

# Doctrine ORM – relacje, N+1 i optymalizacja zapytań

 Piotr Wiśniewski  13.10.2026  314 wyświetleń  3 komentarzy

 Dostępne w językach:

 PL

 EN

[🏠 Start](#) / [Blog](#) / [Bazy Danych](#)

/ Doctrine ORM – relacje, N+1 i optymalizacja zapytań

[← Wróć do bloga](#)



Pobierz PDF

## Rodzaje relacji w Doctrine

```
//ManyToOne – wiele artykułów należy do jednego u  
#[ORM\ManyToOne(targetEntity: User::class)]  
#[ORM\JoinColumn(nullable: false)]  
private ?User $author = null;
```

```
//OneToMany – jeden artykuł ma wiele komentarzy  
#[ORM\OneToMany(targetEntity: Comment::class, mapped  
private Collection $comments;
```

```
//ManyToMany – artykuły i tagi  
#[ORM\ManyToMany(targetEntity: Tag::class, inverseSe
```

```
#[ORM\JoinTable(name: 'article_tags')]  
private Collection $tags;
```

---

## Problem N+1

---

To najczęstszy błąd wydajnościowy w aplikacjach z ORM:

```
// ŹLE – 1 + N zapytań (1 dla listy + po jednym dla  
$articles = $repo->findAll();  
foreach ($articles as $article) {  
    echo $article->getAuthor()->getName(); // ← osł  
}
```

Przy 100 artykułach = **101 zapytań do bazy.**

---

## Rozwiązanie: JOIN FETCH

---

```
// DOBRZE – 1 zapytanie z JOIN  
public function findWithAuthors(): array  
{  
    return $this->createQueryBuilder('a')  
        ->leftJoin('a.author', 'u')  
        ->addSelect('u') //  
        ->leftJoin('a.tags', 't')  
        ->addSelect('t')  
        ->where('a.status = :status')  
        ->setParameter('status', 'published')  
        ->orderBy('a.publishedAt', 'DESC')  
        ->getQuery()  
        ->getResult();  
}
```

## Paginacja z Pagerfanta

---

Nie ładuj wszystkiego — ogranicz wyniki:

```
use Pagerfanta\Doctrine\ORM\QueryAdapter;
use Pagerfanta\Pagerfanta;

public function findPaginated(int $page, int $perPage)
{
    $qb = $this->createQueryBuilder('a')
        ->where('a.status = :status')
        ->setParameter('status', 'published')
        ->orderBy('a.publishedAt', 'DESC');

    $pagerfanta = new Pagerfanta(new QueryAdapter($qb));
    $pagerfanta->setMaxPerPage($perPage);
    $pagerfanta->setCurrentPage($page);

    return $pagerfanta;
}
```

---

## Batch processing — import dużych zbiorów

---

```
public function importArticles(array $data): void
{
    foreach ($data as $i => $row) {
        $article = new BlogArticle();
        $article->setTitle($row['title']);
        // ...
        $this->em->persist($article);

        // co 100 rekordów — flush i czyszczenie Uo
        if ($i % 100 === 0) {
            $this->em->flush();
            $this->em->clear(); // ← bez tego pam:
        }
    }
}
```

```
$this->em->flush(); // reszta  
}
```

## Diagnostyka – SQL Logger

W środowisku deweloperskim włącz logowanie zapytań do profiler Symfony. Szukaj:

- zapytań > 100ms
- powtarzających się zapytań (znak N+1)
- `SELECT *` zamiast konkretnych kolumn

```
# ile zapytań wykonało się dla danego request?  
# Otwórz Symfony Profiler → zakładka "Doctrine"
```

## Ściągawka – fetch mode

| Tryb                            | Kiedy                    | Zachowanie                         |
|---------------------------------|--------------------------|------------------------------------|
| <code>LAZY</code><br>(domyślny) | relacje rzadko potrzebne | ładuje przy dostępie               |
| <code>EAGER</code>              | relacje zawsze potrzebne | ładuje razem z encją               |
| <code>EXTRA_LAZY</code>         | duże kolekcje            | <code>count()</code> bez ładowania |
| <code>JOIN FETCH</code>         | ręczne w DQL             | najlepsza kontrola                 |

🔖 Tagi:

#PHP

#Symfony

#PostgreSQL

Udostępnij:

 Facebook

 X

 LinkedIn

 Kopiuj link



## Komentarze (3)

J

**Jan Kowalski** 21.10.2026, 14:27

N+1 mnie kiedyś kosztował poważny incydent na produkcji. Teraz zawsze addSelect.

A

**Anna Nowak** 21.10.2026, 14:27

Warto też włączyć SQL logger na dewie żeby zobaczyć co leci do bazy.

P

**Piotr Wiśniewski** 21.10.2026, 14:27

Batch processing z clear() to must-have przy importach. Bez tego OOM.

[Zaloguj się](#), aby dodać komentarz.



## FactoryCode

Projekt demonstracyjny zbudowany na Symfony 8, PHP 8.4 i PostgreSQL 16.  
Tłumaczenia zarządzane przez panel admina z AI.

### NAWIGACJA



Start



Blog



Projekty



Kontakt



Polityka prywatności

## KONTAKT

✉ [kontakt@factorycode.pl](mailto:kontakt@factorycode.pl)

🐙 [GitHub](#)

---

© 2026 FactoryCode. Wszelkie prawa  
zastrzeżone.

Zbudowane z [Symfony 8](#) & [Claude](#)  
[AI](#)