Eager Loading vs Lazy Loading in Laravel: Best Practices

Eager Loading vs Lazy Loading in Laravel: Best Practices

When working with Eloquent relationships, performance issues often appear because of the “N+1” query problem. This happens when each record triggers additional queries for related data. Laravel provides **lazy loading** and **eager loading** to control how related data is fetched. In this article, you’ll learn the difference between them, see code examples, and discover best practices for balancing performance and memory usage. We’ll also build a simple profile UI that uses eager loading to display related posts efficiently.

1 – Understanding Lazy Loading

Lazy loading means related data is only loaded when you first access it. This keeps initial queries lightweight but can trigger many additional queries if you loop through related data.

```
// app/Http/Controllers/UserController.php
namespace App\Http\Controllers;

use App\Models\User;

class UserController extends Controller
{
    public function index()
    {
        $users = User::all(); // single query for users

        foreach ($users as $user) {
```

```php
            // Each call triggers a separate query (N+1 problem)
            $posts = $user->posts;
        }

        return view('users.index', compact('users'));
    }
}
```
Code language: PHP (php)

Here, `User::all()` fetches all users. But every time `$user->posts` is accessed in the loop, Laravel issues another query, leading to dozens or hundreds of queries—this is the *N+1 problem*.

2 – Using Eager Loading

Eager loading solves the N+1 problem by fetching related data in advance with one extra query. You use `with()` to tell Eloquent which relations to load.

```php
// app/Http/Controllers/UserController.php (improved)
namespace App\Http\Controllers;

use App\Models\User;

class UserController extends Controller
{
    public function index()
    {
        // One query for users + one query for posts
        $users = User::with('posts')->get();

        return view('users.index', compact('users'));
    }
}
```
Code language: PHP (php)

Instead of dozens of queries, Laravel now runs two: one for users and one for posts. Relations are hydrated into each user automatically, eliminating N+1 overhead.

3 – Nested Eager Loading

You can eager load nested relations by passing arrays to `with()`. Useful for dashboards that display multi-level data.

```
// Fetch users with posts and each post's comments
$users = User::with(['posts.comments'])->get();Code language: PHP (php)
```

This runs three queries: one for users, one for posts, and one for comments. Eloquent maps everything together in memory, saving you from manually stitching queries.

4 – Eager Loading with Constraints

Use a closure inside `with()` to filter related data. This prevents loading unnecessary rows.

```
// Only load published posts for each user
$users = User::with(['posts' => function ($q) {
    $q->where('status', 'published');
}])->get();Code language: PHP (php)
```

Even though a user might have many posts in different statuses, only published ones are pulled into memory. This keeps responses small and efficient.

5 – Lazy Eager Loading (N+1 Fix After the Fact)

If you've already loaded a collection without eager loading, you can still fix N+1 by calling `load()` after the fact.

```
$users = User::all();         // only users
$users->load('posts');        // now posts are fetched in one extra
query

foreach ($users as $user) {
    // no extra queries here
    foreach ($user->posts as $post) {
        echo $post->title;
    }
}
```
Code language: PHP (php)

`load()` attaches the related models after the initial query, solving the N+1 problem if you forgot to use `with()` upfront.

6 – A Simple Profile UI Example

Let's build a profile page that shows the user's information and their posts. We'll use eager loading to prevent N+1 queries.

```
// app/Http/Controllers/ProfileController.php
namespace App\Http\Controllers;
```

```
use App\Models\User;

class ProfileController extends Controller
{
    public function show($id)
    {
        $user = User::with('posts')->findOrFail($id);
        return view('profile.show', compact('user'));
    }
}
```
Code language: PHP (php)

This controller uses `with('posts')` so both the user and their posts are loaded in only two queries.

```
<!-- resources/views/profile/show.blade.php -->
@extends('layouts.app')

@section('content')
<div class="container">
  <h1>{{ $user->name }}’s Profile</h1>
  <p class="text-muted">Email: {{ $user->email }}</p>

  <h3 class="mt-5 mb-3">Posts</h3>
  @foreach($user->posts as $post)
    <div class="card mb-3">
      <div class="card-body">
        <h5 class="card-title">{{ $post->title }}</h5>
        <p class="card-text">{{ Str::limit($post->body,120) }}</p>
      </div>
    </div>
  @endforeach
</div>
@endsection
```
Code language: PHP (php)

The Blade file prints user info and loops through their posts. Because we eager-loaded the posts in the controller, this page will always use a constant two queries, no matter how many posts exist.

Wrapping Up

Lazy loading is simple but can cause N+1 queries in loops. Eager loading (`with()`) fetches related data in bulk and avoids performance pitfalls. Lazy eager loading (`load()`) is handy when you already have a collection but need extra relations. By choosing the right loading method, you keep queries efficient and your app fast—even with complex relationships.

What's Next

- Advanced Eloquent Relationships: Tips and Tricks
- How to Use Eloquent API Resources for Clean APIs
- Query Performance Tuning in Laravel + MySQL