

ТЕМА 7

Робота з базами даних: PDO

Підготовлені запити · захист від SQL-ін'єкцій · транзакції · обробка помилок · CRUD

Бекенд-розробка, сервіси та мікросервіси

Комп'ютерна інженерія · 3 курс · Лекція 7 з 18

7 питань сьогоднішньої теми

01

Підключення: PDO та
mysqlі

02

Виконання запитів
через PDO

03

Підготовлені запити

04

Захист від SQL-ін'єкцій

05

Транзакції

06

Обробка помилок
роботи з БД

07

CRUD-операції на
практиці

PDO vs mysqli

Критерій	mysqli	PDO
Підтримувані СУБД	Лише MySQL/MariaDB	MySQL, PostgreSQL, SQLite та ін.
Іменовані параметри	Не підтримуються	Підтримуються (:name)
Стиль	Процедурний і ООП	Лише ООП

У цьому курсі: використовуємо PDO — гнучкіший, уніфікований синтаксис, практичний стандарт у Laravel і Symfony.

Підключення через PDO

```
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());  
}
```

query() та режими вибірки

```
$stmt = $pdo->query('SELECT id, name, price FROM products');  
$products = $stmt->fetchAll(PDO::FETCH_ASSOC);
```

Режим	Результат
PDO::FETCH_ASSOC	Асоціативний масив (ключі — імена колонок)
PDO::FETCH_NUM	Масив з числовими індексами
PDO::FETCH_OBJ	Об'єкт stdClass

fetch() у циклі та exec()

fetch() — по одному рядку

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

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

exec() — без набору рядків

```
$affected = $pdo->exec(
    "UPDATE products
    SET price = price * 1.1"
);

echo "Оновлено: $affected";
```

Позиційні (?) vs іменовані (:name)

Позиційні параметри

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

Іменовані параметри

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

INSERT, UPDATE, DELETE

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

// 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();
```

Небезпечно: конкатенація рядків

```
$login = $_POST['login']; // зловмисник: admin' --  
$sql = "SELECT * FROM users WHERE login = '$login'";  
$result = $pdo->query($sql);  
  
// Отриманий запит:  
// SELECT * FROM users WHERE login = 'admin' --'  
// -- перетворює залишок рядка на коментар,  
// перевірка пароля зникає із запиту!
```

Дані користувача НІКОЛИ не мають потрапляти в SQL через конкатенацію.

Безпечно: підготовлений запит

```
$login = $_POST['login'];  
$stmt = $pdo->prepare('SELECT * FROM users WHERE login = :login');  
$stmt->execute(['login' => $login]);  
  
// Навіть якщо $login = "admin' --", це просто рядок  
// для порівняння, а не частина SQL-запиту
```

Правило без винятків: жодні зовнішні дані не потрапляють у SQL через конкатенацію — лише через параметри підготовленого запиту.

beginTransaction, commit, rollBack

```
try {
    $pdo->beginTransaction();

    $s1 = $pdo->prepare('UPDATE accounts SET balance = balance - :a WHERE id = :from');
    $s1->execute(['a' => 500, 'from' => 1]);

    $s2 = $pdo->prepare('UPDATE accounts SET balance = balance + :a WHERE id = :to');
    $s2->execute(['a' => 500, 'to' => 2]);

    $pdo->commit();
} catch (PDOException $e) {
    $pdo->rollBack();
    echo 'Помилка переказу: ' . $e->getMessage();
}
```

Методи керування транзакцією

Метод	Призначення
<code>beginTransaction()</code>	Починає транзакцію
<code>commit()</code>	Застосовує (фіксує) усі зміни остаточно
<code>rollback()</code>	Скасовує всі зміни з моменту початку транзакції

Транзакції підтримуються лише для таблиць InnoDB — старіше сховище MyISAM їх ігнорує.

PDOException та коди помилок

```
try {  
    $stmt = $pdo->prepare('INSERT INTO users (email) VALUES (:email)');  
    $stmt->execute(['email' => 'existing@example.com']);  
} catch (PDOException $e) {  
    if ($e->getCode() === '23000') {  
        // Порушення унікальності (email уже існує)  
        echo 'Такий email уже зареєстровано';  
    } else {  
        error_log($e->getMessage()); // деталі — лише в лог  
        echo 'Сталася помилка. Спробуйте пізніше.';  
    }  
}
```

Клас-репозиторій: інкапсуляція SQL

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

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

Підсумок теми 7

- PDO — уніфікований, незалежний від СУБД спосіб роботи з базами даних
- Підготовлені запити (prepare + execute) — стандарт для будь-яких запитів із зовнішніми даними
- Підготовлені запити повністю усувають SQL-ін'єкції
- Транзакції (beginTransaction/commit/rollBack) гарантують атомарність пов'язаних операцій
- PDO::ERRMODE_EXCEPTION + try/catch — коректна обробка помилок без витоку деталей користувачу
- Клас-репозиторій ізолює SQL-запити від решти коду застосунку