Дрогобицький державний педагогічний університет

імені Івана Франка

Олег Наум, Дмитро Карпин, Анна Карпин

ВЕБТЕХНОЛОГІЇ. РОБОТА З БАЗОЮ ДАНИХ

Методичні рекомендації до виконання лабораторних робіт

Дрогобич

2025

УДК 004.42(076.5)

Н34

Рекомендовано до друку вченою радою Дрогобицького державного педагогічного університету імені Івана Франка

(протокол № 7 від 26.06.2025 року)

Рецензенти:

Лучкевич М.В. – кандидат фізико-математичних наук, доцент кафедри інформаційних систем та мереж Національний університет «Львівська політехніка».

Сікора О. В. – кандидат технічних наук, доцент, доцент кафедри фізики та інформаційних систем Дрогобицького державного педагогічного університету імені Івана Франка.

Відповідальний за випуск:

Гольський В. Б. – доцент, завідувач кафедри фізики та інформаційних систем Дрогобицького державного педагогічного університету імені Івана Франка.

Наум Олег, Карпин Дмитро, Карпин Анна.

“**Вебтехнології. Робота з базою даних”** : навчально-методичний посібник до виконання лабораторних робіт з дисципліни “Вебтехнології”. Дрогобич : ДДПУ ім. І. Франка, 2025. 56 с.

Навчально-методичний посібник розроблено відповідно до програми навчальної дисципліни “Вебтехнології” для підготовки фахівців освітньо-кваліфікаційного рівня “Бакалавр” спеціальності 122 “Комп’ютерні науки”. Посібник містить покрокові інструкції до виконання поставлених завдань та завдання для самостійного виконання 4 лабораторних робіт засобами інтерпретатору PHP.

Бібліографія 10 назв.

Зміст

Вступ

В умовах інтенсивного розвитку мережі Інтернету з'являються нові технології, витісняючи ті, що здавалось б, нещодавно ввійшли до повсякденної практики. Сьогодні вебтехнології стали невід'ємною частиною сучасного світу, вони значною мірою впливають на подальший суспільний розвиток людства. Вебтехнології, проникають практично у всі сфери людської діяльності. Інтернет став неодмінним атрибутом робочого місця працівників багатьох професій.

У наші дні роль розробки додатків для WWW в структурі глобальної мережі зростає, відповідно збільшується і середня оцінка складності сценаріїв. Багато систем (наприклад, пошукові) за обсягом коду наближаються до розміру вихідних кодів серйозних прикладних програм. Частка ж статичних сторінок у Веб постійно падає; на зміну їм приходять динамічні сторінки, що генеруються автоматично тим чи тим сценарієм.

Мета цього практикуму – ознайомити студентів з мовою програмування PHP для роботи із базою даних MySQL.

У поданих лабораторних роботах як інструменти для створення скриптів використовується Visual Studio Code, для обробки http-запитів – веб-сервер Apache, СУБД – MySQL.

Матеріал лабораторного практикуму розподілений на 4 роботи, форма викладання яких методично продумана й забезпечує поступове послідовне засвоєння основних конструкцій мови PHP та технологій створення веб-сайту. Кожна лабораторна робота містить теоретичні відомості, у яких досить ґрунтовно описані основні аспекти з певної теми. Теми розташовані у такому порядку, щоб кожна наступна не лише надавала нові знання, але й закріплювала та доповнювала попередні. У кожній темі, крім теоретичного матеріалу, міститься код реалізації прикладів завдань, що допомагають зрозуміти спосіб виконання поставленого завдання. Контрольні запитання

подані в кінці кожної теми дають змогу перевірити, як засвоєні знання з певної тематики.

Простота й послідовність викладеного матеріалу, наявність прикладів із повним описом застосування роблять цей навчальний посібник загальнодоступним. Він успішно може бути використаний для самостійного вивчення роботи з СУБД MySQL за допомогою PHP. Особи, які у повному обсязі засвоять запропонований матеріал, будуть достатньо підготовлені для самостійного розв’язання завдань, що виникають у практичній діяльності.

Зміст практикуму відповідає типовим освітнім програмам і розрахований на студентів ВНЗ, слухачів курсів, а також самоосвіти.

1. Утиліта для роботи з MySQL. Створення бази даних та таблиць.

Мета роботи: навчитись створювати базу даних та таблиці в MySQL.

Теоретичні відомості

База даних — це впорядкована сукупність взаємопов'язаних даних, яка зберігається у вигляді двовимірних таблиць у межах інформаційної системи. Програмне забезпечення, що дозволяє створювати, керувати та колективно використовувати такі бази, називається **системою управління базами даних (СУБД)**.

У контексті використання, база даних **MySQL** являє собою організований набір таблиць із заданими назвами. Кожна таблиця містить однорідні елементи — **записи**, які є основними одиницями інформації. Хоча записи вважаються цілісними, за потреби можна отримувати окремі частини кожного з них.

Кожен запис складається з одного або кількох **іменованих полів**, визначених під час створення таблиці. Поля мають конкретні типи даних, наприклад, цілі числа, текстові рядки або масиви символів.

У таблицю можна додавати нові записи, а також здійснювати пошук за різними умовами. Наприклад, можна сформулювати запит на вибір усіх записів, де: перше поле містить число менше 10, друге — рядок із певним словом (наприклад, «word»), а третє — не дорівнює нулю. Знайдені записи можна частково обробляти або повністю видаляти.

Варто зазначити, що подібні операції виконуються дуже швидко. Наприклад, сервер **MySQL** здатен менш ніж за 0,01 секунди знайти та видалити запис серед 10 мільйонів, якщо значення певного поля збігається з заданим. Така швидкодія забезпечується завдяки структурованому зберіганню даних і постійному підтриманню їх впорядкованого стану.

Архітектура MYSQL

Однією з найпоширеніших систем управління базами даних (СУБД), що активно використовується у вебпрограмуванні, є MySQL. Вона орієнтована на створення відносно невеликих баз даних (приблизно до 100 МБ) і підтримує обмежений набір можливостей мови SQL.

SQL (Structured Query Language — мова структурованих запитів) — це стандартизована мова, призначена для роботи з базами даних. Вона включає команди для:

- створення або видалення таблиць;
- додавання нових записів у таблиці;
- пошуку та групування записів, зокрема в пов’язаних таблицях;
- видалення записів за заданими умовами;
- оновлення окремих полів у наявних записах.

MySQL працює як серверна програма, яка постійно функціонує на комп’ютері. Клієнтські програми (наприклад, веб-сценарії) надсилають їй SQL-запити через сокети — тобто використовуючи мережеві інтерфейси. Сервер обробляє ці запити, зберігає результати й за потреби надсилає відповіді назад клієнтам.

Оскільки MySQL реалізує клієнт-серверну архітектуру, на машині має бути запущено сервер MySQL, який приймає та обробляє запити. Якщо сервер і клієнт знаходяться на одному комп’ютері, то витрати на обробку таких запитів мінімальні — підключення до сервера і передача даних відбувається швидко і з незначними ресурсними затратами.

Структура MySQL є трирівневою: бази даних → таблиці → записи. Один сервер може обслуговувати кілька баз даних одночасно. Доступ до кожної з них визначається логіном і паролем користувача. Після авторизації

користувач може виконувати різні дії: створювати або видаляти таблиці, додавати записи тощо.

Адміністрування бази даних

Для доступу до MySQL можна використовувати як настільні програми для операційної системи, так і вебдодатки, наприклад phpMyAdmin. Цей веб-інструмент написаний на PHP і працює у браузері. Системи адміністрування баз даних надають зручний інтерфейс для перегляду та редагування баз і таблиць, що в них містяться. Вони також дозволяють змінювати структуру таблиць (наприклад, додавати нові поля) та переглядати різноманітну статистику. Однією з ключових переваг є можливість переглядати результати вручну введених SQL-запитів у зручному форматі, що особливо корисно під час налагодження скриптів, коли виникають проблеми з виконанням команд.

Створення таблиць

При створенні таблиць задається назва таблиці, поля та індекси.

Типи полів

Далі перераховані більшість типів полий, які можуть застосовуватися в MYSQL.

Цілі числа

У базах даних існує кілька типів цілих чисел, які відрізняються обсягом пам’яті (в байтах), що виділяється для їх зберігання, та максимальною кількістю значень, які вони можуть містити. Всі ці типи мають схожий формат запису (із певними скороченнями) і подаються у вигляді:

Префікс INT [UNSIGNED]

Опціональний модифікатор **UNSIGNED** вказує, що значення в полі можуть бути лише невід’ємними (тобто більше або рівні нулю). Назви типів, позначені як "префікс INT", наведено в таблиці 1.1.

Таблиця 1.1. Типи цілочисельних даних MySQL

Тип	**Опис**
TINYINT	Може зберігати числа -128 до +127
SMALLINT	Діапазон від -32 768 до 32 767
MEDIUMINT	Діапазон від -8 388 608 до 8 388 607
INT	Діапазон від -2 147 483 648 до 2 147 483 647
BIGINT	Діапазон від -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807

Дійсні числа

Так само, як і цілі числа, дробові числа в MySQL поділяються на кілька типів. У загальному вигляді їхнє оголошення має формат:

Ім'яТипу [(length, decimals)] [ONSIGNED]

Тут length — це загальна ширина поля (тобто кількість символів, що відводиться під число при його передачі, наприклад, у PHP), а decimals — кількість цифр після десяткової крапки. Ключове слово UNSIGNED, як завжди, означає, що значення не може бути від'ємним. Замість ІмяТипу використовується конкретне ім'я одного з типів дійсних чисел, перелічених у таблиці 1.2.

Таблиця 1.2. *Типи раціональних чисел в MySQL*

Тип	**Опис**
FLOAT	Число з плаваючою комою невеликої точності
DOUBLE	Число з плаваючою комою подвійної точності
REAL	Синонім для DOUBLE
DECIMAL	Дробове число, що зберігається у вигляді рядка
NUMERIC	Синонім для DECIMAL

Рядки

Рядки в базі даних — це послідовності символів. Під час пошуку за допомогою запиту SELECT регістр символів зазвичай ігнорується, тому рядки "ручка" і "РУЧКА" вважаються однаковими. Якщо база даних налаштована на автоматичне перекодування тексту при збереженні та читанні (див. нижче), то такі поля зберігатимуться у вказаному кодуванні.

Почнемо з типу рядка, який дозволяє зберігати не більше ніж **length** символів, де **length** може бути від 1 до 255:

VARCHAR { length} [BINARY]

Коли значення записується в поле цього типу, усі пробіли в кінці автоматично видаляються (як при виклику функції rtrim()). Якщо вказано параметр BINARY, то при виконанні запиту SELECT регістр символів буде враховуватись.

Тип VARCHAR має обмеження — він може зберігати лише до 255 символів. Якщо рядок може бути довшим, доцільно використовувати інші текстові типи, перелічені в таблиці 1.3.

Таблиця 1.3. *Рядкові типи даних таблиць MYSQL*

Тип	**Опис**
TINYTEXT	Може зберігати максимум 255 символів
TEXT	Може зберігати не більше 65 535 символів
MEDIUMTEXT	Може зберігати максимум 16 777 215 символів
LONGTEXT	Може зберігати 4 294 967 295 символів

Найчастіше використовують тип TEXT, але якщо існує ймовірність, що розмір даних перевищить 65 536 байтів, краще обрати LONGTEXT.

Бінарні дані

Бінарні дані схожі на дані у форматі TEXT, однак при пошуку в них враховується регістр символів (наприклад, "abc" і "ABC" вважаються різними рядками). Існує чотири типи бінарних даних (див. табл. 1.4).

Таблиця 1.4. *Типи бінарних даних, використовувані в MYSQL*

Тип	**Опис**
TINYBLOB	Може зберігати максимум 255 символів
BLOB	Може зберігати не більше 65 535 символів
MEDIUMBLOB	Може зберігати максимум 16 777 215 символів
LONGBLOB	Може зберігати 4 294 967 295 символів

Дата і час

MySQL підтримує різні типи полів, спеціально розроблених для зберігання дат і часу в різних форматах (табл. 1.5).

Таблиця 1.5. *Представлення дат і часу в базах даних MYSQL*

Тип	**Опис**
DATE	Дата у форматі рррр-мм-дд
TIME	Час у форматі гг:хх:сс

DATETIME	Дата і час у форматі рррр-мм-дд гг:хх:сс
TIMESTAMP	Час і дата представлені у форматі Unix timestamp, але при отриманні значення це поле відображається не як timestamp, а у вигляді ррррммддггххсс, що значно знижує ефективність використання цього формату в PHP.

Варто зазначити, що в деяких випадках в PHP буде зручніше самостійно генерувати дату та час при вставці даних у таблицю, ніж використовувати вбудовані типи MySQL. Наприклад, тип TIMESTAMP, який виглядає привабливо, насправді може бути незручним, оскільки він відображається не в тому форматі, як очікується. Хоча MySQL має ряд вбудованих функцій для перетворення дати та часу в різні текстові формати, на практиці їх використання не завжди є доцільним.

Перелічувальний тип

MySQL також підтримує кілька специфічних типів даних, використання яких у PHP, чесно кажучи, може бути малозначущим. Наприклад, тип ENUM дозволяє визначити, що значення поля не може бути будь-яким рядком чи числом, а лише одним із кількох вказаних при створенні таблиці значень, таких як value1, value2 тощо. Ось як виглядає визначення типу ENUM:

ENUM{value1,value2,value3 ...}

Множини

На відміну від інших типів даних, множини дозволяють зберігати в одному полі не одне значення, а кілька (value1, value2 і так далі, тобто множину значень). Формат подання даних цього типу виглядає наступним чином:

SET(valuel,value2, values ...)

Модифікатори типів

До типу також можна додавати модифікатори, що визначають його "поведінку" та операції, які можна (або заборонено) виконувати з відповідними стовпцями. Найпоширеніші з них наведені в таблиці 1.6.

Таблиця 1.6. *Основні модифікатори типів*

Тип	**Опис**
NOT NULL	означає, що поле не може містити порожнє або невизначене значення. Це поле обов'язково повинно бути ініціалізоване при додаванні нового запису, якщо

	для нього не задано значення за умовчанням.
PRIMARY KEY	визначає поле як первинний ключ, який є унікальним ідентифікатором запису, на який можуть посилатися інші таблиці.
AUTO_INCREMENT	при додаванні нового запису значення цього поля автоматично збільшується, забезпечуючи унікальність значень, щоб в таблиці не було двох однакових значень для цього поля.
DEFAULT '*значення*'	визначає значення за умовчанням для поля, яке буде використано, якщо при додаванні запису це поле не було явно ініціалізовано.

Завдання

Створити базу даних та таблиці у ній у відповідності до обраної предметної області.

Методичні рекомендації

Для виконання даної лабораторної роботи необхідно створити ER-діаграму для обраної предметної області. На рисунку 1.1 наведено ER-діаграма для бази даних з інформацією про студентів та академічні групи.

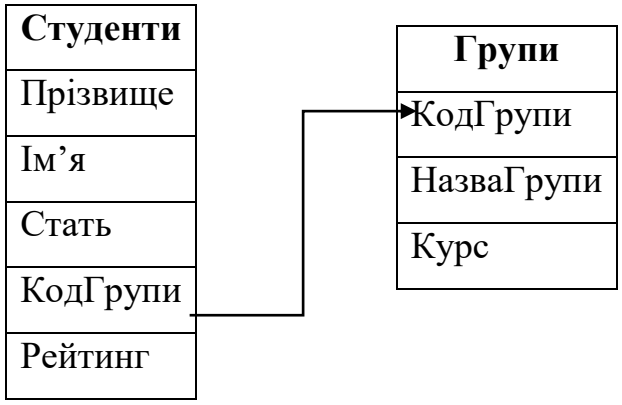


Рис.1.1. ER-діаграма

На рисунках 1.2 та 1.3 зображені структури таблиць «Групи» та «Студенти» створені за допомогою phpMyAdmin.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	group_name	varchar(255)	cp1251_general_ci		No	None			Change Drop More

Рис. 1.2. Структура таблиці "Групи"

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	surname	varchar(100)	cp1251_general_ci		No	None			Change Drop More
3	name	varchar(100)	cp1251_general_ci		No	None			Change Drop More
4	stat	varchar(1)	cp1251_general_ci		No	None			Change Drop More
5	rating	int(11)			No	None			Change Drop More
6	group_id	int(11)			No	None			Change Drop More

Рис. 1.3. Структура таблиці "Студенти"

Контрольні запитання

1. Які дії можна виконати за допомогою SQL-запитів у MySQL?
2. Яке програмне забезпечення можна використовувати для адміністрування баз даних MySQL?
3. Які типи даних для полів підтримує СУБД MySQL?
4. Які формати представлення дати і часу підтримує MySQL?

2. Вибірка даних та вивід. Добавлення, редагування та видалення

Мета роботи: навчитись використовувати функції PHP для виконання запитів до бази даних.

Теоретичні відомості

У всіх PHP-скриптах взаємодія з базою даних відбувається за схожим алгоритмом: спочатку необхідно встановити з'єднання з сервером СУБД, потім — виконати одну або кілька SQL-команд з обробкою можливих помилок. Після завершення роботи з базою з'єднання можна закрити (хоча PHP робить це автоматично при завершенні виконання скрипту).

З'єднання з сервером

Щоб працювати з базою даних, спочатку потрібно встановити з нею мережеве з'єднання та пройти авторизацію. Це виконується за допомогою функції:

```
int mysqli_connect ([string $host]   [, string $user]
[, string $passvord] )
```

Функція mysqli_connect() створює підключення до сервера MySQL, розміщеного за адресою $host (типово — localhost) з вказаним іменем користувача $user та паролем $password. В результаті повертається ідентифікатор підключення, з яким надалі працює скрипт. $host також може містити номер порту у форматі ім'я_хоста:порт. Після завершення скрипту з'єднання автоматично закривається, хоча його можна закрити вручну через mysqli_close().

Перш ніж надсилати запити, потрібно вказати, з якою базою працюватимемо. Для цього використовується функція:

```
int mysqli_select_db (string $dbname   [,int
$link_identifier] )
```

Ця функція вказує, що з'єднання $link_identifier (або останнє відкрите з'єднання) працюватиме з базою $dbname. PHP дозволяє одному користувачу мати доступ до кількох БД, проте на хостингах зазвичай надається доступ лише до однієї бази з іменем, що збігається з логіном користувача.

Обробка помилок

У разі виникнення помилок під час роботи з MySQL (наприклад, через незбалансовані дужки у запиті або відсутні параметри), можна отримати повідомлення про помилку та її код за допомогою двох спеціальних функцій, які буде розглянуто далі.

Важливо використовувати ці функції своєчасно та обережно, адже інакше налагодження ваших скриптів може стати надзвичайно складним. Далі ми наведемо багато прикладів, як правильно це реалізувати.

```
int mysqli_errno ([int $link_identifier])
```

Функція повертає код останньої помилки, що виникла. Якщо під час виконання сценарію було встановлено лише одне з'єднання, параметр *$link_identifier* можна не вказувати.

```
string mysqli_error ([ int $link_identifier])
```

Ця функція повертає не числовий код, а текстове повідомлення з описом помилки. Вона особливо корисна під час налагодження. Найчастіше mysqli_error використовують разом із конструкцією or die(), наприклад:

```
@mysqli_connect("localhost",   "user",   "password")
or die("Error connecting to database: ".mysqli_error());
```

Оператор @ традиційно застосовується для придушення стандартних попереджень, які можуть з'явитися у разі виникнення помилки.

Виконання запитів до бази даних

Настав час розглянути, як саме формувати та надсилати запити до бази даних. Для цього використовується основна функція — mysqli_query(). Вона повертає ідентифікатор результуючого набору даних (так званий result set).

Після виклику цієї функції сам результат запиту одразу не надсилається клієнту. Щоб отримати доступ до даних, використовується повернутий ідентифікатор. Існує багато допоміжних функцій, які приймають цей ідентифікатор як аргумент і дозволяють витягувати потрібну інформацію з результату запиту.

Варто також зазначити, що деякі запити не повертають результуючого набору даних. Наприклад, запит на видалення запису з таблиці.

Кожен запит до MySQL-сервера є звичайним текстовим рядком, написаним мовою SQL. У цьому рядку вказується команда (наприклад, SELECT, DELETE, UPDATE) і, при потребі, дані (в апострофах), які використовуються для пошуку або модифікації записів у таблицях. Наприклад, наступний SQL-запит виконує пошук усіх записів у таблиці Movies, де назва фільму містить "one":

```
SELECT * FROM Movies WHERE title like 'one'
```

```
int mysqli_query(string $query [,int $link_identifier])
```

Ця функція є універсальним інструментом: вона надсилає MySQL-серверу запит $query і повертає ідентифікатор відповіді або результату. Параметр $query — це рядок, сформований відповідно до синтаксису SQL. Для виконання запиту використовується попередньо встановлене з’єднання $link_identifier, а якщо його не вказано — буде використане останнє відкрите з’єднання.

Деякі SQL-команди, наприклад UPDATE, INSERT тощо, не повертають результат у вигляді набору даних, а лише сигналізують про успішність виконання. У таких випадках функція повертає відповідне числове значення — кількість

записів, що були змінені або додані. Якщо повертається нуль, це означає, що жодного запису не було зачеплено.

Натомість для запиту SELECT функція повертає ідентифікатор результуючого набору. Якщо повертається нульове значення, це вказує на те, що сталася помилка при виконанні запиту.

Варто пам’ятати, що одним викликом mysqli_query() можна виконати лише одну SQL-команду. На відміну від інтерфейсу командного рядка MySQL або деяких інших мов програмування, у PHP не підтримується виконання кількох команд одночасно, навіть якщо вони розділені крапками з комами.

Автоматизація підключення до СУБД

Зазвичай на сайті працює кілька скриптів, яким необхідно підключатися до однієї й тієї ж бази даних. Щоб уникнути дублювання коду, найкраще винести логіку підключення до СУБД в окремий PHP-файл, який потім можна підключати в усі потрібні скрипти.

У лістингу 2.1 показано приклад PHP-файлу, що відповідає за з’єднання з базою даних. Надалі ми будемо включати його у всі інші скрипти за допомогою команди:

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

Лістинг 2.1. Файл connect.php

```
<?php
$user = "root";
$pass = "";
$db = "students";

mysqli_connect("localhost" $user, $pass)
  or die("Could not connect: ".mysqli_error());
@mysqli_query('CREATE DATABASE $db');
mysqli_select_db($db)
```

```
  or die("Could not select database: ".mysqli_error());
?>
```

Рекомендується всюди, де це можливо, використовувати апострофи для обмеження рядків, що містять SQL-команди. Це запобігає випадковій інтерполяції змінних (тобто їх автоматичній заміні на значення), що підвищує безпеку скриптів.

Перш ніж рухатися далі, варто уважно розглянути параметри підключення до сервера. localhost — типовий варіант для більшості хостингів, який відповідає локальному комп'ютеру. Натомість ім'я користувача root і порожній пароль не можна вважати «стандартними», хоча вони часто використовуються під час налаштування середовища для розробки.

Користувач root має повні права на MySQL-сервері, включно з можливістю створення нових баз даних — що ми і використовуємо у цьому прикладі.

Мова запитів СУБД MySQL

Усі запити до бази даних надсилаються за допомогою єдиної функції — mysqli_query(). Запит передається цій функції як рядковий параметр. При цьому рядок може бути багаторядковим і містити символи переведення рядка. MySQL дозволяє використовувати будь-яку кількість пробілів, табуляцій або перенесень рядків у тих місцях, де допустимий звичайний пробіл — у цьому плані SQL подібна до PHP та багатьох інших мов програмування.

Можливості SQL надзвичайно широкі: вона дозволяє створювати дуже складні та гнучкі запити. Написання SQL-запитів можна порівняти з мистецтвом, і часто саме досвід розробника визначає, наскільки ефективно і швидко буде працювати запит.

Нижче наведено стислий перелік найбільш уживаних SQL-команд у MySQL.

***INSERT:* вставка запису в таблицю**

```
INSERT INTO Ім'яТаблиці(ім'яПоля1, ім'яПоля2, …) VALUES
('зн1', 'зн2',...)
```

Цей запит додає до таблиці Ім’яТаблиці новий запис, у якому зазначені поля ім’яПоляN отримують відповідні значення знN. Поля, що не згадані у запиті, залишаються без значення або набувають значення за замовчуванням (якщо таке задано).

У MySQL також підтримується альтернативний варіант синтаксису:

```
INSERT INTO Ім’яТаблиці SET ім’яПоля1 = 'зн1', ім’яПоля2= 'зн2', ...
```

На практиці цей формат часто виявляється зручнішим за класичний, особливо коли потрібно вставити значення у велику кількість полів.

***DELETE:* видалення записів**

```
DELETE FROM Ім’яТаблиці WHERE вираз
```

Команда видаляє всі записи з таблиці Ім’яТаблиці, для яких умова, задана у виразі, є істинною:

```
DELETE FROM movies WHERE genre_id=10 AND duration > 60
```

***SELECT:* пошук і вибірка записів**

```
SELECT  *   FROM Ім’яТаблиці
WHERE  вираз [ORDER BY ім’яПоля1 [DESC] ...]
```

Команда SELECT є однією з найважливіших і найскладніших в SQL, призначеною для вибірки даних. Хоча вона має безліч варіацій для складних обчислень, групування та об'єднання даних з кількох таблиць (join-запити), ми розглянемо її найпростіший варіант. Ця форма команди SELECT використовується для пошуку всіх записів у таблиці Ім’яТаблиці, які відповідають умові, заданій у виразі.

Якщо знайдено декілька записів, ви можете впорядкувати їх за допомогою оператора ORDER BY. За ним вказується ім'я поля, за яким потрібно відсортувати результати. Якщо ви хочете зворотний порядок (від більшого до меншого або за алфавітом у зворотному напрямку), додайте ключове слово DESC. Ви також можете сортувати за декількома полями, перерахувавши їх через кому.

Не варто припускати, що записи будуть йти в певному порядку без явного використання ORDER BY. Хоча сортування може здатися операцією, що уповільнює запит, насправді SQL-сервери дуже ефективно оптимізують цей

процес. Швидке сортування є одним з пріоритетних завдань при розробці SQL-серверів.

Отримання числа записів, що задовольняють виразу

Щоб отримати кількість записів, які відповідають певній умові в MySQL, часто використовують функцію COUNT(*). Ось як це працює:

```
SELECT COUNT(*)
FROM Ім'яТаблиці
WHERE вираз
```

Цей запит поверне єдине число — загальну кількість записів у таблиці Ім'яТаблиці, що задовольняють умові, вказаній у виразі.

Набуття унікальних значень стовпців

При роботі з базами даних, часто виникає потреба дізнатися, які унікальні значення містяться в певному стовпці таблиці. Нехай маємо таблицю з інформацією про людей, де є поля city (місто) та age (вік). Якщо потрібно дізнатися, в яких містах мешкають усі люди віком від 25 років, внесені в таблицю, ми можемо скористатися таким запитом:

```
SELECT DISTINCT city
from Ім'яТаблиці
WHERE age >= 25
```

Цей запит поверне результат, що складатиметься з одного стовпця, який міститиме лише унікальні назви країн, де проживають люди віком 30 років і старше. Як саме отримати та обробити цей результат, ми розглянемо далі.

***UPDATE:* оновлення записів**

```
UPDATE Ім'яТаблиці SET (ім'яПоля1 = 'зн1', ім'яПоля2 =
'зн2', ...)
WHERE вираз
```

Ця команда працює так: в таблиці Ім'яТаблиці вона знаходить всі записи, які відповідають умові, заданій у виразі; для кожного знайденого запису вказані поля (наприклад, ім'яПоля1, ім'яПоля2) оновлюються на нові значення; всі інші поля в цих записах залишаються без змін. Команда UPDATE особливо корисна, коли вам потрібно змінити лише певні поля в записі, а не оновлювати весь запис цілком.

Коментарі

Як і більшість мов програмування, SQL дозволяє додавати до запитів коментарі — текст, який ігнорується під час виконання команди:

Типи коментарів в SQL:

Однорядкові коментарі:

- Стандартний SQL: Починаються з подвійного дефіса (--).

```
-- Це однорядковий коментар
SELECT * FROM Users; -- Обираємо всі дані з таблиці Users
```

- MySQL-специфічні: Починаються з символу решітки (#).

```
# Це також однорядковий коментар у MySQL
SELECT * FROM Products; # Обираємо всі продукти
```

Багаторядкові коментарі:

- Починаються з /* і закінчуються */.

```
/*
Це коментар,
що може займати
кілька рядків.
*/
SELECT name, email FROM Customers;
```

Для складних запитів дуже рекомендується додавати коментарі, щоб пояснити логіку та призначення команди. Це значно покращує читабельність коду та полегшує його розуміння іншими розробниками (або вами самими у майбутньому).

Отримання результату

Припустимо, ми виконуємо такий запит до бази даних MySQL:

```
$result = mysqli_query("SELECT * FROM students WHERE
name != '' ORDER BY id") or die(mysqli_error();
```

Після успішного виконання запиту і отримання ідентифікатора результуючого набору даних (result-set identifier) у змінній $result, ми можемо почати працювати з отриманими даними.

Хоча SQL-таблиця також складається із записів (рядків) та полів (комірок), є важлива відмінність: у SQL-таблицях записи та поля не впорядковані. Ви не можете просто звернутися до них за номером, натомість для вибірки потрібен

логічний WHERE-вираз. Натомість, результуючий набір даних завжди є впорядкованим масивом записів, тобто масивом масивів комірок.

Кількість рядків, що увійшли до результуючого набору даних, можна дізнатися за допомогою функції mysqli_num_rows(). Наприклад, якщо у нашому запиті з таблиці students ми отримали 10 записів, де поле name не порожнє, то змінна $result буде містити посилання на ці 10 "рядків".

аючи цей результат, ми можемо перенести будь-який з цих рядків у нашу PHP-програму за допомогою спеціальних функцій, які будуть розглянуті далі.

Перетворення result-set в двовимірний масив

Оскільки результуючий набір даних MySQL схожий на двовимірний масив PHP, ви можете працювати з ним подібним чином. Розробники PHP передбачили зручний спосіб отримання результатів запиту, який нагадує роботу з файлами: існує поняття поточного запису, і кожна наступна операція читання переміщує цей покажчик на одну позицію вперед. Ви також можете встановити покажчик на будь-який конкретний запис.

Розглянемо приклад, який виводить вміст таблиці у браузер (лістинг 2.2), а потім детально опишемо використані функції.

Лістинг 2.2. Вивід таблиці студентів

```
<?php
require_once "connect.php";
$r = mysqli_query('SELECT * FROM students ORDER BY id')
  or die(mysqli_error());
for ($data=array(); $row=mysqli_fetch_assoc($r); $data[]=$row);
echo "<pre>"; print_r($data); echo "</pre>";
?>
```

В результаті виконання цього коду в масиві $data буде збережено список асоціативних масивів. Кожен такий внутрішній масив складається з пар "ім'я поля" => "значення". Наприклад, якщо у вашій таблиці students є поля id та name, то ви отримаєте щось на зразок: [['id' => 1, 'name' => 'John'], ['id' => 1, 'name' => 'Jane']]

Тут використовується ключова функція mysqli_fetch_assoc(), яка отримує черговий рядок з результуючого набору даних.

```
array mysqli_fetch_assoc(int $result)
```

Ця функція повертає асоціативний масив, що містить значення полів поточного рядка результату $result. Якщо покажчик поточної позиції досяг кінця набору даних (тобто рядки закінчилися), функція повертає false.

Після кожного виклику mysqli_fetch_assoc(), покажчик поточної позиції автоматично зміщується до наступного запису, дозволяючи вам послідовно обробляти всі рядки результату.

***AS:* перейменування полів**

Ключі в результуючих масивах, які ви отримуєте після запиту SELECT, зазвичай відповідають назвам полів у вихідній таблиці. Однак, SQL дозволяє перейменовувати поля у результуючому наборі даних, що ще раз підкреслює їхню відмінність від оригінальних полів таблиці. Для цього використовується ключове слово AS:

```
$r = mysqli_query('SELECT id AS number, name AS last_name FROM students);
for ($data=array(); $row=mysqli_fetch_assoc($r); $data[]=$row);
```

У цьому прикладі, після виконання запиту, масив $data міститиме асоціативні масиви, що відповідають рядкам з таблиці students. Проте, замість ключів id та name, ці масиви матимуть ключі number та last_name відповідно. Відбулося перейменування полів.

За допомогою AS можна також виконувати нескладні обчислення та надавати результату зрозуміле ім'я. Розгляньте такий запит:

```
SELECT id+2 FROM students WHERE name=''
```

Якщо виконати цей запит, ім'я комірки в результуючому наборі даних буде "id+2", що не дуже зручно. Щоб уникнути цього, на практиці зазвичай застосовують AS:

```
SELECT id+2 AS value FROM students WHERE name=''
```

Тепер перша комірка в першому рядку результату матиме зручне ім'я value. Це значно покращує читабельність та зручність роботи з даними.

***LIMIT:* обмеження вибірки засобами SQL**

Розглянемо ще одну корисну можливість команди SELECT — оператор LIMIT, який дозволяє обмежити кількість рядків у результуючому наборі даних.

Він додається в кінці запиту і дає змогу вибирати не всі записи, що задовольняють умову WHERE, а лише певну їх кількість. Ось приклад запиту з LIMIT:

```
SELECT * FROM students WHERE name!='' LIMIT 3, 20
```

Ця команда включить до результуючого набору даних не всі записи з непорожнім іменем, а лише 10 записів, починаючи з третього (пам'ятайте, нумерація починається з 0, тому 2 означає пропустити перші два записи).

Загальний вид запиту

```
SELECT * FROM students WHERE name!='' LIMIT зсув, кількість
```

Або, якщо зсув рівний 0, а потрібно тільки обмежити число записів:

```
SELECT * FROM students WHERE name!='' LIMIT кількість
```

Вибірка рядка у вигляді списку

Робота з асоціативними масивами за допомогою mysqli_fetch_assoc() зазвичай дуже зручна, і ви, скоріш за все, будете використовувати її найчастіше. Однак, PHP пропонує додаткову функцію, яка працює трохи швидше, оскільки не дбає про імена полів, а витягує їх значення у звичайний індексований список.

```
list mysqli_fetch_row(int $result)
```

Може виникнути питання: навіщо використовувати числові індекси, коли набагато зручніше звертатися до значень за іменами полів (mysqli_fetch_assoc()) або використовувати AS для перейменування?

Відповідь полягає в тому, що в результаті вибірки можуть бути присутні поля з однаковими іменами, але з різними індексами. Це може трапитися, коли ви вибираєте дані одночасно з декількох таблиць (SQL це дозволяє). У простих додатках таке трапляється нечасто.

Параметри результату

```
int nysql_num_rows(int $result)
```

Функція mysqli_num_rows() повертає кількість рядків у результуючому наборі даних. Простіше кажучи, вона дозволяє дізнатися "вертикальний розмір" отриманого вами "двовимірного масиву" результату запиту.

```
int mysqli_num_fields(int $result)
```

Ця функція повертає кількість полів (колонок) в одному рядку результуючого набору даних $result. Таким чином, mysqli_num_fields() дозволяє визначити "горизонтальний розмір" вашого "двовимірного масиву" результату.

Отримання окремого осередку результату

```
mixed mysqli_result(int $result,  int $row,  mixed $field)
```

Ця функція повертає значення поля $field з рядка результату за номером $row. Параметр $field може бути як іменем поля, так і його числовим індексом (позицією стовпця, починаючи з нуля).

Хоча mysqli_result() дозволяє обійти весь результат по одній комірці, робити це не рекомендується, оскільки вона працює доволі повільно. Набагато ефективніше використовувати функції, описані раніше, такі як mysqli_fetch_assoc() або mysqli_fetch_row(), які витягують цілі рядки.

Є один випадок, коли застосування mysqli_result() робить код більш ясним, ніж виклик mysqli_fetch_assoc(): це коли ваш результуючий набір даних складається лише з однієї комірки.

Наприклад, якщо ви хочете отримати загальну кількість рядків у таблиці за допомогою COUNT(*)::

```
$r = mysqli_query('SELECT COUNT (*) FROM students');
echo "Кількість рядків в таблиці: " . mysqli_result ($r, 0, 0);
```

У цьому випадку, mysqli_result($r, 0, 0) дозволяє безпосередньо отримати значення першої (і єдиної) комірки з першого (і єдиного) рядка результату, що робить код лаконічним.

Інформація про результат

Ось опис кількох корисних функцій PHP, призначених для отримання різної інформації про результат запиту MySQL.

```
string mysqli_field_name(int $result, int $field_index)
```

Функція mysqli_field_name() повертає ім'я стовпця з результуючого набору даних за його числовим зміщенням ($field_index). Варто зазначити, що це ім'я буде однаковим для всіх комірок у даному стовпці. Ця функція використовується не дуже часто, оскільки mysqli_fetch_assoc() зазвичай є зручнішою.

Ця функція працює навіть тоді, коли результуючий набір даних не містить жодного рядка. Вона все одно поверне ім'я стовпця, яке могло б бути присутнім у рядках.

```
string mysqli_field_type(int $result, int
$field_offset)
```

Ця функція схожа на mysqli_field_name(), але замість імені, вона повертає тип відповідної колонки в результаті. Наприклад, вона може повернути INT, DOUBLE тощо.

```
int mysqli_field_len(int $result,  int $field_offset)
```

Функція повертає максимальну довжину комірки в результаті $result. Комірка, як завжди, задається її зміщенням. Важливо розуміти, що тут мається на увазі не розмір даних поля в байтах, а та довжина, яка була вказана при його створенні. Наприклад, якщо поле типу VARCHAR було створено з обмеженням VARCHAR(100), ця функція поверне 100.

```
string mysqli_field_flags (int $result, int
$field_offset)
```

Ця функція повертає модифікатори, які були використані при створенні вказаного поля в таблиці. Повернений рядок є набором слів, розділених пробілами. Ви можете перетворити його на масив за допомогою функції explode():

```
$flags = explode(" ", mysqli_field_flags($r,
$field_offset));
```

Модифікатори, які підтримуються MySQL, перераховані в табл. 2.1.

Таблиця 2.1. *Модифікатори типів полів*

Модифікатор	**Опис**
not_null	Поле обов'язково повинне бути проініціалізовано при вставці рядка в таблицю
primary_key	Поле є первинним ключем, тобто ідентифікатором рядка, який буде використаний для посилань на неї

unique_key	Поле повинне бути унікальним
multiple_key	По цьому полю побудований індекс
blob	Поле може містити бінарний блок даних, який ніяк не інтерпритується
unsigned	Поле містить беззнакові числа
zerofill	Використовувати символи з нульовим кодом замість пропусків
binary	Поле містить бінарні дані
enum	Поле містить елемент перерахування, тобто тільки один елемент з декількох можливих
auto_increment	Це поле автоматично нумерується. Проставляється MYSQL при додаванні новому запису так, щоб в таблиці ніколи не утворювалося декількох рядків з однаковим значенням цього поля
timestamp	У полі динамічно проставляється поточний час при додаванні або зміні запису

Завдання

Виконати запити до основної таблиці у відповідності до обраної предметної області для вибірки, добавлення, редагування та видалення даних.

Методичні рекомендації

Вибірка даних, добавлення, редагування та видалення даних із бази даних спроектованої у попередньому розділі

Лістинг 1. Підключення

```
mysqli_connect('localhost', 'root', '');
mysqli_select_db('stud');
```

2. Вибірка даних

Лістинг 2. Вибірка даних із таблиці students

```
     $res = mysqli_query("SELECT * FROM students");

     while ($row = mysqli_fetch_object($res)){
          echo $row->surname." ". $row->name." ". $row-
>rating."<br/>";
     }
     mysqli_query("INSERT INTO students SET name = 'John',
surname = 'Smith', age = 24");
     $student_id = mysqli_insert_id();
     mysqli_query("UPDATE students SET age = 23 WHERE id = $
student_id");
     mysqli_query("DELETE FROM students WHERE id = $
student_id");
```

Іванов Петро 15
Петров Іван 25
Пиріжков Сергій 36
Сидорова Іванка 85
Пончик Аня 63
Пупкін Андрій 91

Рис.2. Скріншот

Контрольні запитання

1. Опишіть конструкцію мови SQL для видалення даних?
2. Яка функція PHP використовується для встановлення з'єднання з сервером MySQL, і які параметри вона приймає?
3. Яке значення повертає функція mysqli_query() у разі виконання запиту типу SELECT?
4. Як за допомогою PHP-функцій mysqli_fetch_assoc() і mysqli_query() отримати всі результати SQL-запиту у вигляді масиву?
5. Яку роль відіграє оператор ORDER BY у запитах SELECT, і чому не слід покладатися на "натуральний" порядок записів у таблиці?

3. Вивід даних із фільтруванням та сортуванням

Мета роботи: навчитись створювати та використовувати шаблоні затор для розділення PHP-коду та HTML.

Теоретичні відомості

Інформація про таблиці і поля

Розглянемо дві важливі функції, які спрощують взаємодію з таблицями в базах даних MySQL.

```
int mysqli_list_fields (string $dbname, string $tblname
[,int $linkt])
```

Функція mysqli_list_fields() надає деталі про стовпці (поля) зазначеної таблиці ($tblname) у вибраній базі даних ($dbname). Якщо ви вкажете ідентифікатор з'єднання ($linkt), функція використовуватиме його; в іншому випадку буде застосовано останнє відкрите з'єднання.

Результатом виконання функції є ідентифікатор результату. Цей ідентифікатор можна обробити за допомогою стандартних функцій або використовувати спеціальні функції mysqli_field_flags(), mysqli_field_len(), mysqli_field_name() та mysqli_field_type() для отримання конкретних даних про поля. У разі помилки функція повертає -1, а текст помилки можна отримати звичним способом.

```
int mysqli_list_tables(string $database [,int
$link_identifier])
```

Функція mysqli_list_tables() повертає ідентифікатор результату, що містить список усіх таблиць у вказаній базі даних ($database). Цей результат є одноколонковим, і для отримання імен таблиць можна використовувати функцію mysqli_result() з номером колонки 0.

Використання mysqli_list_tables() (або, що краще, SQL-запиту SHOW TABLES) є дуже корисним для оптимізації скриптів, які під час свого виконання створюють багато таблиць. Замість того, щоб щоразу намагатися створити таблицю за допомогою CREATE TABLE IF NOT EXISTS, що призводить до великої кількості запитів до бази даних.

AUTOINCREMENT: унікальні ідентифікатори

У таблицях баз даних, де зберігається безліч записів, часто виникає потреба однозначно ідентифікувати кожен рядок. Розгляньмо приклад таблиці з інформацією про людей. Якщо ми покладаємося лише на такі поля, як прізвище чи ім'я, ми зіткнемося з проблемами однофамільців чи тезок, що унеможливлює точний пошук конкретної особи. Це створює незручності та потенційну плутанину.

Щоб уникнути подібних непорозумінь, до таблиці додають допоміжне поле, зазвичай назване id. Це поле є унікальним для кожного запису, дозволяючи швидко і точно знайти необхідні дані. Крім того, наявність такого ідентифікатора відкриває додаткові можливості. Наприклад, якщо потрібно зафіксувати родинні зв'язки, можна додати поле parent_id, яке зберігатиме id батька, пов'язуючи записи між собою. Цей підхід є надзвичайно зручним та ефективним.

Коли ми додаємо новий запис до таблиці, виникає питання: як автоматично призначити йому унікальний id? Ми не знаємо, які ідентифікатори є "вільними". Можна було б спробувати використовувати поточний час у секундах, але існує ризик конфлікту, якщо одночасно додаються кілька записів. Інший варіант — взяти максимальний id з таблиці, збільшити його на одиницю та призначити новому запису. Однак і тут немає гарантії унікальності, адже інший адміністратор чи процес може виконати ту ж операцію до того, як ваш запис буде додано.

Саме для вирішення цієї проблеми в MySQL існує механізм AUTO_INCREMENT. При створенні таблиці ми можемо оголосити одне з її полів (наприклад, id) наступним чином:

```
id INT AUTO INCREMENT PRIMARY KEY
```

Цей синтаксис, хоч і трохи довший, повністю себе виправдовує! Тепер будь-яка операція INSERT автоматично призначить полю id у доданому записі унікальне значення. MySQL гарантує цю унікальність. У найпростішому випадку, система просто збільшить внутрішній лічильник, глобальний для всієї таблиці, на одиницю та запише його нове значення у відповідне поле запису. При цьому

повністю виключаються проблеми зі спільним доступом, які могли б виникнути при використанні таких методів.

Після виконання операції INSERT часто виникає потреба отримати щойно згенерований AUTO_INCREMENT ідентифікатор. Для цього призначена функція mysqli_insert_id():

```
int mysqli_insert_id([int $link_identifier])
```

Ця функція повертає ідентифікатор запису, який було згенеровано безпосередньо перед її викликом, для поля з AUTO_INCREMENT. Її варто викликати одразу після виконання команди INSERT.

```
mysqli_query("INSERT INTO sudents(name) values('John Smith')");
$id = mysqli_insert_id();
```

Тепер, використовуючи змінну $id, ми можемо звернутися до щойно вставленого запису:

```
$r = mysqli_query("SELECT * FROM students WHERE id=$id"); $row = mysqli_fetch_array($r);
```

Отримання ідентифікатора до вставки

Іноді виникає питання, чи можна отримати унікальний ідентифікатор запису ще до того, як він буде вставлений у таблицю. Відповідь на це питання — категоричне "ні". Причина проста: до моменту вставки запису в базу даних його ідентифікатора просто не існує.

Уявіть, що вам якимось чином вдалося "вирахувати" ID запису до його фактичної вставки. Що станеться, якщо між моментом "отримання" цього ID та моментом його фактичної вставки інший сценарій (або користувач) виконає свою команду INSERT? Ця інша операція може використатти ваш передбачуваний ID, зробивши його вже не унікальним для вашого запису. У результаті ви отримаєте недостовірні дані, і ваша логіка роботи з базою даних буде порушена.

Якщо вам потрібно, щоб ідентифікатор запису був вказаний у ній самій (наприклад, у полі self_id, що посилається на id цього ж запису), доведеться вдатися до використання двох окремих запитів:

```
$name = "John Smith";
mysqli_query("INSERT INTO table SET name='Sname'");
$id = mysqli_insert_id();
mysqli_query("UPDATE table SET self_id=id WHERE
id=$id");
```

Перший запит INSERT додає новий запис до таблиці. Поле self_id на цьому етапі залишається не встановленим (або встановленим у значення за замовчуванням). mysqli_insert_id() після першого запиту повертає унікальний AUTO_INCREMENT ідентифікатор, який MySQL щойно згенерував для доданого рядка. Другий запит UPDATE використовує отриманий $id, щоб оновити поле self_id у щойно вставленому записі, присвоюючи йому його ж власний id.

Цей підхід гарантує коректність і уникнення конфліктів, оскільки ви працюєте з уже існуючим, згенерованим ідентифікатором.

Індекси і продуктивність БД

Системи управління базами даних (СУБД) розробляються з розрахунком на швидке вибірку записів, які відповідають певним критеріям, навіть коли база містить мільйони елементів. Але як це працює на практиці без оптимізації?

Уявімо, що у вашій таблиці є мільйон записів про людей, кожен з яких має унікальний ідентифікатор (id). Якщо вам потрібно знайти інформацію про одну конкретну особу, скажімо, з id = 136, ви виконаєте наступний SQL-запит:

```
SELECT * FROM students WHERE id=136
```

Без додаткових оптимізацій, MySQL, як і будь-яка інша СУБД, змушена буде переглядати кожен запис у таблиці по черзі, поки не знайде той, у якого id дорівнює 136. Це відбувається тому, що записи в таблиці, за замовчуванням, не впорядковані.

У найгіршому випадку, якщо шуканий запис знаходиться в самому кінці таблиці, або його взагалі немає, СУБД доведеться перевірити всі мільйон записів. У середньому, для пошуку одного запису таким чином знадобиться близько півмільйона операцій. Це призводить до значного сповільнення, особливо для великих баз даних.

Саме тут на допомогу приходять індекси, які дозволяють СУБД швидко знаходити потрібні дані, не переглядаючи всю таблицю.

Як "працюють" індекси в базах даних

Якщо ви проведете експеримент, то помітите, що вибірка одного запису з мільйонної таблиці відбувається миттєво, а не за півмільйона операцій, як можна було б очікувати. Це означає, що MySQL не переглядає кожен запис послідовно, а якимось чином одразу знаходить потрібний. Як же це відбувається?

Уявіть, що ви шукаєте опис певної функції в товстій книзі. Ви не будете гортати сторінку за сторінкою. Натомість, ви відкриєте алфавітний покажчик наприкінці книги, знайдете там назву функції, а потім перейдете на вказану сторінку. Секрет швидкості полягає в тому, що покажчик відсортований. Вам навіть не потрібно читати його повністю, щоб знайти потрібний запис.

MySQL працює за схожим принципом. Для кожної таблиці сервер створює своєрідний "покажчик", який у термінах баз даних називається індексом. Завдяки індексу, MySQL може миттєво визначити точне розташування потрібного запису у своїх службових файлах, якщо відоме значення поля, за яким побудовано індекс. Залишається лише зчитати дані з цього місця (подібно до того, як ви відкриваєте потрібну сторінку в книзі).

Для однієї таблиці може існувати не один, а кілька індексів.

- PRIMARY KEY (первинний ключ): Індекс автоматично створюється для поля, яке визначено як PRIMARY KEY (наприклад, наше поле id з AUTO_INCREMENT PRIMARY KEY). Це забезпечує миттєву вибірку за id навіть у таблицях, що містять десятки мільйонів записів.
- Додаткові індекси: При створенні таблиці ви можете вказати додаткові індекси для інших полів. Наприклад, створивши індекс по полю name у таблиці students, ви прискорите виконання запитів типу SELECT * FROM students WHERE name='John Smith', роблячи їх майже миттєвими.
- Складені індекси: Можна створювати також складені індекси, які охоплюють одразу кілька полів у таблиці. Наприклад, якщо в таблиці

students є поля firstname (ім'я) та age (вік), додавання індексу за парою (firstname, age) дозволить швидко витягувати записи за запитами, що містять вирази на кшталт WHERE firstname = 'John' AND age = 25.

Індекси є ключовим елементом для забезпечення високої продуктивності баз даних, особливо при роботі з великими обсягами даних.

Недоліки індексів

Хоча індекси значно прискорюють пошук даних, вони, як і будь-яка інша технологія, мають свої недоліки. Їх можна розділити на дві основні категорії:

- Збільшення дискового простору: Наявність індексів вимагає додаткового місця на диску. Загалом, можна очікувати, що додавання індексу на кожне поле таблиці може подвоїти об'єм, який вона займає. Уявіть, на що перетворився б покажчик цієї книги, якби в ньому були посилання на кожне слово! Цей додатковий об'єм потрібен для зберігання самих індексних структур.
- Уповільнення операцій запису: Індекси дещо уповільнюють операції вставки (INSERT), оновлення (UPDATE) та видалення (DELETE) записів у таблиці. Чому? Тому що при кожній зміні даних у таблиці СУБД також повинна оновити відповідні індекси. Це схоже на те, якби редактору книги доводилося щоразу переробляти алфавітний покажчик при внесенні змін до тексту – це дуже трудомістка робота.

З огляду на ці недоліки, важливо розумно підходити до створення індексів:

- Використовуйте індекси для полів, які беруть участь у WHERE-предикатах: Створюйте індекси для тих полів, які часто використовуються в умовах фільтрації (WHERE) ваших SQL-запитів. Це саме ті поля, за якими СУБД буде шукати дані, і індекси допоможуть їй зробити це швидко.
- Уникайте індексів на непотрібних полях: Наприклад, немає сенсу створювати індекс на текстовому полі гостьової книги, якщо воно ніколи не використовується для порівнянь при вибірці. Це поле не впливає на результуючий набір даних, тому індекс на ньому лише

займатиме місце і сповільнюватиме запис без жодної користі.

- Розгляньте відмову від індексів для таблиць з частими оновленнями та рідкими вибірками: Якщо таблиця оновлюється дуже часто, а вибірки з неї відбуваються рідко (наприклад, таблиця, що зберігає всі HTTP-запити до поточного сервера), можливо, варто взагалі відмовитися від індексів. Це значно збільшить продуктивність операцій запису та зменшить об'єм дискового простору, займаного базою даних.

Завдання

Реалізувати відображення даних із основної таблиці у відповідності до обраної предметної області із елементами інтерфейсу для фільтрування та сортування даних.

Методичні рекомендації

Для зручності перегляду інформації вартує добавити фільтрування та сортування. Для цього скористаємось можливостями SQL, командами WHERE та ORDER BY, також для зручності підготовимо один набір даних, який включає дані з двох таблиць за допомогою конструкції JOIN.

Лістинг 3.1. Вибірка даних із таблиці students із фільтруванням та сортуванням

```
$SQL = "
     SELECT S.*, G.* FROM `students` S
     LEFT JOIN `groups` G ON S.group_id = G.group_id
     WHERE 1=1 %s
     ORDER BY %s
";
$where = '';
if ($fgroup)   $where .= " AND S.group_id = $fgroup ";
if ($fkurs)    $where .= " AND G.kurs = $fkurs ";
if ($frating1) $where .= " AND S.rating >= $frating1 ";
if ($frating2) $where .= " AND S.rating <= $frating2 ";

switch ($sort) {
case 1: $order_by = 'S.rating';
          break;
case 2: $order_by = 'S.rating DESC';
          break;
```

```
     case 3: $order_by = 'G.kurs';
               break;
     case 4: $order_by = 'G.kurs DESC';
               break;
     case 5: $order_by = 'S.surname';
               break;
     case 6: $order_by = 'S.surname DESC';
               break;
     default: $order_by = 'S.id';
     }
     $SQL = sprintf($SQL, $where, $order_by);
     $res = mysqli_query($SQL);
     $astat = Array('m' => 'чоловіча', 'f' => 'жіноча');
     echo "<table border=1>";
     echo
"<tr><td>Курс</td><td>Група</td><td>Прізвище</td><td>Ім’я</td><t
d>Стать</td><td>Рейтинг</td></tr>\n";
     while ($row = mysqli_fetch_object($res)){
          printf("<tr><td>%d</td><td>%s</td><td>%s</td><td>%s
</td><td>%s </td><td>%d</td></tr>\n",
                    $row->kurs,
                    $row->group_name,
                    $row->surname,
                    $row->name,
                    $astat[$row->stat],
                    $row->rating
          );
     }
     echo "</table>";
```

Курс	Група	Прізвище	Ім’я	Стать	Рейтинг
1	КН-15	Іванов	Петро	чоловіча	15
1	КН-15	Петров	Іван	чоловіча	25
1	І-14	Пиріжков	Сергій	чоловіча	36
1	І-14	Сидорова	Іванка	жіноча	85
2	КН-26	Пончик	Аня	жіноча	63
2	КН-26	Пупкін	Андрій	чоловіча	91

Рис.3.1. Скріншот

Формування елементів інтерфейсу для управління користувачами, фільтруванням та сортуванням таблиці:

1) для фільтрування за курсом та групою – список вибору (<select></select>)
2) для фільтрування за рейтингом від … до … – поле вводу(<input type=”text”>)

3) для сортування – список вибору з елементами: курс за зростанням, курс за спаданням, група за зростанням, група за спаданням, прізвище за зростанням, прізвище за спаданням

Лістинг 3.2. Формування html-форми

```
<?php
     $res = mysqli_query("SELECT * FROM groups");
     while ($row = mysqli_fetch_object($res)){
          $grupa_options .= "<option value={$row-
>group_id}>{$row->group_name}</option>";
     }
?>
<form method="get" action="index.php">
<label for="kurs">Курс</label>
<select name="kurs">
<option value="" ></option>
<option value=1>1</option>
<option value=2>2</option>
<option value=3>3</option>
<option value=4>4</option>
<option value=5>5</option>
<option value=6>6</option>
</select>
<label for="group">Група</label>
<select name="group">
<option value="" ></option>
<?php echo $grupa_options; ?>
</select>
Рейтинг від <input name="rating1" size=10> до <input
name="rating2" size=10>
<br/>
Сортувати за
<select name="sort">
<option value="" ></option>
<option value=1>курс за зростанням</option>
<option value=2>курс за спаданням</option>
<option value=3>група за зростанням</option>
<option value=4>група за спаданням </option>
<option value=5>прізвище за зростанням</option>
<option value=6>прізвище за спаданням</option>
</select>
<input type=submit value=Go!>
</form>
<?php
?>
```

Лістинг 3.3. Опрацювання даних отриманих методом GET

```
$fkurs = intval($_GET['kurs']);
$fgroup = intval($_GET['group']);
$frating1 = intval($_GET['rating1']);
```

```
$frating2 = intval($_GET['rating2']);
$sort = intval($_GET['sort']);
```

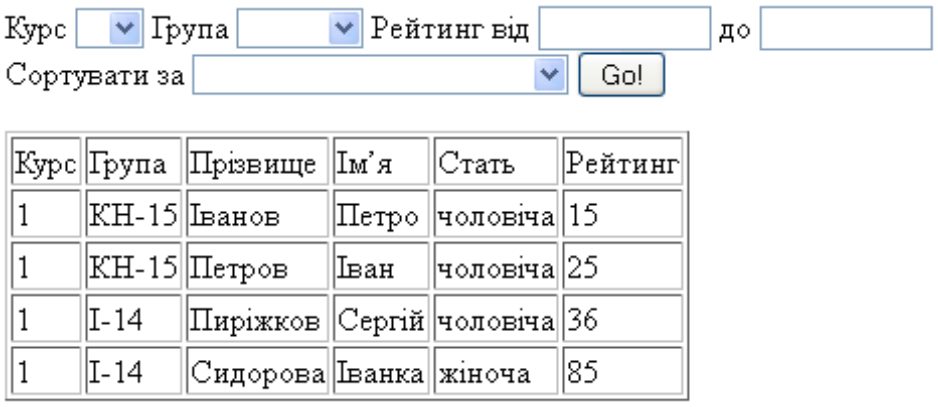


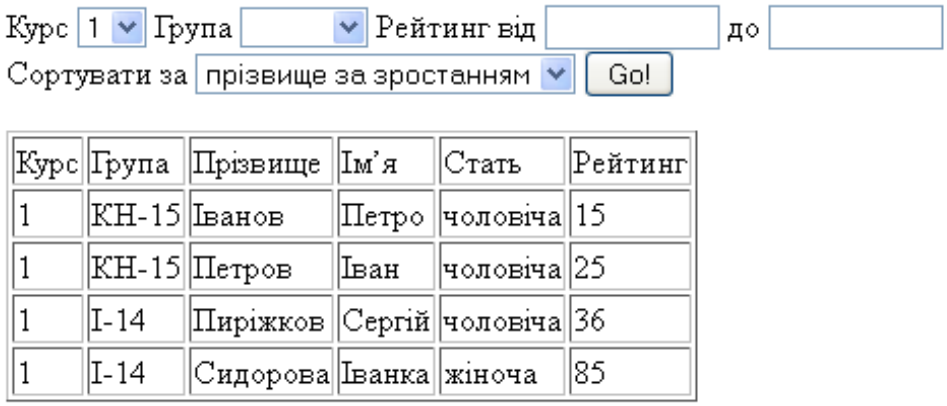
Рис.4. Скріншот

На рисунку 4 відображається таблиця із студентами з 1-го курсу та відсортовані за прізвищем від `а` до `я`, проте у формі цього не видно, код наступного лістинга виправляє цю незручність.

Лістинг 3.4. Формування html-форми

```
     $res = mysqli_query("SELECT * FROM groups");
     while ($row = mysqli_fetch_object($res)){
          $selected = ($row->group_id == $fgroup) ? 'selected' :
'';
          $grupa_options .= "<option value={$row->group_id}
{$selected}>{$row->group_name}</option>\n";
     }

     $akurs = ['1' => 1, '2' => 2, '3' => 3, '4' => 4, '5' => 5,
'6' => 6];
     foreach ($akurs as $kurs_key => $kurs_value){
          $selected = ($kurs_key == $fkurs) ? 'selected' : '';
          $kurs_options .= "<option value={$kurs_key}
{$selected}>{$kurs_value}</option>\n";
     }

     $asort = ['1' => 'курс за зростанням', '2' => 'курс за
спаданням',
                    '3' => 'група за зростанням', '4' =>
'група за спаданням',
                    '5' => 'прізвище за зростанням', '6' =>
'прізвище за спаданням'];
     foreach ($asort as $sort_key=>$sort_value){
          $selected = ($sort_key == $sort) ? 'selected' : '';
          $sort_options .= "<option value={$sort_key}
{$selected}>{$sort_value}</option>\n";
```

```
     }
?>
<form method="get" action="index.php">
     <label for="kurs">Курс</label>
     <select name="kurs">
          <option value="" ></option>
          <?php echo $kurs_options; ?>
     </select>
     <label for="group">Група</label>
     <select name="group">
          <option value="" ></option>
          <?php echo $grupa_options; ?>
     </select>
     Рейтинг від <input name="rating1" size=10 value="<?php echo
$frating1; ?>"> до <input name="rating2" size=10  value="<?php
echo $frating2; ?>">
     <br/>
     Сортувати за
     <select name="sort">
          <option value="" ></option>
          <?php echo $sort_options; ?>
     </select>
     <input type=submit value=Go!>
</form>
```



Курс	Група	Прізвище	Ім'я	Стать	Рейтинг
1	КН-15	Іванов	Петро	чоловіча	15
1	КН-15	Петров	Іван	чоловіча	25
1	І-14	Пиріжков	Сергій	чоловіча	36
1	І-14	Сидорова	Іванка	жіноча	85

Рис.5. Скріншот

Контрольні запитання

1. Яка структура мови SQL служить для сортування даних?
2. Яку проблему вирішує використання поля AUTO_INCREMENT у таблицях MySQL?
3. Навіщо використовувати функцію mysqli_insert_id() після вставки запису?
4. Які переваги дають індекси в таблицях MySQL і як вони працюють?

4. Внесення змін у базу даних, перевірка вхідних даних.

Мета роботи: навчитись створювати форми для добавлення та редагування даних.

Теоретичні відомості

MYSQL і проблеми безпеки

Запити, які ми надсилаємо до сервера MySQL через PHP, є звичайними текстовими рядками. Досі ми писали команди на кшталт:

```
mysqli_query("INSERT INTO table SET name='$name'");
```

При цьому ми не замислювалися, що змінна $name може містити спеціальні символи, наприклад, апострофи. Розгляньмо, що станеться, якщо $name дорівнюватиме cat's::

```
INSERT  INTO table  SET name='cat's'
```

Ця команда є синтаксично некоректною для MySQL і призведе до помилки під час виконання. СУБД інтерпретуватиме апостроф у cat's як закриття рядка, а наступний символ s буде вважатися нерозпізнаним елементом синтаксису.

Суть проблеми

Ми вже згадували, що необроблений ввід може призвести до синтаксичних помилок у SQL-запитах. Але може бути й гірше. Розгляньмо такий запит:

```
mysqli_query("DELETE FROM table WHERE name='$name'");
```

Уявіть, що змінна $name отримує своє значення з форми, яку заповнив зловмисник. Якщо він введе туди рядок: !' OR 1=1 OR '!', то після підстановки змінної $name у запит, до бази даних надійде наступна команда:

```
DELETE FROM table WHERE name='!' OR 1=1 OR '!'
```

Ви бачите, що цей запит робить? Він видаляє всі записи з таблиці table! Чому? Тому що SQL-вираз 1=1 завжди істинний. Навіть якщо name='!' буде хибним, умова OR 1=1 зробить увесь предикат істинним для кожного запису, що призведе до видалення всієї таблиці.

Екранування спецсимволів

Щоб уникнути синтаксичних помилок і, що важливіше, загроз безпеки, таких як SQL-ін'єкції, необхідно екранувати спеціальні символи у значеннях змінних,

перш ніж передавати їх до SQL-запитів. Екранування полягає у додаванні зворотного слеша (\) перед такими символами.

```
string mysqli_escape_string(string $str)
```

Ця функція працює аналогічно addslashes(), але екранує ширший набір спеціальних символів відповідно до стандарту MySQL. До таких символів належать: нульовий байт (\x00), новий рядок (\n), повернення каретки (\r), одинарна лапка ('), подвійна лапка ("), та символ (\x1A).

Зверніть увагу, що до "жертв" екранування потрапив і символ з нульовим ASCII-кодом. Це означає, що mysqli_escape_string() можна застосовувати не тільки для текстових, але й для бінарних даних. Наприклад, ви можете прочитати вміст GIF-зображення у змінну за допомогою file_get_contents(), а потім вставити його в базу даних, попередньо екранувавши всі спеціальні символи. Після вилучення зображення буде в тому ж вигляді, в якому воно було спочатку.

Важливо розуміти: екранування символів — це лише спосіб забезпечення синтаксично коректних SQL-виразів. Самі дані при цьому не змінюються і зберігаються в базі даних без додаткових слешів, у своєму первісному вигляді. Це як назвати олівець "олівцем" або "pencil" – він все одно залишається олівцем і буде зрозумілий як предмет, а не як слово. Вимова назви реального об'єкта, по суті, є таким же екрануванням, перетворенням в синтаксично коректні слова, адже мова не може оперувати предметами безпосередньо.

Використовуючи mysqli_escape_string(), наш попередній приклад з видаленням виглядатиме так:

```
mysqli_query (
"DELETE FROM table WHERE
name='".mysqli_escape_string($name)."'"
);
```

Як бачимо, цей підхід є довгим, незграбним і непривабливим. Він вимагає ручного екранування кожної змінної, що легко може призвести до помилок і незручностей, особливо у складних запитах. Саме тому сучасна розробка баз

даних надає перевагу підготовленим запитам (prepared statements), які автоматично вирішують ці проблеми, роблячи код чистішим і безпечнішим.

Використання підготовлених запитів

Найбільш надійним способом роботи з плейсхолдерами в MySQLi є використання підготовлених запитів (prepared statements) через функцію mysqli_prepare().

```
mysqli_prepare($link, "SELECT District FROM City WHERE Name=?");
```

Переваги mysqli_prepare():

- Захист від SQL-ін'єкцій: Це головна перевага. mysqli_prepare() автоматично екранує всі спеціальні символи в значеннях, які ви прив'язуєте до плейсхолдерів. Це усуває ризик SQL-ін'єкцій, оскільки дані передаються окремо від SQL-коду, і MySQL ніколи не інтерпретує їх як частину запиту.
- Підвищена продуктивність: Для запитів, які виконуються багаторазово з різними значеннями (наприклад, у циклі), mysqli_prepare() може значно підвищити продуктивність. Запит аналізується і компілюється MySQL лише один раз, а потім багаторазово виконується з новими даними. Це зменшує накладні витрати на парсинг запиту.
- Зручність роботи з типами даних: mysqli_prepare() дозволяє явно вказувати типи даних для значень, що прив'язуються, що допомагає MySQL оптимізувати обробку запиту.
- Чіткість коду: Хоча синтаксис може здатися трохи довшим, він є більш явним і зрозумілим, що сприяє підтримці коду.

Перепишемо наш приклад DELETE FROM table WHERE name='?' за допомогою mysqli_prepare():

```
<?php

// Припустимо, у вас вже є встановлене з'єднання з базою даних $mysqli
```

```
$name = "!' OR 1=1 OR '!"; // Значення від користувача, що
містить потенційно шкідливі символи

// 1. Підготовка запиту
// Створюємо підготовлений запит з плейсхолдером '?'
$stmt = mysqli_prepare($mysqli, "DELETE FROM table WHERE name =
?");

// 2. Прив'язка параметрів
// 's' вказує, що $name є рядковим (string) значенням.
// Інші можливі типи: i (integer), d (double), b (blob)
mysqli_stmt_bind_param($mysqli, "s", $name);

// 3. Виконання запиту
mysqli_stmt_execute($stmt);

// 4. Закриття підготовленого запиту та з'єднання
$stmt->close();
$mysqli->close();

?>
```

Використання mysqli_prepare() є рекомендованою практикою для всіх запитів, які містять користувацький ввід, оскільки це забезпечує найвищий рівень безпеки та оптимізації.

Завдання

Реалізувати форму для добавлення та редагування даних основної таблиці у відповідності до обраної предметної області.

Методичні рекомендації

Для вставка, редагування та видалення даних використовуються конструкції SQL: INSERT, UPDATE та DELETE, а також функція mysqli_query для виконання конструкцій SQL та деяких додаткових функцій для подальшого опрацювання результату виконання функції mysqli_query, такі як: mysqli_insert_id, mysqli_affected_rows

Для вставки та редагування даних користувачем будемо використовувати елементи форму, та урізноманітнимо їх порівняно із попередніми прикладами.

Добавлення даних

Лістинг 4.3. Формування html-форми для вставки даних про студента

```
<?php

    // Формування списку груп для випадаючого списку
```

```
     $res = mysqli_query("SELECT * FROM groups");
     while ($row = mysqli_fetch_object($res)){
          $grupa_options .= "<option value={$row-
>group_id}>{$row->group_name}</option>\n";
     }
?>
<div>
     <?php echo $notice; ?>
</div>
<form method="post" action="">
<table>
<tr>
     <td><label for="group">Група</label></td>
     <td><select name="group" style="width:146px;">
     <option value="" ></option>
     <?php echo $grupa_options; ?>
     </select></td>
</tr>
<tr>
     <td><label for="surname">Прізвище</label></td>
     <td><input name="surname"></td>
</tr>
<tr>
     <td><label for="name">Ім'я</label></td>
     <td><input name="name"></td>
</tr>
<tr>
     <td><label for="stat">Стать</label></td>
     <td><input type=radio name="stat" value="m"> чоловіча <input
type=radio name="stat" value="f"> жіноча</td>
</tr>
<tr>
     <td><label for="rating">Рейтинг</label></td>
     <td><input name="rating"></td>
</tr>
<tr>
     <td colspan=2 align=center><input type="submit"
name="submit"></td>
</tr>
</table>

</form>
```

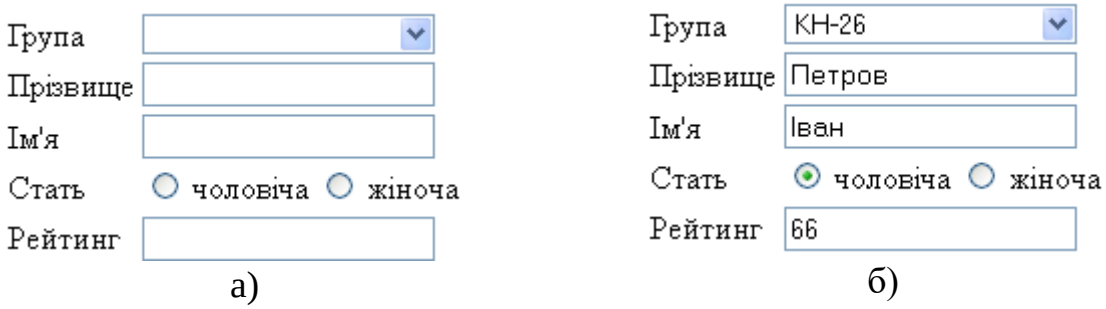


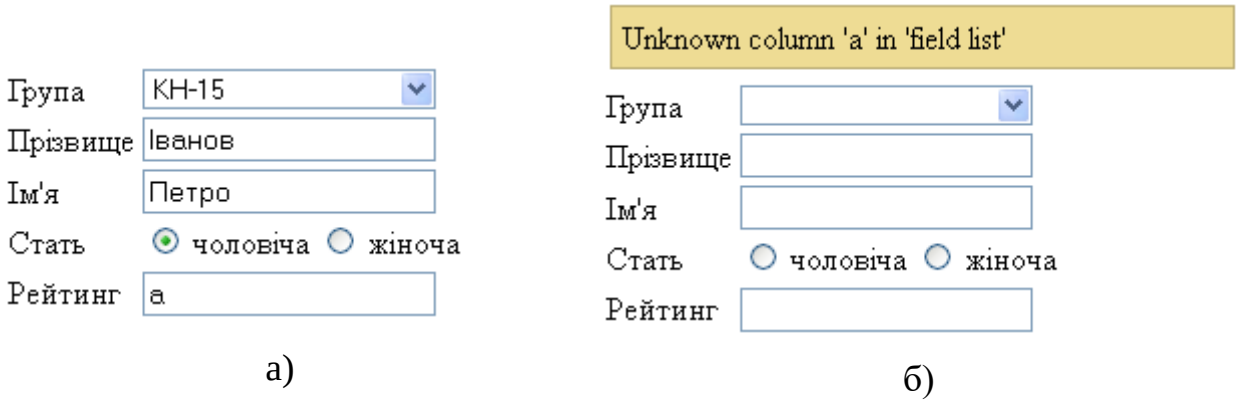
Рис.4.1. Скріншоти а) пустої та б) заповненої форм

На даному етапі натискання кнопки «Відправити запит» приводить до перевантаження сторінки та очищення елементів форми, тобто опрацювання відправлених даних ще не відбувається.

Опрацювання даних, версія 1

Лістинг 4.4. Опрацювання переданих даних

```
     if (isset($_POST['submit'])) { // прийшли дані з нашої форми
          mysqli_query("
               INSERT INTO `students`
               (`group_id`, `surname`, `name`, `stat`, `rating`)
               VALUES
               ({$_POST['group']}, '{$_POST['surname']}',
'{$_POST['name']}', '{$_POST['stat']}', {$_POST['rating']})");

          if (mysqli_errno() != 0) $notice = mysqli_error();
     }
```



а) б)

Рис.4.2. Скріншот. а) невірно заповнена форма б) форма після відправлення даних

Код у лістингу 9 виконує свою основну роботу – вставляє запис у таблицю «`students`», проте якщо користувач ввів невірні дані (рис. 4.2а), ми отримаємо результат зображений на рис. 4.2б. При цьому дані у таблицю вставлені не будуть, помилка в SQL-синтаксисі сформованого запиту до СУБД MySQL, помилковий запит має вигляд: INSERT INTO `students` (`group_id`, `surname`, `name`, `stat`, `rating`) VALUES (1, 'Іванов', 'Петро', 'm', а). В даному прикладі помилка в тому, що у переліку значень не може бути символьних даних не обрамлених знаками «'». Але якщо це виправити, то

буде вже наступна помилка, оскільки поле «rating» цілочисельне і не може приймати значень із стрічки символів.

Опрацювання даних, версія 2

Лістинг 4.5. Опрацювання переданих даних

```
     if (isset($_POST['submit'])) { // прийшли дані з нашої форми
          $SQL = "
               INSERT INTO `students`
               (`group_id`, `surname`, `name`, `stat`, `rating`)
               VALUES
               (%d, '%s', '%s', '%s', %d)";

          $group_id = intval($_POST['group']);

          $surname =
mysqli_real_escape_string($_POST['surname']);
          $name = mysqli_real_escape_string($_POST['name']);
          $stat = mysqli_real_escape_string($_POST['stat']);

          $rating = intval($_POST['rating']);

          $SQL = sprintf($SQL, $group_id, $surname, $name, $stat,
$rating);

          if (mysqli_errno() != 0) $notice = mysqli_error();
     }
```

Код у лістигну 4.5 є більш безпечний ніж попередній, проте тут не перевіряється на допустимість даних. Добавимо наступні вимоги:

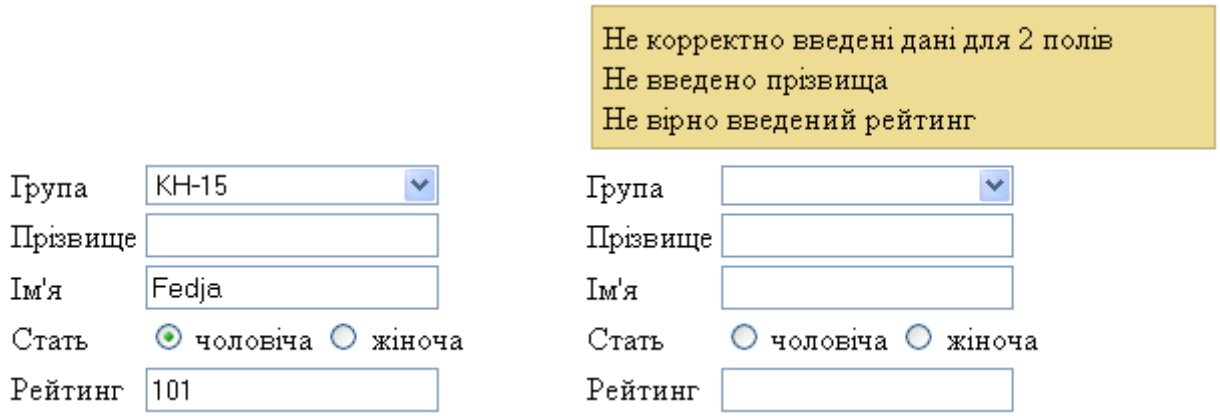
1) Передані дані не повинні бути пустими
2) Код групи повинен існувати у таблиці групи;
3) Прізвище та ім’я повинні складатись із кириличних символів та знаку «-», і їх довжина не повинна перевищувати 30 символів
4) Стать може містити тільки символи «m»,«f».
5) Рейтинг повинен бути в межах від 0 до 100.

Лістинг 4.6. Опрацювання переданих даних із перевіркою на допустимість даних

```
     if (isset($_POST['submit'])) { // прийшли дані з нашої форми
          $SQL = "
               INSERT INTO `students`
               (`group_id`, `surname`, `name`, `stat`, `rating`)
               VALUES
               (%d, '%s', '%s', '%s', %d)";
```

```
          $error_count = 0;
          $errors = Array();

          if (isset($_POST['group']) && !empty($_POST['group']))
{
               $group_id = intval($_POST['group']);
               $res = mysqli_query("SELECT * FROM groups WHERE
group_id = $group_id");
               if (mysqli_num_rows($res) == 0)    {
                    $errors['group'] = 'Такої групи немає';
               }
          } else $errors['group'] = 'Не вибрано групи';

          if (isset($_POST['surname']) &&
!empty($_POST['surname'])) {
               $surname = $_POST['surname'];
               if ( strlen($surname) > 30 ) {
                    $errors['surname'] = 'Введене прізвище
занадто довге';
               } else {
                    if (!preg_match('/[А-ЯІЇЄа-яіїя\']+/',
$surname))
                         $errors['surname'] = 'У введеному
прізвищі є недопустимі символи';
               }
          } else $errors['surname'] = 'Не введено прізвища';

          if (isset($_POST['name']) && !empty($_POST['name'])) {
               $name = $_POST['name'];
               if ( strlen($name) > 30 ) {
                    $errors['name'] = 'Введене ім\'я занадто
довге';
               } else {
                    if (!preg_match('/[А-ЯІЇЄа-яіїя\']+/',
$name))
                         $errors['name'] = 'У введеному імені є
недопустимі символи';
               }
          } else $errors['name'] = 'Не введено імені';

          if (isset($_POST['stat']) && !empty($_POST['stat'])) {
               $stat = $_POST['stat'];
               if (!in_array($stat, Array('f', 'm')))
                    $errors['stat'] = 'Не вірно вказана стать';
          } else $errors['stat'] = 'Не введено статі';

          if (isset($_POST['rating']) &&
!empty($_POST['rating'])) {
               $rating = intval($_POST['rating']);
               if (($rating<0) || ($rating > 100))
                    $errors['rating'] = 'Не вірно введений
рейтинг';
          } else $errors['rating'] = 'Не введено рейтингу';
```

```
          if (count($errors) == 0) {
               $SQL = sprintf($SQL, $group_id,
mysqli_real_escape_string($surname),
mysqli_real_escape_string($name), $stat, $rating);
               mysqli_query($SQL);
               if (mysqli_errno() != 0)  $message = "Помилка
SQL".mysqli_error();
                                        else $message =
"Добавлення даних пройшло успішно";
          } else {
               $message = "Не корректно введені дані для
".count($errors)." полів<br/>\n";
               $message .= implode("<br/>\n", $errors);
          }
     }
```



а) б)

Рис.4.4. Скріншот. а) невірно заповнена форма б) форма після відправлення даних

Опрацювання даних, версія 3

Залишилось добавити два штиха для підвищення зручності користування формою:

1) якщо користувач ввів якісь дані, то не заставляти його вводити їх заново
2) виводити інформацію про помилки біля відповідного поля

Видалимо стрічку $message .= implode("
\n", $errors); з лістингу 4.4, форму приведемо до наступного виду:

Лістинг 4.7. Форма для введення даних

```
<form method="post" action="">
<table>
<tr>
```

```
     <td><label for="group">Група</label></td>
     <td><select name="group" style="width:146px;">
     <option value="" ></option>
     <?php echo $grupa_options; ?>
     </select></td>
     <td><?php if (isset($errors['group'])) echo
$errors['group']; ?></td>
</tr>
<tr>
     <td><label for="surname">Прізвище</label></td>
     <td><input name="surname" value="<?php echo
htmlspecialchars($surname); ?>"></td>
     <td><?php if (isset($errors['surname'])) echo
$errors['surname']; ?></td>
</tr>
<tr>
     <td><label for="name">Ім'я</label></td>
     <td><input name="name" value="<?php echo
htmlspecialchars($name); ?>"></td>
     <td><?php if (isset($errors['name'])) echo $errors['name';
?></td>
</tr>
<tr>
     <td><label for="stat">Стать</label></td>
     <td><input type=radio name="stat" value="m" <?php if ($stat
== 'm') echo "checked"; ?>> чоловіча
          <input type=radio name="stat" value="f" <?php if ($stat
== 'f') echo "checked"; ?>> жіноча</td>
     <td><?php if (isset($errors['stat'])) echo $errors['stat'];
?></td>
</tr>
<tr>
     <td><label for="rating">Рейтинг</label></td>
     <td><input name="rating" value="<?php echo
htmlspecialchars($rating); ?>"></td>
     <td><?php if (isset($errors['rating'])) echo
$errors['rating']; ?></td>
</tr>
<tr>
     <td colspan=3 align=center><input type="submit"
name="submit"></td>
</tr>
</table>
</form>
```

Зауваження. У коді лістингу 4.7 використовується функція htmlspecialchars, це робиться на випадок, якщо користувач введе теги html або символ «”».

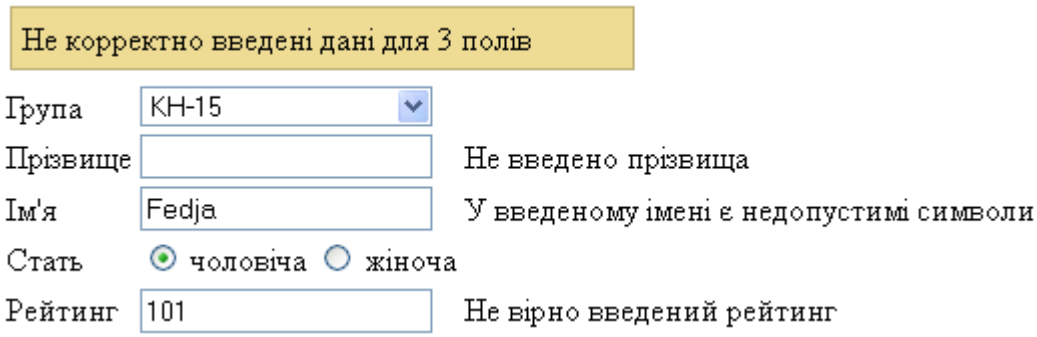


Рис.4.5. Скріншот. форма після відправлення даних

Редагування даних

Для створення програмного коду можна піти двома шляхами:

1) створення окремого файлу для форми та опрацювання даних при редагуванні даних із таблиці;
2) удосконалити попередній код із добавлення даних

Оскільки багато написаного нами програмного коду підійде для редагування даних то ми підемо другим шляхом.

Код запису, який маємо будемо передавати методом **GET** параметром з іменем **id.**

Після вставки даних будемо переходити в режим редагування вставленого запису, а інформацію про ідентифікатор будемо вже передавати за допомогою елементу форми **hidden**.

Лістинг 4.8.

```
     if (isset($_GET['id'])) {
          $id = intval($_GET['id']);
          $res = mysqli_query("SELECT * FROM `students` WHERE id
= $id");
          if (mysqli_num_rows($res) != 0)    {
               $student = mysqli_fetch_object($res);
               $group_id = $student->group_id;
               $surname  = $student->surname;
               $name     = $student->name;
               $stat     = $student->stat;
               $rating   = $student->rating;
          }
     }

     if (isset($_POST['submit'])) { // прийшли дані з нашої форми
          if (isset($_POST['id'])) {
               $id = intval($_POST['id']);
          }
```

```
          if (count($errors) == 0) {
               if (isset($id)) {
                    $SQL = "
                         UPDATE `students`
                         SET
                              `group_id` = %d,
                              `surname`  = '%s',
                              `name`  = '%s',
                              `stat`  = '%s',
                              `rating`  = %d
                         WHERE
                              `id` = %d";

                    $SQL = sprintf($SQL, $group_id,
mysqli_real_escape_string($surname),
mysqli_real_escape_string($name), $stat, $rating, $id);
                    mysqli_query($SQL);
                    if (mysqli_errno() != 0)  $message = "Помилка
при оновленны даних: ".mysqli_error();
                                             else $message =
"Оновлення даних пройшло успішно";
               } else {
                    $SQL = "
                         INSERT INTO `students`
                         (`group_id`, `surname`, `name`, `stat`,
`rating`)
                         VALUES
                         (%d, '%s', '%s', '%s', %d)";

                    $SQL = sprintf($SQL, $group_id,
mysqli_real_escape_string($surname),
mysqli_real_escape_string($name), $stat, $rating);
                    mysqli_query($SQL);
                    if (mysqli_errno() != 0)  $message = "Помилка
при добавленню даних: ".mysqli_error();
                                             else {
                                                  $id =
mysqli_insert_id();
                                                  $message =
"Добавлення даних пройшло успішно";
                                             }
               }
          } else {
               $message = "Не корректно введені дані для
".count($errors)." полів<br/>\n";
          }
     }
```

```
<form method="post" action="">
<?php if (isset($id)) echo "<input type=\"hidden\" name=\"id\"
value=\"{$id}\">\n"; ?>
```

Видалення даних

Ідентифікатор запису будемо передавати також методом $_GET через URI

Лістинг 4.8.

```
     if (isset($_GET['id'])) {
          $id = intval($_GET['id']);
          $res = mysqli_query("SELECT * FROM `students` WHERE id
= $id");
          if (mysqli_num_rows($res) == 0)    {
               echo "Такого запису не існує";
          } else {
               mysqli_query("DELETE FROM `students` WHERE id =
$id");
               if (mysqli_errno() != 0)  echo "Помилка при
видаленні даних: ".mysqli_error();
                                                  else echo
"Видалення даних пройшло успішно";
          }
          }
```

Контрольні запитання

1. Що таке SQL-ін’єкція?
2. Чому слід перевіряти дані введені користувачем?
3. Як індекси допомагають покращити продуктивність запитів до великої таблиці?
4. Назвіть два основні недоліки використання індексів у базах даних.

Література

1. Пасічник В.В., Пасічник О.В., Угрин Д.І. Веб-технології. Львів : “Магнолія 2006”, 2018. 336 с.
2. Васильєв О. Програмування мовою PHP. Київ : Ліра-К, 2022. 368с.
3. Роман Мельник. Програмування веб-застосувань (фронт-енд та бек-енд) Львів : Львівська політехніка, 2018. 248с.
4. Luke Welling, Laura Thomson. PHP and MySQL Web Development. Sams Publishing, 2003. 871с.
5. Marcus Baker, Chris Shiflett, Dagfinn Reiersol. PHP in Action : Objects, Design, Agility – Simon and Schuster, 2007. 552с.
6. Antonio Lopez. Learning PHP 7 - Packt Publishing Ltd., 2016. 363с.
7. George Schlossnagle. Advanced PHP Programming: Developing Large-Scale Web Applications with PHP 5 - Sams, 2007. - 700с.
8. Бородкіна І. Л., Бородкін Г. О. Web-технології та Web-дизайн: застосування мови HTML для створення електронних ресурсів. Київ : Ліра-К, 2020. 212с.
9. Dave W. Mercer. Beginning Php 5 - Wiley India Pvt. Limited, 2009. 507с.
10. Rasmus Lerdorf, Kevin Tatroe, Bob Kaehms, Ric McGredy. Programming PHP – O’Reilly Media, Inc., 2002. 507с.

Додаток 1. Предметні області для виконання лабораторних робіт

1. База тестових завдань – структура тестів, питання, відповіді, статистика.
2. Система управління курсами – студенти, викладачі, курси, оцінки.
3. Сервіс репетиторів – профілі репетиторів, фільтрація, відгуки.
4. Міні-магазин товарів – продукти, категорії, кошик, замовлення.
5. Сервіс оренди техніки – техніка, бронювання, доступність.
6. Порівняння цін – база даних з цінами з різних сайтів/магазинів.
7. Медична картка пацієнта – візити, діагнози, рецепти.
8. Трекер фізичної активності – вправи, тренування, статистика.
9. Сервіс запису до лікаря – розклади, бронювання, нагадування.
10. Гід по місту – туристичні місця, маршрути, рейтинги.
11. Сервіс бронювання турів – дати, країни, опис, знижки.
12. Щоденник мандрівника – фото, описи, геолокація.
13. Онлайн-галерея – автори, картини, стилі.
14. Бібліотека фільмів/книг – жанри, рейтинги, переглянуте.
15. Сервіс афіш подій – концерти, театри, реєстрація.
16. Менеджер завдань – користувачі, задачі, статуси.
17. Система бронювання кімнат – кімнати, дати, доступність.
18. CRM-система – клієнти, комунікації, угоди.
19. Міні-соцмережа – користувачі, дописи, коментарі.
20. Сервіс опитувань – опитування, відповіді, результати.
21. Форум – теми, повідомлення, теги.
22. Система збору аналітики – перегляди, дії користувачів.
23. Сервіс звітів для компаній – графіки, таблиці, фільтрація.
24. Опитування з візуалізацією результатів – збереження відповідей та графіки.
25. Сервіс рекомендацій – користувачі, інтереси, рекомендоване.
26. Портфоліо фахівця – досвід, навички, проєкти.
27. Система резюме – особа, досвід, освіта, навички.
28. База поломок техніки – пристрої, симптоми, рішення.
29. База патентів/винаходів – опис, автори, дата.
30. Система контролю версій документів – версії, зміни, історія.

Додаток 2. Зразок оформлення звіту

ЗВІТ

про виконання лабораторної роботи № 2

“Вибірка даних та вивід. Добавлення, редагування та видалення”

з дисципліни “Вебтехнології ”

студента групи КН-2428

Петренка І.І.

Завдання: Виконати запити до основної таблиці у відповідності до обраної предметної області для вибірки, добавлення, редагування та видалення даних.

Код програми:

```
mysqli_connect('localhost', 'root', '');
mysqli_select_db('stud');
//mysqli_query("SET …");


$res = mysqli_query("SELECT * FROM students");
while ($row = mysqli_fetch_object($res)){
     echo $row->surname." ". $row->name." ". $row-
>rating."<br/>";
}
```

Результат виконання програми:

Іванов Петро 15
Петров Іван 25
Пиріжков Сергій 36
Сидорова Іванка 85
Пончик Аня 63
Пупкін Андрій 91

Висновок: на цій лабораторній роботі я навчився створювати php-сценарій для виводу та модіфікації даних у таблиці БД.

Електронне навчально-методичне видання

Олег Наум, Дмитро Карпин, Анна Карпин

ВЕБТЕХНОЛОГІЇ. РОБОТА З БАЗОЮ ДАНИХ: МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

Дрогобицький державний педагогічний університет імені Івана Франка

Редактор
Ірина Невмержицька
Технічний редактор
Ірина Артимко

Здано до набору 04.07.2027 р. Формат 60x90/16. Гарнітура Times. Ум. друк. арк. 3,50. Зам. 58.

Дрогобицький державний педагогічний університет імені Івана Франка. (Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виготівників та розповсюджувачів видавничої продукції ДК № 5140 від 01.07.2016 р.). 82100, Дрогобич, вул. Івана Франка, 24, к. 203