Дисципліна

«Бекенд-розробка, сервіси та мікросервіси»

КОНСПЕКТ ЛЕКЦІЇ. ТЕМА 7

Робота з базами даних: PDO та рідні методи MySQL, підготовлені запити, транзакції, захист від SQL-ін'єкцій

Спеціальність: Комп'ютерна інженерія, 3 курс
Обсяг: 1 год 20 хв (лекція 7 з 18)

План лекції

1. Підключення до бази даних: PDO та mysqli
2. Виконання запитів і отримання результатів через PDO
3. Підготовлені запити (prepared statements)
4. Захист від SQL-ін'єкцій
5. Транзакції: атомарність операцій
6. Обробка помилок роботи з базою даних
7. CRUD-операції на практиці

> *Мета лекції: навчитися безпечно й надійно працювати з реляційною базою даних із PHP — підключатися, виконувати запити через підготовлені вирази, керувати транзакціями та коректно обробляти помилки.*

1. Підключення до бази даних: PDO та mysqli

PHP пропонує два основні способи роботи з MySQL: розширення mysqli (MySQL Improved) — «рідне» розширення, розроблене спеціально для MySQL, і PDO (PHP Data Objects) — універсальний, незалежний від конкретної СУБД інтерфейс, який працює однаково з MySQL, PostgreSQL, SQLite та іншими базами даних, змінюючи лише рядок підключення.

1.1. Підключення через mysqli

```
<?php
$mysqli = new mysqli('localhost', 'user', 'password', 'shop_db');

if ($mysqli->connect_error) {
    die('Помилка підключення: ' . $mysqli->connect_error);
}

echo 'Підключено успішно';
```

1.2. Підключення через PDO

```
<?php
try {
    $pdo = new PDO(
        'mysql:host=localhost;dbname=shop_db;charset=utf8mb4',
        'user',
        'password'
    );
    $pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
} catch (PDOException $e) {
    die('Помилка підключення: ' . $e->getMessage());
}
```

Критерій	mysqli	PDO
Підтримувані СУБД	Лише MySQL/MariaDB	MySQL, PostgreSQL, SQLite, та інші (12+)
Підготовлені запити	Так (окремий синтаксис)	Так (уніфікований синтаксис)
Об'єктний і процедурний стиль	Обидва підтримуються	Лише об'єктно-орієнтований
Іменовані параметри	Не підтримуються	Підтримуються (:name)

> *У цьому курсі надалі використовуємо PDO як основний інструмент: він гнучкіший (не прив'язує код до конкретної СУБД), має уніфікований і зручніший синтаксис підготовлених запитів та є практичним стандартом у сучасних PHP-фреймворках (Laravel, Symfony).*

2. Виконання запитів і отримання результатів через PDO

2.1. query() для запитів без параметрів

Метод query() виконує SQL-запит одразу й повертає об'єкт PDOStatement, з якого можна вибрати результат. Цей спосіб підходить лише для запитів без зовнішніх (введених користувачем) даних — тобто фіксованих запитів.

```
<?php
$stmt = $pdo->query('SELECT id, name, price FROM products');

// Отримати всі рядки одразу як масив асоціативних масивів
$products = $stmt->fetchAll(PDO::FETCH_ASSOC);

foreach ($products as $product) {
    echo $product['name'] . ' — ' . $product['price'] . "\n";
}
```

2.2. Режими вибірки (fetch modes)

Режим	Результат
PDO::FETCH_ASSOC	Асоціативний масив з іменами колонок як ключами
PDO::FETCH_NUM	Масив з числовими індексами (0, 1, 2...)
PDO::FETCH_OBJ	Об'єкт stdClass із властивостями-колонками
PDO::FETCH_CLASS	Екземпляр заданого власного класу

2.3. fetch() для покрокової обробки

Коли результат запиту може містити багато рядків, fetchAll() завантажує їх усі в пам'ять одразу, що не завжди ефективно. Метод fetch() повертає по одному рядку за викликом, дозволяючи обробляти результат у циклі.

```
<?php
$stmt = $pdo->query('SELECT id, name FROM products');

while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo $row['id'] . ': ' . $row['name'] . "\n";
}
```

2.4. exec() для запитів без результату

Для запитів, які не повертають набір рядків (INSERT, UPDATE, DELETE без параметрів), зручніше використовувати exec() — він повертає кількість змінених рядків замість об'єкта PDOStatement.

```php
<?php
$affected = $pdo->exec("UPDATE products SET price = price * 1.1");
echo "Оновлено рядків: $affected";
```

3. Підготовлені запити (prepared statements)

Підготовлений запит — це SQL-вираз із заповнювачами (placeholders) замість конкретних значень, який спочатку компілюється базою даних окремо від даних, а потім виконується з підставленими значеннями. PDO підтримує два синтаксиси заповнювачів: позиційні знаки питання (?) та іменовані параметри (:name).

3.1. Позиційні параметри

```
<?php
$stmt = $pdo->prepare('SELECT * FROM products WHERE category = ? AND price
< ?');
$stmt->execute(['electronics', 20000]);

$products = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

3.2. Іменовані параметри

Іменовані параметри роблять запит читабельнішим, особливо коли параметрів багато — не потрібно рахувати позицію кожного знака питання.

```
<?php
$stmt = $pdo->prepare(
    'SELECT * FROM products WHERE category = :category AND price
< :maxPrice'
);
$stmt->execute([
    ':category' => 'electronics',
    ':maxPrice' => 20000,
]);

$products = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

3.3. INSERT, UPDATE, DELETE з підготовленими запитами

```
<?php
// INSERT
$stmt = $pdo->prepare('INSERT INTO products (name, price) VALUES
(:name, :price)');
$stmt->execute(['name' => 'Клавіатура', 'price' => 850]);
$newId = $pdo->lastInsertId();   // ID щойно вставленого рядка

// UPDATE
$stmt = $pdo->prepare('UPDATE products SET price = :price WHERE id
= :id');
$stmt->execute(['price' => 900, 'id' => $newId]);

// DELETE
$stmt = $pdo->prepare('DELETE FROM products WHERE id = :id');
$stmt->execute(['id' => $newId]);
echo $stmt->rowCount();   // кількість видалених рядків
```

> *bindParam() і bindValue() — альтернативні способи прив'язки значень до підготовленого запиту (замість передачі масиву в execute()), які дозволяють додатково вказати тип даних (PDO::PARAM_INT, PDO::PARAM_STR) явно. На практиці масив в execute() покриває більшість типових сценаріїв і використовується частіше через лаконічність.*

4. Захист від SQL-ін'єкцій

SQL-ін'єкція — одна з найнебезпечніших і водночас найпоширеніших вразливостей вебзастосунків: вона виникає, коли дані від користувача підставляються безпосередньо в текст SQL-запиту без належної обробки, дозволяючи зловмиснику змінити структуру запиту й виконати довільні SQL-команди.

```
<?php
// НЕБЕЗПЕЧНО: НІКОЛИ так не робіть
$login = $_POST['login'];   // зловмисник передає: admin' --
$sql = "SELECT * FROM users WHERE login = '$login'";
$result = $pdo->query($sql);

// Отриманий запит:
// SELECT * FROM users WHERE login = 'admin' --'
// Символи -- перетворюють залишок рядка на коментар,
// перевірка пароля повністю зникає із запиту!
```

Підготовлені запити повністю усувають цю вразливість: значення параметрів передаються окремо від тексту запиту й ніколи не інтерпретуються як частина SQL-синтаксису, незалежно від їхнього вмісту.

```
<?php
// БЕЗПЕЧНО: параметр завжди трактується як значення, а не SQL-код
$login = $_POST['login'];
$stmt = $pdo->prepare('SELECT * FROM users WHERE login = :login');
$stmt->execute(['login' => $login]);

// Навіть якщо $login містить "admin' --", це просто рядок для порівняння,
// а не частина SQL-запиту
```

> *Правило без винятків: жодні дані, що надходять від користувача (з форми, URL, cookie, JSON-тіла), не повинні потрапляти в SQL-запит через конкатенацію рядків (.) чи інтерполяцію в лапках. Єдиний надійний спосіб — підготовлені запити з параметрами. Це стосується навіть, здавалося б, «безпечних» числових ідентифікаторів.*

5. Транзакції: атомарність операцій

Транзакція — це послідовність операцій з базою даних, які повинні виконатись усі разом, як єдине ціле: якщо хоча б одна з них зазнає невдачі, усі попередні зміни в межах транзакції скасовуються (відкат), і база даних повертається у стан до початку транзакції. Це критично важливо для операцій, що зачіпають кілька пов'язаних таблиць.

Класичний приклад — переказ коштів між банківськими рахунками: списання з одного рахунку і зарахування на інший повинні відбутися разом. Якщо після списання станеться збій перед зарахуванням, гроші «зникнуть» — транзакція запобігає такому сценарію.

```
<?php
try {
    $pdo->beginTransaction();

    $stmt1 = $pdo->prepare('UPDATE accounts SET balance = balance
- :amount WHERE id = :from');
    $stmt1->execute(['amount' => 500, 'from' => 1]);

    $stmt2 = $pdo->prepare('UPDATE accounts SET balance = balance
+ :amount WHERE id = :to');
    $stmt2->execute(['amount' => 500, 'to' => 2]);

    $pdo->commit();   // застосувати обидві зміни остаточно
    echo 'Переказ виконано успішно';
} catch (PDOException $e) {
    $pdo->rollBack();   // скасувати всі зміни транзакції
    echo 'Помилка переказу: ' . $e->getMessage();
}
```

Метод	Призначення
beginTransaction()	Починає транзакцію
commit()	Застосовує (фіксує) усі зміни транзакції остаточно
rollBack()	Скасовує всі зміни, зроблені з моменту beginTransaction()

> *Транзакції підтримуються лише для таблиць зі сховищем InnoDB (стандартне сховище MySQL за замовчуванням); старіше сховище MyISAM транзакції ігнорує. Це варто перевірити, якщо транзакції в коді «не працюють» без видимої причини.*

6. Обробка помилок роботи з базою даних

У режимі PDO::ERRMODE_EXCEPTION (рекомендований і встановлюваний одразу після підключення) будь-яка помилка бази даних — некоректний SQL-синтаксис, порушення обмеження унікальності, втрата з'єднання — призводить до винятку PDOException, який слід перехоплювати конструкцією try/catch.

```
<?php
try {
    $stmt = $pdo->prepare('INSERT INTO users (email) VALUES (:email)');
    $stmt->execute(['email' => 'existing@example.com']);
} catch (PDOException $e) {
    if ($e->getCode() === '23000') {
        // Порушення обмеження унікальності (напр. email уже існує)
        echo 'Користувач із такою поштою вже зареєстрований';
    } else {
        // Логуємо технічну помилку, але НЕ показуємо деталі користувачу
        error_log($e->getMessage());
        echo 'Сталася помилка. Спробуйте пізніше.';
    }
}
```

> *Показувати повний текст помилки PDOException (getMessage()) безпосередньо користувачу на реальному сайті небезпечно — повідомлення може розкрити структуру бази даних (назви таблиць, колонок) зловмиснику. Деталі помилки варто записувати в лог-файл (error_log() чи спеціалізовану систему логування), а користувачу показувати узагальнене повідомлення.*

7. CRUD-операції на практиці

CRUD (Create, Read, Update, Delete) — чотири базові операції, з яких складається робота практично будь-якого застосунку з даними. Нижче — узагальнений приклад невеликого класу-репозиторію, що інкапсулює всі CRUD-операції для однієї таблиці, поєднуючи все розглянуте в цій лекції.

```
<?php
class ProductRepository {
    public function __construct(private PDO $pdo) {}

    public function findAll(): array {
        return $this->pdo->query('SELECT * FROM products')-
>fetchAll(PDO::FETCH_ASSOC);
    }

    public function findById(int $id): ?array {
        $stmt = $this->pdo->prepare('SELECT * FROM products WHERE id
= :id');
        $stmt->execute(['id' => $id]);
        $result = $stmt->fetch(PDO::FETCH_ASSOC);
        return $result ?: null;
    }

    public function create(string $name, float $price): int {
        $stmt = $this->pdo->prepare(
            'INSERT INTO products (name, price) VALUES (:name, :price)'
        );
        $stmt->execute(['name' => $name, 'price' => $price]);
        return (int) $this->pdo->lastInsertId();
    }

    public function update(int $id, float $price): bool {
        $stmt = $this->pdo->prepare('UPDATE products SET price = :price
WHERE id = :id');
        return $stmt->execute(['price' => $price, 'id' => $id]);
    }

    public function delete(int $id): bool {
        $stmt = $this->pdo->prepare('DELETE FROM products WHERE id
= :id');
        return $stmt->execute(['id' => $id]);
    }
}

// Використання
$repo = new ProductRepository($pdo);
$newId = $repo->create('Монітор', 7500);
$product = $repo->findById($newId);
```

> *Такий клас-репозиторій — це вже елемент архітектурного патерну Repository, який ізолює SQL-запити від решти коду застосунку: якщо завтра потрібно буде змінити структуру таблиці чи навіть перейти на іншу СУБД, зміни торкнуться лише цього одного класу, а не всього коду, що працює з товарами.*

Питання для самоперевірки

- Чим PDO принципово відрізняється від mysqli з погляду підтримки різних СУБД?
- У чому різниця між методами query() і prepare()+execute()? Коли обов'язково потрібен другий варіант?
- Чим позиційні параметри (?) відрізняються від іменованих (:name) у підготовлених запитах?
- Поясніть на прикладі, як саме підготовлені запити захищають від SQL-ін'єкції.
- Що станеться з базою даних, якщо після beginTransaction() виконати кілька запитів, а потім викликати rollBack()?
- Чому режим PDO::ERRMODE_EXCEPTION вважається рекомендованим порівняно з режимом за замовчуванням?
- Чому не варто показувати користувачу повний текст повідомлення PDOException?
- Яку архітектурну перевагу дає винесення SQL-запитів в окремий клас-репозиторій?

Короткий глосарій

Термін	Означення
PDO (PHP Data Objects)	Універсальний, незалежний від СУБД інтерфейс доступу до баз даних у PHP
Підготовлений запит (prepared statement)	SQL-запит із заповнювачами, що компілюється окремо від переданих значень
SQL-ін'єкція	Вразливість, за якої дані користувача змінюють структуру SQL-запиту
Транзакція	Група операцій з базою даних, що виконуються як єдине ціле (усі або жодна)
Відкат (rollback)	Скасування всіх змін, зроблених у межах поточної транзакції
CRUD	Create, Read, Update, Delete — чотири базові операції з даними
Repository (патерн)	Клас, що інкапсулює логіку доступу до даних однієї сутності