядка довжиною, заданою параметром *довжина*, починаючи з символу, зазначеного параметром *позиція_початкового_символу*. Позиція, з якої починається підрядок, що виділяється, може бути як позитивним цілим числом (відлік ведеться від початку рядка), так і негативним (відлік елементів проводиться з кінця рядка). Якщо параметр *довжина* опущено, то substr() повертає підрядок від зазначеного символу і до кінця заданого рядка. Довжина підрядка теж може бути задана негативним числом (вказане число символів відкидається з кінця рядка):

Приклад 4.14

```
$pure_str = substr($word, 3, –4);
/* виділяємо підрядок, починаючи з 3-го символу, не включаючи 4 символи з кінця рядка */
```

Якщо потрібно отримати один конкретний символ з рядка, знаючи його порядковий номер, можна обійтися без функції substr(). Можна скористатися простішим синтаксисом, записуючи номер символу у фігурних дужках після імені змінної:

Приклад 4.15

```
$text = “Вітаю!”;
echo $text{1};
/* виводить символ “і”, нумерація символів рядка починається з нуля */
```

Функція ***strlen(string)*** визначає довжину рядка.

Функція ***str_replace(шукане_значення, значення_для_заміни, об’єкт)*** використовується для заміни входження підрядка. Функція шукає в розглянутому об’єкті шукане значення і замінює його на значення для заміни. Починаючи з PHP 4.0.5, будь-який аргумент цієї функції може бути масивом. Якщо об’єкт, в якому проводиться пошук і заміна, є масивом, то ці дії виконуються для кожного елемента масиву і в результаті повертається новий масив.

Приклад 4.16

```
<?php
$greeting = array(“Привіт”, “Привіт всім!”, “Привіт, милий!”);
$new_greet = str_replace(“Привіт”, “Добрий ранок”, $greeting);
// робить заміну
print($new_greet);
/* отримаємо: Array(
[0]=> Добрий ранок
[1]=> Добрий ранок всім!
[2]=> Добрий ранок, милий!) */
?>
```

Якщо шукане значення і значення для заміни – масиви, то береться по одному значенню з кожного масиву і проводиться їх пошук і заміна в об’єкті. Якщо значень для заміни менше, ніж значень для пошуку, то як нові значення використовується порожній рядок.

Функція ***substr_replace(заданий_рядок, рядок_для_заміни, позиція_початкового_символу[, довжина])*** поєднує в собі властивість двох функцій str_replace() і substr(). Функція замінює частину рядка рядком, призначеним для заміни. Замінюється та частина підрядка, яка починається з позиції, зазначеної параметром *позиція_початкового_символу*. За допомогою додаткового аргументу *довжина* можна обмежити число символів, що заміняються.

Як і у випадку з функцією substr(), аргументи *позиція_початкового_символу* і *довжина* можуть бути негативними:

– якщо позиція початкового символу негативна, то заміна відбувається, починаючи з цієї позиції щодо кінця рядка;

– негативна довжина задає, скільки символів від кінця рядка не повинно бути замінено;

– якщо довжина не вказується, то заміна відбувається до кінця рядка.

Функція ***explode(роздільник, вихідний_рядок[, максимальна_кількість_елементів])*** ділить *вихідний_рядок* на підрядки, кожний з яких відділений від сусіднього за допомогою зазначеного *роздільника*, і повертає масив отриманих рядків. Якщо задано додатковий параметр *максимальна_кількість_елементів*, то число елементів у масиві буде не більше цього параметра, в останній елемент записується решта рядка. Якщо як *роздільник* указано порожній рядок (“”), то функція explode() поверне false. Якщо символу *роздільника* в заданому рядку немає, то повертається масив з одним елементом у вигляді початкового рядка.

Приклад 4.17

```
<?php
$pizza  = “piece1 piece2 piece3 piece4 piece5 piece6”;
$pieces = explode(“ ”, $pizza);
echo $pieces[0]; // piece1
echo $pieces[1]; // piece2
?>
```

Протилежність функції explode() – функція ***implode(string glue, array pieces)***, яка об'єднує елементи масиву в рядок, повертає рядок, отриманий об'єднанням рядкових подань елементів масиву *pieces*, зі вставкою рядка *glue* між сусідніми елементами.

Приклад 4.18

```
<?php
$data = array("Іванов", "Іван", "Іванович");
$str = implode(" ", $data);
echo $str;
?>
```

У функції implode() існує псевдонім – функція ***join()***, тобто ці дві функції відрізняються лише іменами.

4.5. Регулярні вирази в PHP

Для роботи з рядками в PHP використовуються регулярні вирази.

Регулярні вирази – технологія, яка дозволяє створити шаблон і здійснити пошук даних, що відповідають цьому шаблону, в заданому тексті, поданому у вигляді рядка.

Основна перевага регулярних виразів над функціями полягає в можливості організації більш гнучкого пошуку, тобто можливості знайти те, про що є приблизне уявлення, але немає точного знання, наприклад, семизначний номер телефону.

Зазвичай за допомогою регулярних виразів виконуються три дії:

– перевірка наявності відповідного шаблону підрядка;

– пошук і видача користувачу відповідних шаблону підрядків;

– заміна відповідних шаблону підрядків.

PHP підтримує два стандарти регулярних виразів:

– POSIX (Portable Operating System Interface for Unix) вважається більш застарілим і повільним;

– PCRE (Perl Compatible Regular Expressions) – бібліотека, яка реалізує роботу регулярних виразів у стилі Perl.

Регулярний вираз являє собою рядок, що складається:

– з шаблону;

– спеціального символу роздільника (це можуть бути символи “/”, “|”, “{”, “!” тощо). Обмежувачем можуть виступати довільні символи, крім букв і цифр, і зворотного слешу “\”. Починаючи з PHP 4.0.4 як обмежувач, доступні комбінації: (), {}, [] і <>;

– модифікатора, що впливає на спосіб обробки регулярного виразу.

У шаблон можуть бути включені звичайні символи і метасимволи, символи, які мають спеціальні значення:

1) метасимвол “\” – зворотний слеш має кілька значень:

– змінює тип символу, наступного за ним, на протилежний, тобто якщо це був звичайний символ, то він може перетворитися в метасимвол, якщо це був метасимвол, то він втрачає своє спеціальне значення і стає звичайним символом (це потрібно для того, щоб вставляти в текст спеціальні символи як звичайні).

Наприклад, символ “d” в звичайному режимі не має ніяких спеціальних значень, але “\d” – метасимвол, що означає “будь-яка цифра”. Символ “.” в звичайному режимі означає “будь-який одиничний символ”, а “\.” означає просто крапку;

– кодування недрукованих символів, таких, як:

\n – символ переведення рядка;

\e – символ escape;

\t – символ табуляції;

\xhh – символ в шістнадцятковому коді, наприклад “\x41” є буква “A” і т.д.;

– позначення символьних типів, що генеруються (тобто позначення класу можливих значень), таких, як:

\d – будь-яка десяткова цифра (0–9);

\D – будь-який символ, який не є десятковою цифрою;

\s – будь-який пустий символ (пробіл або табуляція);

\S – будь-який символ, який не є пустим;

\w – символ, який використовується для написання Perl-слів (це літери, цифри і символ підкреслення), так званий “словниковий символ”;

\W – несловниковий символ (всі символи, крім визначених “\w”).

Наприклад, /\d\d\d plus \d is \w\w\w/ – шаблон, що позначає три будь-які цифри, слово “plus”, будь-яка цифра, слово “is”, потім слово з трьох словникових символів;

2) метасимвол “[ ]” – квадратні дужки застосовуються для опису підмножин і всередині регулярного виразу розглядаються як один символ, який може приймати значення, перераховані всередині цих дужок. Якщо першим символом всередині дужок є (^), то значеннями символьного класу можуть бути тільки символи, не перераховані всередині дужок.

Розрізняють дві множини метасимволів: ті, що розпізнаються в будь-якому місці шаблону, за винятком середини квадратних дужок, і ті, що розпізнаються всередині квадратних дужок (табл. 4.1).

Таблиця 4.1 – Метасимволи, які розпізнаються всередині квадратних дужок

Метасимвол	Значення
\	Перехідний символ
^	Заперечення класу, але тільки якщо це перший символ (наприклад, “^\d” задає все, крім цифр)
-	Задає діапазон символів (наприклад, “0–9” задає всі цифри, “A–Z” – всі латинські букви)
]	Обчислює символьний клас

1. Символьний клас **[абвгд]** задає один з символів “а”, “б”, “в”, “г”, “д”, а клас **[^абвгд]** задає будь-який символ, крім “а, б, в, г, д”.

2. Якщо написати **[2бул] ки]**, то цей вираз інтерпретується як один із символів “2”, “б”, “у”, “л”, за яким йде рядок “ки]”, тому що перша квадратна дужка, що закривається, закінчує визначення символьного класу. Тобто цей регулярний вираз співпаде з одним з рядків “2ки] ”, “бки] ”, “уки] ”, “лки]”.

3. За допомогою регулярного виразу **[0-9 А-Я а-я]** можна задати будь-яку букву або цифру.

4. Регулярний вираз **/\d\d/m** можна зіставити з наступними підрядками: “11”, “22”, “33”.

Метасимволи, які розпізнаються за межами квадратних дужок, можна розділити на групи в такий спосіб (табл. 4.2):

– ті, що визначають положення шуканого тексту в рядку, пов’язані з підвиразами, що обмежують символьний клас;

– квантифікатори;

– перерахування альтернатив.

Таблиця 4.2 – Метасимволи, які розпізнаються за межами квадратних дужок

Метасимвол	Значення
\	Перехідний символ
^	Оголошує початок об’єкта (або рядка в багаторядковому режимі). Наприклад, “^abc” – рядок починається з “abc”
$	Маркер кінця рядка (або рядка в багаторядковому режимі). Наприклад, “abc $” – рядок закінчується на “abc”
.	Відповідає будь-якому символу, крім символу переведення рядка (“\n”)
[	Починає визначення символьного класу
]	Закінчує визначення символьного класу
|	Розділяє перерахування альтернативних варіантів
(	Починає підшаблон (регулярний підвираз)
)	Закінчує підшаблон

Продовження таблиці 4.2

Метасимвол	Значення
?	Квантифікатор мінімізації (попередній символ зустрічається 0 або 1 раз)
*	Квантифікатор (попередній символ зустрічається 0 або більше разів)
+	Квантифікатор (попередній символ зустрічається 1 або більше разів)
{	Починає мінімальний / максимальний квантифікатор, указує кількість повторень (інтервал повторень {min, max}). Наприклад, “\w{2,3}” – два або три словникових символи
}	Закінчує мінімальний / максимальний квантифікатор

Використання квантифікаторів PHP. Повторення описуються за допомогою так званих квантифікаторів (метасимволів, які задають кількісні відношення). Існує два типи квантифікаторів:

– загальні (задаються за допомогою фігурних дужок);

– скорочені (історично сформовані скорочення найбільш поширених квантифікаторів).

Квантифікатори можуть йти за будь-яким з перерахованих елементів:

– одиничний символ (можливий в комбінації зі зворотним слешем);

– метасимвол “.”;

– символьний клас;

– зворотне посилання;

– підшаблон.

Загальні квантифікатори задають мінімальне і максимальне число дозволених повторень елемента; ці два числа розділені комою та поміщені у фігурні дужки. Числа не повинні перевищувати 65 536, і перше число повинно бути менше або дорівнюти другому. Якщо другий параметр відсутній, але кома є, то повторень може бути скільки завгодно:

1) **x {1, 3}**. Цьому шаблону відповідають рядки: “x”, “xx”, “xxx”.

2) **[aeuoi] {2,}** означає, що будь-який із символів “a”, “e”, “u”, “o”, “i” в рядку може повторюватися два і більше разів.

Три скорочені квантифікатори, які найбільш часто зустрічаються, мають такі позначення:

1) * еквівалентно {0,} – нуль і більше повторень;

2) + еквівалентно {1,} – одне і більше повторень;

3) ? еквівалентно {0,1} – нуль або одне повторення.

“Жадібність” квантифікаторів. Початково всі квантифікатори “жадібні”, вони намагаються захопити якомога більше повторень елемента.

Квантифікаторам у регулярних виразах відповідає максимально довгий рядок з можливих: якщо вказати, що символ повинен повторюватися один і більше разів (наприклад, за допомогою *), збіг відбудеться з рядком, що містить найбільшу кількість повторень зазначеного символу.

Наприклад, це може створити проблеми при спробі виділити коментарі в програмі на мові C або PHP. Коментарі в C і PHP записуються між символами /* і */, всередині яких теж можуть зустрічатися символи * і /. І спроба виявити C-коментарі за допомогою шаблону **/*.**/** в рядку:

/ перший коментар */*

не коментар

/ другий коментар */*

не увінчається успіхом через “жадібність” елемента **“.*”** (буде знайдено також рядок “не коментар”).

Наприклад, шаблон (<.*>), що описує, на перший погляд, теги HTML, насправді буде виділяти більші фрагменти в документі. Наприклад, рядок вигляду:

* <i>Регулярні вирази<i> – зручний інструмент для пошуку в рядках *

Для розв’язання цієї проблеми можна використовувати два підходи.

1. У регулярному виразі враховуються символи, що не відповідають бажаному зразку (наприклад, <[^>] *> для випадку, що описаний вище).

2. Визначення квантифікатора як нежадібного (ледачого). Більшість реалізацій дозволяють це зробити, додавши після нього знак питання. Тоді квантифікатор спробує захопити якомога менше число повторень елемента, до якого він застосований.

Таким чином утворюються “нежадібні” модифікації квантифікаторів (табл. 4.3).

Таблиця 4.3 – “Нежадібні” модифікації квантифікаторів

Квантифікатор	Опис
*?	“нежадібний” еквівалент *
+?	“нежадібний” еквівалент +
{n,}?	“нежадібний” еквівалент {n,}

1. Шаблон **/*.*?*/** успішно виділяє С-коментарі.

2. За шаблоном (**<.*?>**) будуть знайдені всі теги з розглянутого рядка.

Слід, однак, мати на увазі, що використання “ледачих” квантифікаторів може призвести до ситуації, коли виразу відповідає занадто короткий, зокрема, порожній рядок.

Підвирази (підшаблони). У регулярних виразах підшаблони (або “підвирази”) виділяють, поміщаючи в круглі дужки. Підшаблони можуть бути вкладеними. Виділення частини регулярного виразу у вигляді регулярного підвиразу дозволяє, наприклад, виділити множину альтернатив.

Наприклад, шаблон **жар(ке|птиця)** збігається з одним із слів “жарке”, “жарптиця” і “жар”. Тоді як без дужок це було б “жарке”, “птиця” і порожній рядок.

Модифікатори PCRE. Модифікатори помітно спрощують роботу з регулярними виразами. Модифікатори перераховуються відразу ж після регулярного виразу – наприклад, **/string/i**.

Приклади модифікаторів подані в таблиці 4.4.

Таблиця 4.4 – Модифікатори PCRE

Модифікатор	Опис
m(PCRE_MULTILINE)	Початково рядок, що подається на вхід інтерпретатора, розглядається як такий, що складається з одного рядка. Цей модифікатор включає підтримку багаторядкового режиму
s(PCRE_DOTALL)	За змістом протилежний модифікатору m – при пошуку фрагмент інтерпретується як один рядок, а всі внутрішні символи нового рядка ігноруються. Якщо встановлено цей модифікатор, то метасимвол “.” збігається з будь-яким символом, включаючи символ переведення рядка
i(PCRE_CASELESS)	Пошук виконується без урахування регістру символів
U(PCRE_UNGREEDY)	Модифікатор інвертує “жадібність” квантифікаторів, тобто вони є “нежадібними” та стають “жадібними”, якщо передують символу “?”

4.6. Функції для роботи з регулярними виразами

Основні функції для роботи з Perl-сумісними регулярними виразами:

– ***preg_match(pattern, string[, result, flags])*** виконує перевірку на відповідність регулярному виразу;

– ***preg_match_all(pattern, string, result[, flags])*** шукає в рядку *string* співпадіння з шаблоном *pattern* і поміщає результат у масив *result*:

- *pattern* – шаблон регулярного виразу;
- *string* – рядок, в якому проводиться пошук;

• *result* – масив результатів (нульовий елемент масиву містить відповідність усьому шаблону, перший – першому “захопленому” підшаблону і т.д.);

• *flags* – необов’язковий параметр, що визначає те, як впорядковані результати пошуку.

Функція preg_match() здійснює пошук за шаблоном і повертає кількість знайдених відповідностей. Це може бути 0 (збіги не знайдені) та 1, оскільки preg_match() припиняє свою роботу після першого знайденого збігу. Функція preg_match() повертає false в разі, якщо під час виконання виникли будь-які помилки.

Якщо необхідно знайти або порахувати всі збіги, слід скористатися функцією preg_match_all().

Приклад 4.19

```
<?
// рядок, в якому потрібно щось знайти
$str = “Мій телефонний номер: ”.
“33-22-44. Номер мого редактора: ”.
“222-44-55 та 323-22-33";
//шаблон, за яким шукати
//задає пошук семизначних номерів
$pattern = “/\d{3}-\d{2}-\d{2}/m”;
//функція, що здійснює пошук
$num_match = preg_match_all($pattern, $str, $result);
//виведення результатів пошуку
for($i=0; $i<$num_match; $i++)
echo “Співпадання $i: ”. $result[0][$i]. “<br>”;
?>
```

Рядки, що містять HTML-код. Досить часто необхідно працювати з рядками, що містять HTML-теги. Якщо відобразити такий рядок у браузері за допомогою звичайних функцій відображення даних echo() або print(),

то ми не побачимо самих HTML-тегів, а отримаємо відформатований відповідно до цих тегів рядок. Браузер обробляє всі HTML-теги відповідно до стандарту мови HTML. Іноді нам потрібно бачити безпосередньо рядок, без обробки його браузером. Щоб цього досягти, потрібно перед тим, як виводити, застосувати до нього функцію ***htmlspecialchars(рядок[, стиль лапок[, кодування]])***. Функція переводить спеціальні символи, такі, як “<”, “>”, “&”, “"”, “'” в такі сутності мови HTML, як “<”, “>”, “&”, “"”, “'” відповідно.

Додатковий аргумент *стиль лапок* визначає, як повинні інтерпретуватися подвійні й одинарні лапки. Він може мати одне з трьох значень: ENT_COMPAT, ENT_QUOTES, ENT_NOQUOTES.

Константа ENT_COMPAT означає, що подвійні лапки повинні бути переведені в спецсимволи, а одинарні повинні залишитися без змін.

ENT_QUOTES означає, що повинні конвертуватися і подвійні і одинарні лапки.

ENT_NOQUOTES залишає і ті й інші лапки без змін.

У пункті *кодування* можуть бути задані такі кодування, як UTF-8, ISO-8859-1 та інші.

Приклад 4.20

```
<?php
$new = htmlspecialchars(“<a href=‘mailto:au@gmail.com’>
Написати лист</a>”, ENT_QUOTES);
echo $new;
/* наш рядок перекодується в такий:
&lt;a href=&#039;mailto:au@gmail.com&#039;&gt;
Написати лист &lt;/a&gt; */
У браузері ми побачимо:
<a href=‘mailto:au@gmail.com’>
Написати лист </a>
```

Функція htmlspecialchars() перекодовує тільки спецсимволи, що найбільш часто використовуються. Якщо необхідно конвертувати всі символи в сутності HTML, слід задіяти функцію ***htmlentities()***. Українські літери при використанні цієї функції теж кодуються спеціальними послідовностями. Наприклад, буква “А” замінюється комбінацією “À”. Її синтаксис і принцип дії аналогічний синтаксису і принципу дії htmlspecialchars().

Функції для роботи з рядками. Система PHP містить ряд функцій для роботи з рядками. Наступний список містить деякі з найбільш поширених рядкових функцій:

– strlen(string) визначає довжину рядка *string*;

– ltrim(string) видаляє символи-роздільники на початку рядка *string*;

– rtrim(string) видаляє символи-роздільники в кінці рядка *string*;

– strpbrk(string, char) шукає в рядку *string* символ *char*. Повертає false або рядок, що починається зі знайденого символу;

– strtoupper(string) перетворить рядок *string* у верхній регістр;

– strtolower(string) перетворить рядок *string* у нижній регістр;

– strrev(string) повертає рядок *string* в зворотному порядку;

– eregi(pattern, subject) виконує незалежне від регістру символів порівняння з виразом. У рядку *subject* відбувається пошук регулярного виразу, заданого рядком *pattern*.

Наступний блок коду демонструє, як використовувати рядкові функції PHP.

Приклад 4.21

```
<?php
$string = “Hello World”;
$another_string = “Welcome to PHP”;
echo strlen($string);
echo strtoupper($another_string);
echo strrev($another_string);
echo strpbrk($string, “W”); ?>
```

11
WELCOME TO PHP
PHP ot emocleW
World

Перший рядок виводить довжину рядка “Hello World”, що дорівнює 11. Далі рядок “Welcome to PHP” перетворюється у верхній регістр і виводиться у вікні браузера. Цей рядок використовується також з функцією strrev() для зміни порядку символів рядка на зворотний. Нарешті, в рядку відбувається пошук символу “W”. Оскільки перша поява символу відбувається в тексті “World”, виводиться цей рядок.

Контрольні запитання

1. Поясніть термін «функція».
2. Яким чином викликати функцію в PHP?
3. Як визначити власну функцію?
4. Назвіть функції, що забезпечують підключення до сценарію сценаріїв з інших файлів.
5. Які способи визначення рядка в PHP?
6. Назвіть PHP-функції роботи з рядками.
7. Які функції використовуються для пошуку та заміни входження підрядка?
8. Дайте визначення терміна «регулярний вираз».
9. Для чого використовуються регулярні вирази в PHP?
10. Які квантифікатори застосовуються в регулярних виразах PHP?
11. Поясніть термін «жадібність квантифікаторів».
12. Назвіть основні функції для роботи з регулярними виразами PHP.

РОЗДІЛ 5
ОБРОБКА ДАНИХ ФОРМ У PHP

5.1. Використання HTML-форм для пересилання даних на сервер

Для пересилання даних на сервер у HTML використовуються форми, за допомогою яких дані відправляються для обробки спеціальною програмою на сервері (наприклад, скрипт на PHP). Залежно від введених користувачем даних ця програма може формувати різні Web-сторінки, відправляти запити до бази даних, запускати різні програми і т.п.

Форми застосовуються в програмах на PHP в два етапи (рис. 5.1). На першому етапі відображається форма. З цією метою виконується HTML-розмітка форми дескрипторами для опису відповідних елементів інтерфейсу, призначеного для користувача.

Коли сторінка з формою з'явиться перед користувачем, він повинен ввести інформацію, запитувану у формі, а потім натиснути на кнопку submit, щоб відправити форму назад на Web-сервер. Обробка інформації з переданої форми і становить другий етап застосування форми.

Форми вводяться в документ за допомогою тегу <FORM> </FORM>. Всіма браузерами підтримуються тільки три параметри цього тегу:

– ACTION = "" – єдиний обов'язковий параметр. Його значенням є URL-адреса CGI-програми (Common Gateway Interface), ім'я програми, яка буде обробляти дані форми;

– METHOD = "" повідомляє браузеру метод пересилання даних. Може приймати два значення: GET (початково) і POST;

– ENCTYPE = “” визначає формат кодування при пересиланні даних від браузера до сервера.

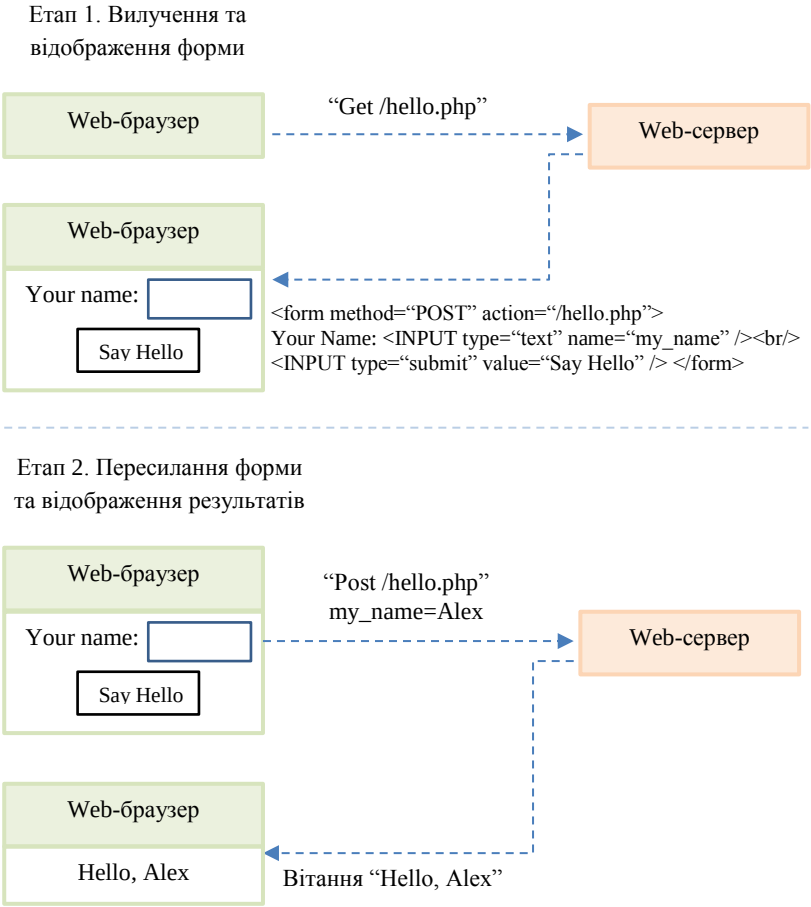

Рисунок 5.1 – Етапи застосування форми

Усередині тегу <FORM> знаходиться одна або кілька команд <INPUT>, <SELECT>, <TEXTAREA>.

Атрибути тегу <INPUT>:

1) TYPE тегу <INPUT> визначає тип елемента, що створюється:

– TYPE = TEXT створює елемент для введення тексту з одного рядка;

– TYPE = RADIO створює елемент-перемикач у складі групи, з якої може бути вибраний тільки один;

– TYPE = SUBMIT створює кнопку пересилання, натиснувши на яку, викликає пересилання вмісту форми на сервер. Для зміни напису на кнопці використовується параметр VALUE.

– TYPE = RESET створює кнопку скидання, натискання якої приводить форму в первинний вигляд, тобто в той стан, в якому вона була до заповнення;

– TYPE = CHECKBOX створює поле для установлення прапорця, надає користувачеві вибір І-АБО.

2) VALUE – початкове значення поля;

3) NAME визначає ім'я поля, що створюється, для ідентифікації даних при пересиланні сервера.

Тег <TEXTAREA> </ TEXTAREA> створює елемент для багаторядкового введення тексту. Цей тег використовує три атрибути: COLS, NAME і ROWS (рис. 5.2).

Пример 5.1

```
<H2>Форма для реєстрації учасників</H2>
<FORM action="1.php" method=POST>
Ім'я <INPUT type=text name="first_name"
value="Введіть ваше ім'я"><br>
Прізвище <INPUT type=text name="last_name"><br>
E-mail <INPUT type=text name="email"><br>
<p>Виберіть курс, який ви б хотіли відвідати:</p>
<input type=radio name="kurs" value="PHP">PHP<br>
<input type=radio name="kurs" value="Lisp">Lisp<br>
<input type=radio name="kurs" value="Perl">Perl<br>
```

<input type=radio name="kurs" value="Unix">Unix
Що ви хочете, щоб ми знали про вас?
<TEXTAREA name="comment" cols=32 rows=5></TEXTAREA>
<INPUT name="confirm" type=checkbox
checked> Підтвердити отримання
<INPUT type=submit value="Надіслати">
<INPUT type=reset value="Скасувати">
</FORM>

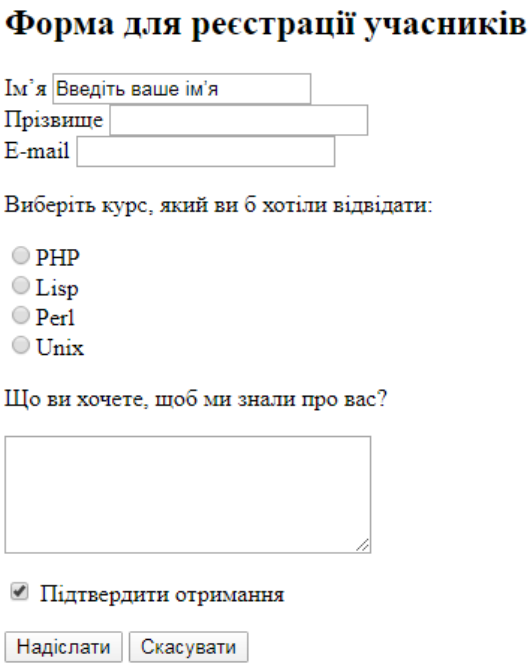

Рисунок 5.2 – Форма для реєстрації учасників

5.2. Методи пересилання даних сервера

При відправленні даних форми за допомогою методу **GET**(method = = get) вміст форми додається до URL після знака питання у вигляді пар ім'я = значення (name = value), об'єднаних за допомогою амперсанда &:

action? name1 = value1 & name2 = value2 & name3 = value3.

Action – це URL-адреса програми, яка повинна обробляти форму (це або програма, задана в атрибуті *action* тегу *form*, або сама поточна програма, якщо цей атрибут опущено).

Імена *name1, name2, name3* відповідають іменам елементів форми, а *value1, value2, value3* – значенням цих елементів.

Для полів введення тексту і пароля (це елементи *input* з атрибутом type = text і type = password), значенням буде те, що введе користувач. Якщо користувач нічого не вводить в таке поле, то в рядку запиту буде присутній елемент *name*, який відповідає імені цього елемента форми.

Для кнопок типу *checkbox* значення *value* визначається атрибутом VALUE у тому випадку, коли кнопка позначена. Не позначені кнопки при складанні рядка запиту ігноруються повністю. Кілька кнопок типу *checkbox* можуть мати один атрибут NAME (і різні VALUE), якщо це необхідно.

Для кнопок типу *radio button* значення *value* визначається атрибутом VALUE в тому випадку, коли кнопка позначена. Не позначені кнопки при складанні рядка запиту ігноруються повністю. Кнопки типу *radio button* призначені для одного з усіх запропонованих варіантів, і тому повинні мати однаковий атрибут NAME і різні атрибути VALUE.

Всі спеціальні символи, включаючи "=" і "&", в іменах або значеннях цих параметрів будуть закодовані. Тому не потрібно використовувати в назвах або значеннях елементів форми ці символи і символи кирилиці в ідентифікаторах.

Якщо в полі для введення ввести який-небудь службовий символ, то він буде переданий в його шістнадцятковому коді, наприклад, символ $ замінюється на "%24".

Недоліки пересилання даних методом GET:

1. Дані, що передаються цим методом, відкриті і доступні для підробки, ім'я користувача і пароль повністю видно на екрані та їх можливо зберігати в пам'ті браузера клієнта.

2. Довжина URL обмежена (в початковій специфікації HTML зазначено, що довжина рядка запиту не повинна перевищувати 255 символів; пізніше це обмеження було знято, але залишилася рекомендація дотримуватися межі в 255 символів).

3. На цей момент метод GET розглядається як застарілий, але використовується часто при налагодженні скриптів.

На теперішній час кращим методом відправлення даних є метод **POST**. Якщо використовується метод POST, то набір даних форми, який включено в тіло форми, відправляється на обробку скрипту як блок окремої інформації.

Основною перевагою методу є можливість передавати до декількох кілобайт інформації, а не кілька сотень символів, як за допомогою методу GET.

Пересилання даних методом POST не є більш безпечним методом, ніж GET. Відвідувач може переглядати дані і змінні, що відправляються як за допомогою методу POST, так і за допомогою методу GET, але при використанні методу POST дані, що передаються, не показуються в адресному рядку.

При відправленні даних на сервер будь-яким методом передаються не тільки самі дані, введені користувачем, а й так звані змінні оточення, що характеризують клієнта, історію його роботи, шляхи до файлів і т.п.:

– REMOTE_ADDR – IP-адреса хоста (комп'ютера), що відправляє запит;

– REMOTE_HOST – ім'я хоста, з якого надіслано запит;

– HTTP_REFERER – адреса сторінки, що посилається на поточний скрипт;

– REQUEST_METHOD – метод, який був використаний при відправленні запиту;

– QUERY_STRING – інформація, яка перебуває в URL після знака питання;

– SCRIPT_NAME – віртуальний шлях до програми, яка повинна виконуватися;

– HTTP_USER_AGENT – інформація про браузер, який використовує клієнт.

5.3. Обробка запитів за допомогою PHP

Усередині PHP-скрипту є кілька способів отримання доступу до даних, переданих клієнтом за протоколом HTTP. Інтерпретатор PHP автоматично присвоює значення змінним на новій сторінці після відправлення набору даних за допомогою методу GET або POST. У зв'язку з таким автоматичним присвоєнням значень змінним необхідно завжди використовувати атрибут NAME для кожного елемента INPUT.

На початку кожного запиту інтерпретатор PHP встановлює ряд автоглобальних масивів, що містять значення будь-яких параметрів, переданих у формі або в URL. Параметри URL і форм, що вилучаються методом GET, розміщуються в масиві **$_GET**, а параметри форм, що передаються методом POST, – в масиві **$_POST**.

Для того щоб дізнатися значення будь-якої змінної оточення, наприклад метод, який використовувався при пересиланні запиту, або IP-адреса комп'ютера, з якої відправлено запит, можна використовувати функцію ***getenv()***. Вона повертає значення змінної оточення, ім'я якої передане їй як параметр:

Приклад 5.2

```
<?php
getenv('REQUEST_METHOD');
// поверне метод
```

```
echo getenv('REMOTE_ADDR');
// виведе IP-адресу користувача, з якої відправлено запит
?>
```

Приклад обробки запиту за допомогою PHP. Потрібно написати форму для реєстрації учасників заочної школи програмування і після реєстрації відправити учаснику повідомлення. Змінимо початковий варіант форми реєстрації таким чином, щоб кожен, хто реєструється, міг вибрати скільки завгодно курсів для відвідування, і не будемо підтверджувати отримання реєстраційної форми.

Приклад 5.3

```
<H2>Форма для реєстрації учасників</H2>
<FORM action="1.php" method=POST>
Ім'я <INPUT type=text name="first_name"
value="Введіть ваше ім'я"><br>
Прізвище <INPUT type=text name="last_name"><br>
E-mail <INPUT type=text name="email"><br>
<p>Виберіть курс, який ви б хотіли відвідати:</p>
<INPUT type=checkbox name="kurs[]" value="PHP">PHP<br>
<INPUT type=checkbox name="kurs[]" value="Lisp">Lisp<br>
<INPUT type=checkbox name="kurs[]" value="Perl">Perl<br>
<INPUT type=checkbox name="kurs[]" value="Unix">Unix<br>
<p>Що ви хочете, щоб ми знали про вас?</p>
<TEXTAREA name="comment" cols=32 rows=5></TEXTAREA><br>
<INPUT type=submit value="Надіслати">
<INPUT type=reset value="Скасувати"></FORM>
```

Спосіб пересилання значень елемента – *checkbox*. Коли ми пишемо в імені елемента *kurs[]*, це означає, що перший зазначений елемент *checkbox* буде записано в перший елемент масиву *kurs*, другий зазначений *checkbox* – у другий елемент масиву і т.д. Можна, звичайно, просто дати

різні імена елементам *checkbox*, але це ускладнить обробку даних, якщо курсів буде багато.

Скрипт, який все це буде розбирати і обробляти, називається *1.php* (форма посилається саме на цей файл, що записано в її атрибуті *action*). За отриманими даними від зареєстрованого користувача скрипт генерує відповідне повідомлення. Якщо користувач обрав якісь курси, то йому виводиться повідомлення про час їх проведення і про лекторів, які їх читають. Якщо нічого не обрано, то виводиться повідомлення про наступні збори заочної школи програмістів.

Приклад 5.4

```
<?php
$times = array(“PHP”=>“14.30”, “Lisp”=>“12.00”,
“Perl”=>“15.00”, “Unix”=>“14.00”);
$lectors = array(“PHP”=>“Василь Васильович”,
“Lisp”=>“Іван Іванович”, “Perl”=>“Петро Петрович”,
“Unix”=>“Семен Семенович”);
define(“SIGN”, “З повагою, адміністрація”);
define(“MEETING_TIME”, “18.00”);
$date = “12 травня”;
$str = “Добрий день, шановний ”. $_POST[“first_name”]. “ ”. $_POST[“last_name”]. “!<br>”;
$str .= “Повідомляємо Вам, що ”;
$kurses = $_POST[“kurs”];
if (!isset($kurses))
{
$event = “наступні збори студентів”;
$str .= “$event відбудяться $date ”. MEETING_TIME . “<br>”;
}
else
{
$event = “обрані Вами лекції відбудяться $date <ul>”;
```

```
$lect = “”;
for ($i=0; $i<count($kurses); $i++)
{
$k = $kurses[$i];
$lect = $lect . “<li>лекція з $k в $times[$k] ”;
$lect .= “(Ваш лектор, $lectors[$k]) ”;
}
$event = $event . $lect . “</ul>”;
$str .= “$event”;
}
$str .= “<br>”. SIGN;
echo $str;
?>
```

Контрольні запитання

1. Яким чином використовується HTML-форма для пересилання даних на сервер?

2. Які три атрибути тегу <FORM> підтримуються усіма Web-браузерами?

3. Назвіть основні переваги та недоліки методів пересилання даних на сервер.

4. Як застосовуються суперглобальні масиви $_POST та $ _GET?

5. Які метадані, що характеризують клієнта, можна передавати на сервер?

РОЗДІЛ 6
РОБОТА З ФАЙЛАМИ В PHP

6.1. Маніпулювання файлами.

6.2. Читання та запис даних з файла / у файл.

6.3. Завантаження файла на сервер.

6.1. Маніпулювання файлами

На мові PHP існують функції для роботи з файлами, що дозволяють виконати такі дії:

– перевірити існування файла;

– створити файл;

– дописати інформацію в кінець файла;

– перейменувати файл;

– видалити файл.

Залежно від типу операційної системи, де працює інтерпретатор PHP, імена файлів можуть бути чутливі до регістру букв або ні: у Windows і Mac OS X імена файлів не чутливі до регістру букв, а в Unix – чутливі.

Функція ***bool file_exists(ім'я_файла_або_директорії)*** використовується для перевірки існування файла, аргументом функції є *ім'я файла*, що перевіряється. Якщо файл існує, повертається значення true, в протилежному випадку – false.

Приклад 6.1

```
<?php
$file_name = “file_exists.php”;
if(file_exists($file_name))
{
echo(“Файл $file_name знайдено”);
}
```

```
else
{
echo("Файл $file_name не знайдено");
}
?>
```

Функції ***is_readable(ім'я_файла), is_writable(ім'я_файла)***, ***is_executable(ім'я_файла)*** дозволяють перевірити можливість читання з файла, запису у файл і виконання файла відповідно.

Для створення файла в PHP не існує спеціальної функції, призначеної саме для створення файла, більшість функцій працюють з уже існуючими файлами у файловій системі сервера. Для створення файла можна скористатися функціями:

– touch(ім'я_файла);

– fopen(ім'я_файла, тип_доступу[, use_include_path]).

Параметр *ім'я_файла* повинен бути рядком, що містить:

– URL-адресу файла в мережі, тоді *ім'я_файла* починається з вказівки протоколу доступу (наприклад, http:// ... або ftp:// ...) і шукає обробник зазначеного в URL протоколу;

– локальне ім'я файла. *Ім'я_файла* не починається з протоколу.

Параметр *тип_доступу* вказує, для чого відкривається дане зображення (табл. 6.1):

1) r – відкриває файл тільки для читання; встановлює вказівник позиції у файлі на початок файла;

2) w – відкриває файл тільки для запису; встановлює вказівник файла на його початок і зрізує файл до нульової довжини. Якщо файл не існує, то намагається створити його;

3) a – відкриває файл тільки для запису; встановлює вказівник файла в його кінець. Якщо файл не існує, то намагається створити його.

Таблиця 6.1 – Значення, що приймаються параметром тип_доступу

Тип доступу	Опис
r	Відкриває файл тільки для читання; встановлює вказівник позиції у файлі на початок файла
r+	Відкриває файл для читання і запису; встановлює вказівник файла на його початок
w	Відкриває файл тільки для запису; встановлює вказівник файла на його початок і зрізує файл до нульової довжини. Якщо файл не існує, то намагається створити його
w+	Відкриває файл для читання і запису; встановлює вказівник файла на його початок і зрізує файл до нульової довжини. Якщо файл не існує, то намагається створити його
a	Відкриває файл тільки для запису; встановлює вказівник файла в його кінець. Якщо файл не існує, то намагається створити його
a+	Відкриває файл для читання і запису; встановлює вказівник файла в його кінець. Якщо файл не існує, то намагається створити його
x	Створює і відкриває файл тільки для запису; розміщує вказівник файла на його початок. Якщо файл вже існує, то fopen() повертає false і генерується попередження. Якщо файл не існує, то робиться спроба створити його. Цей тип доступу підтримується, починаючи з версії PHP 4.3.2, і працює тільки з локальними файлами

Продовження таблиці 6.1

Тип доступу	Опис
x+	Створює і відкриває файл для читання і запису; поміщає вказівник файла на його початок. Якщо файл вже існує, то fopen() повертає false і генерується попередження. Якщо файл не існує, то робиться спроба створити його. Цей тип доступу підтримується, починаючи з версії PHP 4.3.2, і працює тільки з локальними файлами

До типу доступу можна додавати прапор типу файла (бінарний ***b*** або текстовий *t*) (a+b). Для переносності програми на різні платформи рекомендується завжди використовувати прапор *b* при відкритті файлів за допомогою fopen().

Параметр *use_include_path*, який встановлено в значення 1 або true, змушує інтерпретатор шукати зазначений в fopen() файл в *include_path*.

Include_path – директива з файла налаштувань PHP, задає список директорій, в яких можуть знаходитися файли для включення. Крім функції fopen(), вона використовується функціями include() і require().

Якщо відкрити або створити файл за допомогою fopen() не вдається, в цьому випадку PHP генерує попередження, а функція fopen() повертає як результат своєї роботи значення false, в разі успішного відкриття файла функція повертає вказівник на файл. Для заборони подібного попередження використовується символ @:

Приклад 6.2

```
$h = @fopen(“dir/another_file.txt”, “w +”)
```

Закриття з’єднання: ***fclose(покажчик_на_файл)*** повертає true, якщо з’єднання успішно закрито, і false в іншому випадку: fclose($h).

Видалення файла: ***unlink(string ім’я_файла)*** повертає true, якщо з’єднання успішно закрито, і false в іншому випадку.

6.2. Читання і запис даних з файла / у файл

Запис даних у файл, на який вказує *вказівник_на_файл*, відкритий за допомогою функції fopen():

int fwrite(вказівник_на_файл, рядок[, довжина]).

Якщо вказано додатковий аргумент *довжина*, то запис закінчується після того, як записано кількість символів, що дорівнює значенню цього аргументу, або коли буде досягнуто кінця *рядка*.

У результаті своєї роботи функція fwrite() повертає число записаних байтів або false у разі помилки.

Для того щоб прочитати дані з файла, потрібно скористатися однією зі спеціальних функцій: fread(), fgets(), fgetc(), readfile(), file(), file_get_contents():

1) ***string fread(вказівник_на_файл, довжина)*** – читання даних заданої довжини в байтах. Читання здійснюється до тих пір, поки не зустрінеться кінець файла або поки не буде прочитано вказане параметром *довжина* число байтів. У результаті роботи функція повертає рядок зі зчитаною з файла інформацією.

Для визначення довжини файла використовується функція ***filesize(ім'я_файла)***, яку можна використовувати тільки для отримання розміру локальних файлів, у разі помилки функція поверне false:

Приклад 6.3

```
$content = fread($h, filesize("dir\another_file.txt"));
echo "$content";
```

Функція filesize() кешує результати своєї роботи для забезпечення більшої продуктивності. Тобто якщо змінити розмір файла і знову запустити скрипт, то результат буде той самий. В цьому випадку необхідне очищення статичного кеша: ***void clearstatcache(void)***.

За допомогою функції fgets() можна зчитати з файла рядок тексту. Синтаксис цієї функції практично такий самий, як і у fread(), за виключенням того, що довжину рядка, який зчитується, вказувати необов'язково.

2) ***string fgets(вказівник_на_файл[, довжина])*** повертає рядок довжиною (довжина–1) байт з файла, на який вказує *вказівник_на_файл*. Читання закінчується, якщо прочитано «довжина–1» символів або зустрівся символ переведення рядка або кінець файла. У разі помилки функція повертає false.

Використовуючи булеву функцію feof(), яка перевіряє, чи дивиться вказівник позиції файла на кінець файла, можна зчитати всі рядки файла:

Приклад 6.4

```
$hh=fopen("text.txt","r");
while(!feof($hh))
{
$content=fgets($hh);
echo $content. "<br>";
}
```

Якщо можна зчитувати інформацію з файла порядково, то можна зчитувати її посимвольно. Для цього призначена функція fgetc().

3) ***string fgetc(вказівник_на_файл)*** – функція повертає символ з файла, на який посилається *вказівник_на_файл*, і значення, що обчислюється як false, у разі кінця рядка, тобто зчитує інформацію з файла посимвольно.

4) ***int readfile(ім'я_файла[, use_include_path])*** зчитує файл, ім'я якого передано функції як параметр, і виводить його вміст на екран. Повертає число лічених байтів (символів) файла, а в разі помилки – false. Повідомлення про помилку в цій функції можна подавити оператором @:

Приклад 6.5

```
$n= readfile(“dir\another_file.txt”);
echo $n;
```

5) не завжди потрібно виводити вміст файла на екран. Функція file() призначена для зчитування інформації з файла в змінну типу масив: ***array file(ім’я_файла[, use_include_path])***.

Кожен елемент цього масиву подає рядок файла, інформація з якого зчитується. Символ нового рядка теж включається в кожен з елементів масиву:

Приклад 6.6

```
$arr = file(“text.txt”);
foreach($arr as $i => $a)
echo $i. “:”. $a. “<br>”;
```

6) починаючи з версії PHP 4.3, можна зчитувати вміст файла в рядок: ***string file_get_contents(ім’я_файла[, use_include_path])***.

Функція абсолютно ідентична функції file(), тільки повертає вона вміст файла у вигляді рядка. Крім того, вона безпечна для обробки бінарних даних і може зчитувати інформацію з віддалених файлів, якщо це не заборонено налаштуванням сервера.

6.3. Завантаження файла на сервер

Для завантаження файла на сервер необхідно створити HTML-форму (рис. 6.1):

– в тезі <form> повинен бути атрибут *enctype* зі значенням *multipart/form-data*;

– елемент <input> повинен мати тип type = file;

– бажано використовувати у формі приховане поле, яке містить максимально допустимий розмір файла в байтах. При спробі завантажити файл, розмір якого більше зазначеного в цьому полі значення, буде зафіксована помилка.

Приклад 6.7

```
<FORM enctype=“multipart/form-data”
action=“parse.php”  method=“post”>
<INPUT type=“hidden” name=“MAX_FILE_SIZE” value=“30000” />
Завантажити файл: <INPUT type=“file” name=“myfile” /><br>
<INPUT type=“submit” value=“Надіслати файл” />
</FORM>
```

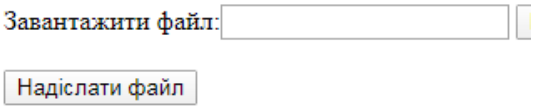

Рисунок 6.1 – HTML-форма

Починаючи з версії PHP 4.1.0, вся інформація про завантажений на сервер файл міститься в глобальному масиві **$_FILES**. Якщо включена директива *register_globals*, то значення переданих змінних доступні просто за їхніми іменами.

Масив $_FILES завжди має такі елементи:

– $_FILES[‘ім’я_елемента_форми’][‘name’] – ім’я, яке мав файл на машині клієнта;

– $_FILES[‘ім’я_елемента_форми’][‘type’] – тип відправленого файла, якщо браузер надав цю інформацію. У нашому прикладі це *text/html*;

– $_FILES[‘ім’я_елемента_форми’][‘size’] – розмір завантаженого файла в байтах;

– $_FILES[‘ім’я_елемента_форми’][‘tmp_name’] – тимчасове ім’я файла, під яким він був збережений на сервері;

– $_FILES[‘ім’я_елемента_форми’][‘error’] – код помилки, що з’явилася при завантаженні.

Ім’я елемента форми, в нашому випадку *myfile*, – це ім’я елемента форми, за допомогою якого було проведено завантаження файла на сервер.

Виділяють п’ять типів помилок при завантаженні в PHP і відповідно $_FILES[‘myfile’][‘error’] може мати п’ять значень:

0 – помилки не відбулося, файл завантажено успішно;

1 – завантажений файл перевищує розмір, встановлений директивою *upload_max_filesize* у файлі налаштувань php.ini;

2 – завантажений файл перевищує розмір, встановлений елементом MAX_FILE_SIZE форми html;

3 – файл був завантажений частково;

4 – файл завантажений не був.

Якщо завантажити з комп’ютера клієнта файл з ім’ям critics.htm розміром 15136 байт, то команда print_r($_FILES) виведе на екран таке:

Приклад 6.8

```
Array ([myfile] => Array(
[name] => critics.html
[type] => text/html
[tmp_name] => C:\WINDOWS\TEMP\php49F.tmp
[error] => 0
[size] => 15136))
```

Завантажені файли зберігаються в тимчасовій директорії сервера, якщо інша директорія не зазначена за допомогою опції *upload_tmp_dir* у файлі налаштувань php.ini.

Перемістити завантажений файл у потрібну директорію можна за допомогою функції ***bool move_uploaded_file(тимчасове_ім’я_файла, місце_призначення)***. Функція перевіряє, чи дійсно файл, позначений рядком *тимчасове_ім’я_файла*, був завантажений через механізм заванта-

ження HTTP методом POST. Якщо це так, то файл переміщується у файл, заданий параметром *місце_призначення* (цей параметр містить як шлях до нової директорії для зберігання, так і нове ім'я файла).

Якщо *тимчасове_ім'я_файла* задає неправильний завантажений файл, то ніяких дій проведено не буде, і move_uploaded_file() поверне false. Те ж саме станеться, якщо файл з якихось причин не може бути переміщений. В цьому випадку інтерпретатор виведе відповідне попередження. Якщо файл, заданий параметром *місце_призначення*, існує, то функція move_uploaded_file() перезапише його.

Контрольні запитання

1. Які дії виконують функції роботи з файлами?
2. Назвіть функції перевірки існування файла.
3. Яким чином створюється файл, встановлюється з'єднання з файлом і виконується закриття файла в PHP?
4. Укажіть можливі значення параметра *тип_доступу* функції fopen().
5. Які функції використовуються для читання і запису даних з файла / в файл в PHP?
6. Які атрибути тегу <FORM> застосовуються для завантаження файла через форму?
7. Яке призначення глобального масиву $_FILES?
8. Укажіть існуючі типи помилок при завантаженні файлів.

РОЗДІЛ 7
ВЗАЄМОДІЯ PHP З БАЗОЮ ДАНИХ

7.1. СУБД MySQL.

7.2. Взаємодія PHP та MySQL.

7.3. Виконання запитів MySQL.

7.4. Вибірка наборів даних.

7.5. Реалізація засобів контролю помилок.

7.6. Відображення результатів запитів SQL в таблицях HTML.

7.7. Способи застосування численних запитів або складних операторів виведення.

7.8. Додавання даних до таблиці.

7.1. СУБД MySQL

База даних – це сукупність пов'язаних даних, організованих за певними правилами, що передбачають загальні принципи опису, зберігання і маніпулювання, незалежно від прикладних програм. Звернення до баз даних здійснюється за допомогою системи керування базами даних (СУБД).

MySQL – це реляційна система керування базами даних, тобто дані в ній зберігаються у вигляді логічно пов'язаних між собою таблиць, доступ до яких здійснюється за допомогою мови запитів SQL.

Працювати з MySQL можна:

– у текстовому режимі. Команда вводиться в командний рядок, і результат виводиться на екран;

– у графічному режимі. Існує популярний візуальний інтерфейс (написаний на PHP) для роботи з цією СУБД: phpMyAdmin, що дозволяє значно спростити роботу з базами даних в MySQL.

Для того щоб мати можливість працювати з базою даних, повинен бути обліковий запис користувача і пароль, що дозволяє обмежити коло

користувачів, що володіють правом доступу до таблиць на сервері. Якщо MySQL встановлено на своєму комп'ютері, то можна використовувати пароль, який призначений при установленні. Відразу після установлення пароль користувача *root* – порожній рядок.

PhpMyAdmin – застосунок на PHP, який надає Web-інтерфейс для роботи з MySQL.

Для того щоб почати працювати в цьому застосунку, необхідно запустити XAMPP Control Panel, потім у командному рядку набрати http://localhost/xampp/, на сторінці в групі Tools вибрати phpMyAdmin (рис. 7.1).

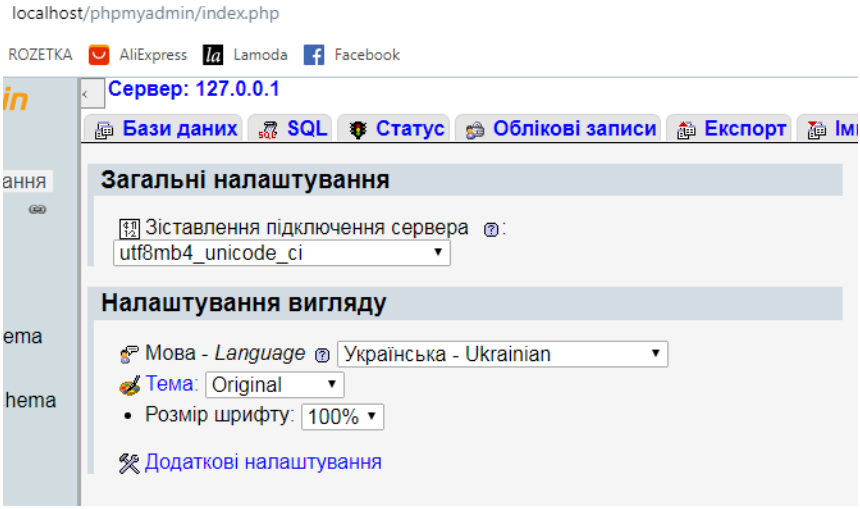

Рисунок 7.1 – Застосунок phpMyAdmin

Обравши пункт *Бази даних*, ввести ім'я бази даних і потім натиснути клавішу *Створити* (рис. 7.2).

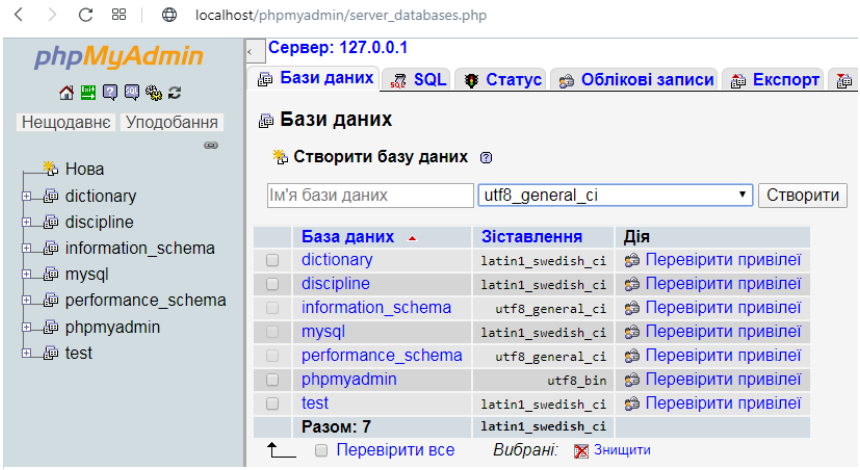

Рисунок 7.2 – Вкладка *Бази даних*

Ім’я створеної бази даних з’явиться зліва в списку доступних пунктів, потрібно її вибрати і натиснути *Створити таблицю* (рис. 7.3).

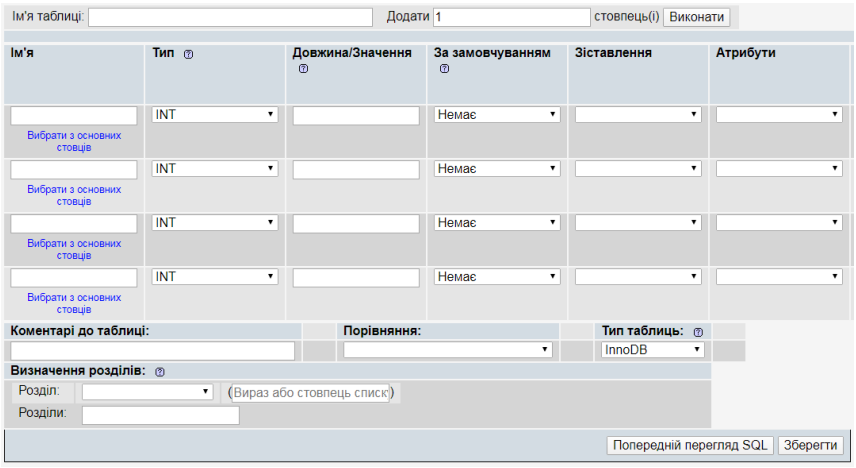

Рисунок 7.3 – Пункт *Створити таблицю*

7.2. Взаємодія PHP та MySQL

Перш ніж почати роботу з базою даних, необхідно встановити з'єднання з сервером.

Функція ***mysqli_connect([string server[, string usernsme[, string password]]])*** встановлює з'єднання з сервером, повертає вказівник на з'єднання або false в іншому випадку.

У MySQL для користувача з логіном *root* пароля не існує. Щоб можна було отримати доступ з PHP до баз даних MySQL, потрібно написати таке:

mysqli_connect("localhost", "root", "");

З'єднання з сервером закривається при завершенні виконання скрипту, якщо воно до цього не було закрито за допомогою функції ***mysqli_close(вказівник_на_з'єднання)***.

Після установлення з'єднання потрібно вибрати або створити базу даних, з якою потрібно працювати. Для створення бази даних краще використовувати команду ***mysqli_query(string query[, resource link_ identifier])***, яка надсилає запит активній базі даних сервера, на який посилається вказівник. Якщо параметр *link_identifier* опущено, використовується останнє відкрите з'єднання.

Функція ***die([string status])*** є псевдонімом функції ***exit([string status])***, закінчує виконання скрипту і друкує *status* безпосередньо перед виходом.

Приклад 7.1

```
<?php
$conn=mysqli_connect("localhost", "root")
or die("Не вдається виконати з'єднання:". mysqli_error());
echo("З'єднання встановлено");
```

```
$result=mysqli_query(“create database my_database”)
or die(“Помилка створення бази даних”. mysqli_error());
mysqli_close($conn);
?>
```

Якщо база даних вже створена, то після установлення з’єднання необхідно вибрати базу даних, з якої буде здійснена робота. У PHP для цього існує функція ***mysqli_select_db(database_name[, link_identifier])***. Ця функція повертає true в разі успішного вибору бази даних і false – в іншому випадку.

Приклад 7.2

```
<?php
$conn=mysqli_connect(“localhost”, “root”)
or die(“Не вдається виконати з’єднання:”. mysqli_error());
echo (“З’єднання встановлено”);
mysqli_select_db(“my_database”)
or die(“Помилка відкриття бази даних ”. mysqli_error());
mysqli_query(“Create table IF NOT EXISTS Persons(
id INT AUTO_INCREMENT,
FirstName CHAR(20),
LastName CHAR(50),
BirthDate DATE,
salary INT,
PRIMARY KEY(id))”)
or die(“Неможливо створити таблицю:”. mysqli_error());
mysqli_close($conn);
?>
```

Для того щоб отримати список полів таблиці, можна вчинити так:

1. Використовуючи функцію ***mysqli_list_fields(string database _name, string table_name[, resource link_identifier])***, можна отримати ресурс (змінну типу *resource*) або посилання, що містить інформацію про поля таблиці *table_name* бази даних *database_name*, включаючи їхні назви, типи і прапори.

2. Використовуючи функції:

– mysqli_field_name(resource result, int field_offset);

– mysqli_field_type(resource result, int field_offset);

– mysqli_field_flags(resource result, int field_offset);

– mysqli_field_len(resource result, int field_offset);

– mysqli_num_fields(resource result),

можна отримати ім'я поля, тип поля, список прапорів, записаних через пробіл і довжину поля.

Result – це ідентифікатор результату запиту (наприклад, запиту, відправленого функціями mysqli_list_fields або mysqli_query), а *field_offset* – порядковий номер поля в результаті.

Функція ***mysqli_num_fields(resource result)*** повертає кількість полів результату запиту.

Приклад 7.3

```
$list_f=mysqli_list_fields("my_database", "Persons", $conn);
$n=mysqli_num_fields($list_f);
echo $n;
for($i=0; $i<$n; $i++)
{
$name_f=mysqli_field_name($list_f, $i);
$type_f=mysqli_field_type($list_f, $i);
$len_f=mysqli_field_len($list_f, $i);
$flag_f=mysqli_field_flags($list_f, $i);
echo "Ім'я поля: ".$name_f. "<br>";
echo "Тип поля: ".$type_f. "<br>";
```

```
echo “Довжина поля: ”.$len_f. “<br>”;
echo “Прапори поля: ”.$flag_f. “<br>”;
}
mysqli_close($conn);
```

Результати створення таблиці бази даних SQL за допомогою PHP-скрипту, що дозволяє за допомогою форми заповнювати таблицю значеннями, які вводить користувач (рис. 7.4, 7.5):

Приклад 7.4

Файл task.php:

```
<?php
$conn=mysqli_connect(“localhost”, “root”);
if ($conn==TRUE)
{
echo “Connection established <br>”;
}
$database = “discipline”;
mysqli_select_db(“discipline”);
$sql = “CREATE TABLE disciplines(
Subject char(50),
Lector char(30),
Course int(5),
ExamCredit char(20))”;
mysqli_query($sql, $conn);
$list = mysqli_list_fields(“discipline”, “disciplines”, $conn);
$n = mysqli_num_fields($list);
echo “<b>Enter:</b><br>”;
echo “<FORM method=post action=insert.php>”;
echo “<TABLE>”;
echo “<TR><TD> Subject: <INPUT type=text name=Subject />
</TD></TR>”;
```

```
echo “<TR><TD> Lector: <INPUT type=text name=Lector /> </TD></TR>”;
echo “<TR><TD> Course: <INPUT type=text name=Course /> </TD></TR>”;
echo “<TR><TD> Exam or credit: <INPUT type=text name=Exam Credit /> </TD></TR>”;
echo “</TABLE> <br>”;
echo “<INPUT type=submit name=‘add’ value=‘Add’>”;
echo “</FORM>”;
mysqli_close($conn);
?>
```

Файл insert.php:

```
<?php
$conn=mysqli_connect(“localhost”, “root”);
$database=“discipline”;
$table_name= “disciplines”;
mysqli_select_db($database);
$list_f=mysqli_list_fields($database, $table_name, $conn);
$n=mysqli_num_fields($list_f);
$sql = “INSERT INTO $table_name
VALUES (‘$_POST[Subject]’, ‘$_POST[Lector]’, ‘$_POST[Course]’, ‘$_POST[ExamCredit]’)”;
if (!mysqli_query($sql, $conn))
{
die(‘Error:’ . mysqli_error());
}
echo “1 record added”;
mysqli_select_db($database);
$list_f=mysqli_list_fields($database, $table_name, $conn);
$n=mysqli_num_fields($list_f);
```

```
for ($j=0; $j<$n; $j++)
{
$names[]=mysqli_field_name($list_f, $j);
}
$sq1="SELECT * FROM $table_name";
$q=mysqli_query($sq1, $conn) or die("помилка запиту");
$n1=mysqli_num_rows($q);
echo "<TABLE width=90% align=center>
<TR><TD  align=center>  <b> $table_name </b> </TD></TR>
</TABLE>";
echo "<TABLE border=5 width=90% align=center>";
echo "<TR>";
foreach($names as $val)
{
echo "<TH ALIGN=CENTER> <FONT size=2> $val</FONT></TH>";
}
echo "</TR>";
for ($i=0; $i<$n1; $i++)
{
echo "<TR>";
foreach($names as $val)
{
$value=mysqli_result($q, $i, $val);
echo "<TD ALIGN=CENTER> <FONT size=2> $value </FONT>
</TD>";
}
echo "</TR>";
}
echo "</TABLE>";
?>
```

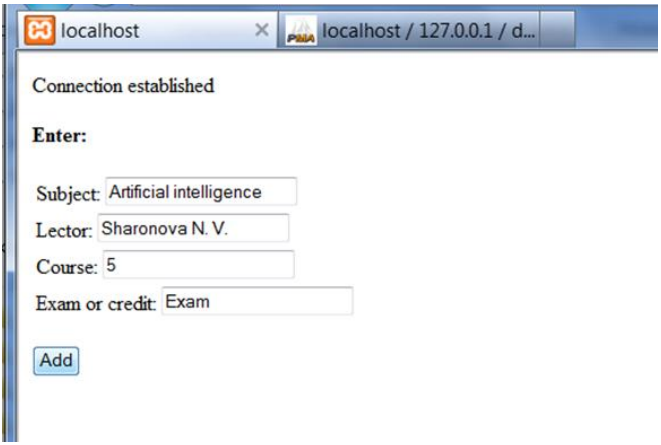

Рисунок 7.4 – Форма для введення даних

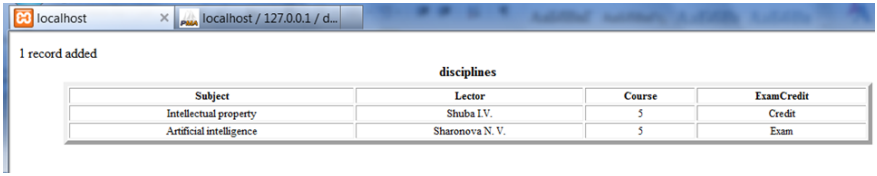

Рисунок 7.5 – Результат запиту до бази даних

7.3. Виконання запитів MySQL

Запит до бази даних, що надходить зі сценарію PHP, – це запит MySQL, поміщений у функцію ***mysqli_query()***, яка приймає як параметри:

1) рядок запиту, який поміщено в подвійні лапки;

2) необов'язковий ідентифікатор підключення, який отримано після виклику функції mysqli_conect().

Ця функція дозволяє використовувати основні оператори SQL – SELECT, INSERT, UPDATE, DELETE (це оператори роботи з даними) і CREATE, DROP (це оператори створення і видалення таблиць), але ця функція не дозволяє створювати і видаляти бази даних. Функція повертає

цілочислове значення, еквівалентне логічному true, якщо запит виконано успішно, і цілочислове значення, еквівалентне логічному false, якщо запит недопустимий або не був виконаний.

Рекомендується при цьому створювати додаткову змінну для використання у функції:

Приклад 7.5

```
<?php
$query=“SELECT Surname FROM personal WHERE ID>10”;
$result=mysqli_query($query);
if (!$result)
{
echo ‘’. mysqli_error();
exit();
}
?>
```

7.4. Вибірка наборів даних

Після вибору даних з таблиці запитом SQL вони не надходять відразу в PHP. Для того щоб дані стали повністю доступні в середовищі PHP, необхідно використовувати одну з функцій вибірки (функції отримання даних) з префіксом mysqli_fetch:

– mysqli_fetch_row() повертає рядок у вигляді масиву з числовим індексом;

– mysqli_fetch_object() повертає рядок у вигляді об’єкта;

– mysqli_fetch_array() повертає рядок у вигляді асоціативного масиву;

– mysqli_result() повертає один елемент масиву даних.

Приклад 7.6

```
$query = “SELECT ID, LastName, FirstName
FROM user WHERE Status =1”;
$result = mysqli_query($query);
$name_row = mysqli_fetch_row($result);
while($name_row)
{
echo $name_row[0];
echo $name_row[1];
echo $name_row[2]. “<br>”;
}
```

Функція mysqli_fetch_object() виконує в основному таке саме завдання, тільки рядок, що повертається, подано у вигляді об’єкта, а не у вигляді масиву. Об’єкт, що повертається, містить властивості (поля об’єкта), однойменні полям результатів запиту. Значення цих властивостей текстового типу і дорівнюють значенням відповідних полів.

Цикл з перебору всіх записів буде виглядати таким чином:

Приклад 7.7

```
$query = “SELECT ID, LastName, FirstName
FROM user WHERE Status =1”;
$result = mysqli_query($query);
$row = mysqli_fetch_object($result);
while($row){
echo $row->ID, $row->LastName, $row->FirstName, “<br>”;
$row = mysqli_fetch_object($result);
}
```

Найбільш зручна функція вибірки ***array mysqli_fetch_array(resource result[, int result_type])***, яка дозволяє отримати результат видачі у вигляді асоціативного масиву або масиву з числовими індексами (аналогічно функції mysqli_fetch_row()), тобто на додаток до функції mysql_fetch_ row(), коли до значень полів запису можна отримати доступ за іменами, а не за їхніми номерами полів.

Приклад 7.8

```
$query = “SELECT ID, LastName, FirstName FROM user WHERE Status =1”;
$result = mysqli_query($query);
while($row = mysqli_fetch_array($result))
{
echo $row[‘ID’], $row[‘LastName’], $row[‘FirstName’], “<br>”;
}
```

Функція ***mixed mysqli_result(resource result, int row[, mixed field])*** застосовується рідше за інші функції виведення і використовується в тому випадку, коли з бази даних потрібно повернути одну комірку результату запиту, оскільки функція працює повільніше за інші і її не можна змішувати з іншими функціями, що працюють з результатами запитів.

Аргумент поля може бути номером поля, ім’ям поля, ім’ям таблиці, ім’ям поля, аліас:

Приклад 7.9

```
$result = mysqli_query(“SELECT name FROM work.employee”)
or die(“Could not query: ”. mysqli_error());
echo mysqli_result($result, 2);
```

Функція ***int mysqli_num_rows(resource $result)*** повертає кількість рядків результату запиту. Команда працює тільки із запитами SELECT або SHOW.

Приклад 7.10

```
$result = mysqli_query(“SELECT * FROM table1”, $link);
$num_rows = mysqli_num_rows($result);
echo “Отримано $num_rows рядків”;
```

7.5. Реалізація засобів контролю помилок

Для контролю помилок роботи з базою даних зазвичай використовується функція ***die()***, яка завершує сценарій і повертає рядок, заданий як параметр. При цьому використовується синтаксична структура, яка використовує два виклики функцій, розділених оператором *or*.

Приклад 7.11

```
$result = mysqli_query(“SELECT name FROM work.employee”)
or die(“Could not query: ”. mysqli_error());
echo mysqli_result($result, 2);
```

Якщо необхідно отримати повідомлення про помилку інтерпретатора PHP, то необхідно використовувати функцію ***mysqli_errno()***, яка повертає умовний номер, відповідний помилці кожного типу, або ***mysqli_error()***, яка повертає текстове повідомлення.

7.6. Відображення результатів запитів SQL в таблицях HTML

Для відображення результату оператора SELECT зазвичай використовуються таблиці HTML, в яких для виведення кожного поля бази даних застосовується окремий стовпець таблиці HTML. Якщо об’єкт бази даних має *m* стовпців і *n* рядків, то у вікні браузера необхідно створити таблицю з розмірами *m* на *n*, заповнюючи кожну клітинку відповідним чином.

Приклад 7.12

```
<?php
$glodal_dbh = msqli_connect(“localhost”, “root”)
or die(“Could not connect to database”);
mysqli_select_db(“my_database", $glodal_dbh);
function display_db_table($tablename, connection)
{
$query_string = “SELECT * FROM  $tablename”;
$result_id = mysqli_query($query_string, $connection);
$column_count = mysqli_num_field($result_id);
print (“<TABLE BORDER=1>”);
while ($row = mysqli_fetch_row($result_id))
{
print (“<TR ALING=LEFT  VALING=TOP>”);
for ($column_num=0; $column_num < $column_count;
$column_num++)
print(“<TD> $row [column_num] </TD>”);
print (“</TR>”);
}
print (“</TABLE>”);
}
?>
<HTML>
<HEAD>
<TITLE> Cities and countries </TITLE>
</HEAD>
<BODY>
<TABLE>
<TR>
<TD>
<?php
display_db_table(“country”, $global_dbh);
```

```
?>
</TD>
<TD>
<?php
display_db_table(“city”, $global_dbh);
?>
</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Функція ***display_db_table()*** є універсальною. Для виведення рядків у функції використовується цикл while, а для окремих елементів використовується цикл for, кількість рядків визначається за допомогою функції mysqli_num_rows().

7.7. Способи застосування численних запитів або складних операторів виведення

Структура виведення може не збігатися зі структурою результатів запиту. Наприклад, запит, отриманий за результатами видачі записів з двох пов’язаних таблиць:

Приклад 7.13

```
SELECT country.continent, country.countryname, city.cityname
FROM country, city
WHERE city.countryID = country.ID
ORDER BY continent, countrytame, cityname
```

У результаті цього запиту для кожного міста буде показано, в якій країні та на якому континенті воно знаходиться, що і повинно відобразитися окремими рядками HTML. Тобто кожен раз країна і континент будуть повторюватися. Нам необхідно, щоб спочатку було вказано континент, потім країна, а потім усі міста, які знаходяться в цій країні. Тобто структура виведення повинна збігатися зі структурою найбільш зручної форми запиту. У такому випадку можна використовувати два підходи:

1. Створити спеціалізований запит до бази даних.
2. Використовувати більш складний код відображення.

Наприклад, використання декількох запитів:

Приклад 7.14

```
<?php
$glodal_dbh = msqli_connect(“localhost”, “root”)
or die(“Could not connect to database”);
mysqli_select_db(“my_database”, $glodal_dbh)
or die(“Could not select database”);
function display_cities($db_connection)
{
$country_query =“SELECT id, continent, countryname
FROM country
ORDER BY continent, countryname”;
$country_result=mysqli_query($country_query, $db_connection);
print (“<TABLE BORDER=1>”);
print (“<TR> <TH> Continent </TH> <TH> Country </TH>
<TH> City </TH> </TR>”);
while($country_row = mysqli_fetch_row($country_result))
{
$country_id=$country_row[0];
$continent=$country_row[1];
$country_name =$country_row[2];
print (“<TR ALING = LEFT VALING= TOP>”);
```

```
print("<TD> $continent </TD>");
print("<TD> $country_name </TD>");
print ("<TD>");
$city_query = "SELECT cityname from city
WHERE countryID=$country_id
ORDER BY cityname";
$city_result = mysqli_query($city_query, $db_connection)
or die(mysqli_error());
while($city_row =mysqli_fetch_row($city_result))
{
$city_name= $city_row[0];
print ("$city_name <br>");
}
print ("</TD> </TR>");
}
print ("</TABLE>")
}
?>
<HTML>
<HEAD>
<TITLE> Cities by countries </TITLE>
</HEAD>
<BODY>
<?php
display_cities($global_dbh);
?>
</BODY>
<HTML>
```

Такий підхід є простим, але не досить ефективним, тому що збільшується число окремих запитів до бази даних.

При використанні складних операторів друку виконуємо один запит, але інформація виводиться вибірково, коли кожен рядок HTML буде відповідати більше ніж одному рядку бази даних:

Приклад 7.15

```
<?php
$glodal_dbh = msqli_connect(“localhost”, “root”)
or die(“Could not connect to database”);
mysqli_select_db(“my_database”, $glodal_dbh)
or die(“Could not select database”);
function display_cities($db_connection)
{
$query = “SELECT country.id, country.continent,
country.countryname, city.cityname
FROM country, city
WHERE city.countryID = country.id
ORDER BY country.continent,
country.countryname,
city.cityname”;
$result_id=mysqli_query($query, $db_connection)
or die(mysqli_error($query));
print (“<TABLE BORDER=1>”);
print (“<TR> <TH> Continent </TH> <TH> Country </TH>
<TH> City </TH> </TR>”);
$old_country_id=0;
while($row_array = mysqli_fetch_row($result_id))
{
$country_id=$country_row[0];
if (country_id != $old_country_id)
{
$continent=$country_row[1];
$country_name =$country_row[2];
```

```
if ($old_country_id !=0)
print("</TD> <TR>");
print("<TR ALING = LEFT VALING= TOP>");
print("<TD> $continent </TD>");
print("<TD> $country_name </TD> <TD>");
$old_country_id=$country_id;
}
$city_name= $row_array[3];
print("$city_name <br>");
}
print("</TD> </TR></TABLE>");
}
?>
<HTML>
<HEAD>
<TITLE> Cities by countries </TITLE>
</HEAD>
<BODY>
<?php
display_cities($global_dbh);
?>
</BODY>
</HTML>
```

Другий спосіб набагато складніший, ніж перший. Тут дані обробляються послідовно, але держави виводяться в браузер тільки спочатку списку міст, що знаходяться на території цієї держави, виведення назв континентів все ще здійснюється неодноразово.

7.8. Додавання даних до таблиці

Для додавання даних до таблиці призначено команду INSERT:

INSERT INTO table COLUMNS ([стовпці]) VALUES ([значення]);

Якщо перелік стовпців *COLUMNS* не вказано, значення повинні йти в тому ж порядку, в якому визначалися стовпці при створенні таблиці.

Зауваження:

– числові значення повинні вказуватися без лапок;

– рядкові значення завжди повинні бути в лапках;

– дата й час завжди повинні бути в лапках;

– функції мають бути вказані без лапок;

– значення NULL ніколи не повинно поміщатися в лапки.

Приклад 7.16

```
CREATE TABLE books(
title_id INT NOT NULL AUTO_INCREMENT,
title VARCHAR(150),
pages INT,
PRIMARY KEY(title_id));
CREATE TABLE authors(
author_id INT NOT NULL AUTO_INCREMENT,
title_id INT NOT NULL,
author VARCHAR(125),
PRIMARY KEY(author_id));
```

Існує три основних синтаксиси:

1. INSERT ... VALUES – вказується порядок проходження полів та їхніх значень:

Приклад 7.17

INSERT INTO ‘books’ (‘title’, ‘pages’) VALUES (‘PHP: Manual’, 645);

2. INSERT ... SET – кожному полю, присутньому в таблиці, присвоюється значення у вигляді ім’я_поля => ‘значення’:

Приклад 7.18

```
INSERT INTO ‘books’
SET
‘Title’ => ‘PHP: Manual’,
‘Pages’ => 645
```

3. INSERT ... SELECT – більш складний синтаксис, що дозволяє внести в таблицю більшу кількість записів за один раз, причому з різних таблиць:

Приклад 7.19

```
INSERT INTO ‘books_new’
SELECT * FROM ‘books’
WHERE ‘pages’ > 700
```

У всіх трьох випадках, якщо для полів, присутніх у таблиці, не встановити значення, то використовуються значення, встановлені початково при створенні таблиць.

Наприклад, таблиці, інформація з яких відображалася в попередньому прикладі (приклад 7.15), можуть бути побудовані і заповнені за допомогою застосунку phpMyAdmin, а можуть бути побудовані і заповнені засобами PHP.

Приклад 7.20

```
<?php
$glodal_dbh = mysqli_connect(“localhost”, “root”)
or die(“Could not connect to database”);
mysqli_select_db(“my_database”, $glodal_dbh)
or die(“Could not select database”);
function add_new_country($dbh, $continent, $countryname,
$city_array)
{
$country_query = “INSERT INTO country(continent, countryname)
VALUE(‘$continent’, ‘$countryname’)”;
$result_id = mysqli_query($country_query)
OR die($country_query.mysqli_error());
if($result_id)
{
$countryID = mysqli_insert_id($dbh);
```

/* функція mysqli_insert_id() повертає ID, що генерується для стовпця з атрибутом AUTO_INCREMENT попереднім запитом. Цю функцію слід використовувати після виконання запиту INSERT в таблиці, що містить поле AUTO_INCREMENT */

```
for($city = current($city_array);
$city;
$city = next($city_array))
{
$city_query= “INSERT INTO city(countyID, cityname)
VALUES(‘$countryID’, ‘$city’)”;
mysqli_query($city_query, $dbh)
OR die($city_query. mysqli_error());
}
}
}
```

```
function populate_cities_db($dbh)
{
mysqli_query("DROP TABLE city", $dbh);
mysqli_query("DROP TABLE country", $dbh);
mysqli_query("CREATE TABLE country(
ID int null auto_increment primary key,
continent varchar(50),
countryname varchar (50))", $dbh)
OR die(mysqli_error());
add_new_country($dbh, 'Africa', 'Kenya',
array('Naiobi', 'Mombasa', 'Meru'));
add_new_country($dbh, 'South America', 'Brazil',
array('Rio de Janeiro', 'Sau Paulo', 'Salvador', 'Belo Horizonte'));
add_new_country($dbh, 'North America', 'USA',
array('Chicago', 'New York', 'Houston'));
add_new_country ($dbh, 'North America', 'Canada',
array('Monreal', 'Windsor', 'Winnipeg'));
print ("Sample database created <br>");
}
?>
<HTML>
<HEAD>
<TITLE> Creating a sample database </TITLE>
 </HEAD>
<BODY>
<?php
populate_cities_db($global_dbh);
?>
</BODY>
</HTML>
```

За допомогою команди INSERT в таблиці створюються нові рядки і дані, що передаються, є рядковими даними. В прикладі 7.20 в таблицю передається довільна кількість міст у розрахунку на кожну державу за допомогою масиву. При цьому необхідно, щоб кожен рядок *city* мав відповідний ідентифікатор *countryID*, який повинен дорівнювати фактичним значенням у полі *ID* відповідного рядка таблиці з назвами держав. Але ці значення *countryID* автоматично присвоюються послідовно базою даних MySQL і не знаходяться під нашим контролем.

Для того щоб визначити правильне значення *countryID*, що має бути присвоєно, використовується функція *mysqli_insert_id()*, яка вибирає значення *ID*, пов'язане з останнім запитом *INSERT*: виконується вставка назви нової держави, вибірка значення ID з новоствореного рядка, після чого це значення використовується в запитах для вставки назв міст.

Функції *next()* і *current()* використовуються для роботи з масивами. Функція *current()* повертає значення елемента масиву, на який вказує його внутрішній вказівник. Якщо внутрішній вказівник за межами списку елементів, *current()* повертає false.

Функція *next()* перед тим, як повернути значення елемента масиву, передає його внутрішній вказівник на одну позицію вперед. Якщо під час руху внутрішній вказівник вийде за межу масиву, то функція поверне false.

Контрольні запитання

1. Поясніть термін MySQL.
2. Як працює застосунок phpMyAdmin?
3. Яким чином взаємодіє PHP і MySQL?
4. Як записати дані в базу даних за допомогою HTML-форми?
5. Назвіть способи обробки запитів MySQL.
6. Яким чином вносяться дані в таблицю?
7. Які функції вибірки наборів даних застосовуються в PHP?

РОЗДІЛ 8
ОБ'ЄКТИ PHP

8.1. Клас у PHP

Мова PHP підтримує об'єктну модель. У PHP клас визначається за допомогою наступного синтаксису, в якому використовується ключове слово ***class*** (імена класів і функцій не можна починати з символу підкреслення), імена властивостей класу визначаються ключовим словом ***var***, а методи, що застосовуються до об'єктів класу, описуються функціями:

```
class ім'я_класу
{
var $ім'я_власивості;
//список властивостей
function ім'я_методу( )
{
// визначення методу
}
//список методів
}
```

Усередині визначення класу можна використовувати ключове слово ***this*** для звернення до поточного представника класу.

Приклад 8.1

```
<?
class Articles
{
var $title;
var $author = “Іванов”;
var $description;
var $publisher;
function _construct()
{
// метод, який присвоює значення атрибутам класу
$this->title = $_POST[“title”];
$this->description = $_POST[“description”];
$this->$publisher = date(“Y-m-d”);
}
function show_article()
{
// метод для відображення екземплярів класу
$art = $this->title . “<br>” .
$this->description . “Автор: ” .
$this->author;
echo $art;
}
}
?>
```

У PHP існують два стандартних методи ініціалізації значення:

– за допомогою оператора *var* – ініціалізація константних значень (в прикладі 8.1 автор дорівнює рядку *Іванов*);

– за допомогою конструктора, який викликається автоматично при створенні об’єкта класу.

Існує два способи створення об’єкта:

$Ім’я_об’єкта = new ім’я_класу;

$Ім’я_об’єкта = new ім’я_класу(параметри);

Для доступу до властивостей і методів об’єкта використовується такий синтаксис:

$Ім’я_об’єкта -> назва_властивості;

$Ім’я_об’єкта -> назва_методу(список аргументів);

Перед назвою властивості або методу знак $ не ставлять.

Приклад 8.2

```
<?php
$art = new Articles;
echo($art->title);
$another_art = new Articles;
$another_art->show_article();
?>
```

8.2. Наслідування класів у PHP

PHP не підтримує множинне наслідування, тобто розширювальний клас завжди наслідує тільки один базовий клас. Для створення підкласів використовується ключове слово ***extends***.

Приклад 8.3

```
<?php
class Person
{
var $first_name;
var $last_name;
```

```
function make_person($t, $a)
{
$this->first_name = $t;
$this->last_name = $a;
}
function show_person()
{
echo (“<h2>” . $this->first_name . “ ” . $this->last_name . “</h2>”);
}
}
class Programmer extends Person
{
var $langs = array(“Lisp”);
function set_lang($new_lang)
{
$this->langs[] = $new_lang;
}
}
$progr = new Programmer;
$progr->set_lang(“PHP”);
// методи, визначені для класу Programmer
print_r($progr->langs);
// методи, визначені для класу Person
$progr->make_person(“Bill”, “Gates”);
$progr->show_person();
?>
```

Клас *Programmer* має ті ж змінні і функції, що і клас *Person*, плюс змінну *$langs*, в якій міститься список вивчених програмістом мов, і функцію *set_lang()* для додавання ще однієї мови до списку вивчених. Створити представника класу програмістів можна звичайним способом за допомогою конструкції *new*. Після цього можна встановлювати і отриму-

вати список мов, які знає програміст, і в той же час можна використовувати функції, задані для класу *Person*, тобто встановлювати і отримувати ім'я і прізвище програміста і відображати відомості про нього в браузері.

Коли створюється підклас і його конструктор, то першою інструкцією цього конструктора повинно бути звернення або виклик конструктора батьківського класу, для цього так само використовується інструкція **parent::**.

Приклад 8.4

```
<?php
$object = new Tiger();
echo “У тигрів є… <br>”;
echo “хутро:”.$object->fur. “<br>”;
echo “смуги:” .$object->stripes;
class Wildcat
{
public $fur;
function_construct()
{
$this->fur = “TRUE”;
}
}
class Tiger extends Wildcat
{
public $stripes;
function_conctruct()
{
parent::_construct();
this->stripes = “TRUE”;
}
}
?>
```

Для того щоб підклас не міг перевантажити метод суперкласу, необхідно використовувати ключове слово ***final function ім'я_функції()***.

Початково екземпляри і методи класу є відкритими (public). Крім відкритих можуть бути закриті (private) і захищені (protected) поля і методи.

8.3. Функції самодіагностики

Самодіагностика дозволяє з об'єктів отримувати інформацію про класи цих об'єктів і батьківські класи.

Функція ***string get_class(object obj)*** повертає ім'я класу, екземпляром якого є об'єкт *obj*.

Функція ***string get_parent_class(mixed obj)*** повертає ім'я батьківського класу для об'єкта або класу.

Функція ***array get_class_vars(string class_name)*** повертає асоціативний масив властивостей класу та їхні значення. Масив має формат *властивість => ініціалізоване значення*.

Функція ***array get_class_methods(mixed class_name)*** повертає масив рядків, що подають імена методів визначених для класу *class_name*.

Приклад 8.5

```
<?php
class myclass
{
var $var1;
// змінна не має початкового значення
var $var2 = "xyz";
var $var3 = 100;
function myclass()
{
$this->var1 = "foo";
$this->var2 = "bar";
```

```
return true;
}
}
$my_class = new myclass();
$class_vars = get_class_vars(get_class($my_class));
/* get_class поверне клас $class_vars, якому належить об’єкт $my_class */
foreach($class_vars as $name=>$value)
{
echo “$name : $value”;
}
?>
```

Контрольні запитання

1. Опишіть синтаксис для визначення класу в PHP.
2. Яким чином ініціалізуються значенняв PHP?
3. Назвіть основні способи створення об’єктів й доступу до властивостей та методів об’єктів.
4. Яке ключове слово використовується для створення підкласів?
5. Яка конструкція використовується для наслідування класів у PHP?
6. Укажіть функції самодіагностики.

СПИСОК ЛІТЕРАТУРИ

1. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5 / Р. Никсон. – Санкт-Петербург : Питер, 2016. – 768 с.

2. Прохоренок Н. А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера / Н. А. Прохоренок. – Санкт-Петербург : БХВ-Петербург, 2010. – 912 с.

3. Котеров Д. В. PHP 7 / Д. В. Котеров, И. В. Симдянов. – Санкт-Петербург : БХВ-Петербург, 2016. – 1088 с.

4. PHP: Hypertext Preprocessor [Електронний ресурс]. – Режим доступу : https://www.php.net/

5. PHP.SU: Портал PHP, MySQL [Електронний ресурс]. – Режим доступу : http://www.php.su/

ЗМІСТ