1v0 Ship v1.0 Faster

How to Speed Up Laravel with Database Indexing

How to Speed Up Laravel with Database Indexing

As your Laravel app scales, database queries often become the bottleneck. A single missing index can slow down a query from milliseconds to seconds. Indexing helps the database find rows faster by creating lookup structures on columns. In this guide, we'll cover different types of indexes, how to create them with migrations, and how to verify performance improvements.

1 – Why Indexing Matters

Without indexes, databases must scan entire tables to find records. With indexes, searches are optimized like looking up words in a dictionary. The bigger your tables, the more critical indexes become. For high-traffic Laravel apps, indexing is as important as caching and queues (see 10 Proven Ways to Optimize Laravel for High Traffic).

2 – Adding Indexes in Migrations

Laravel migrations make it easy to add indexes when creating or modifying tables.

```
// database/migrations/2025_08_27_000001_add_indexes_to_users.php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
return new class extends Migration {
    public function up(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->index('email');
            $table->index(['first_name', 'last_name']); // composite
index
        });
    }

    public function down(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropIndex(['email']);
            $table->dropIndex(['first_name', 'last_name']);
        });
    }
};Code language: PHP (php)
```

This migration adds a single-column index on `email` and a composite index on `first_name` + `last_name`. The database can now quickly locate users by email or full name.

3 – Unique Indexes

Indexes can also enforce uniqueness, preventing duplicate values.

```
// database/migrations/2025_08_27_000002_add_unique_index.php
Schema::table('users', function (Blueprint $table) {
    $table->unique('username');
});Code language: PHP (php)
```

This ensures no two users can have the same `username`. The index speeds up lookups and enforces integrity at the same time.

4 – Full-Text Indexes

For searching text columns, use full-text indexes. Supported in MySQL and PostgreSQL, they allow fast search with `MATCH ... AGAINST`.

```php
// database/migrations/2025_08_27_000003_add_fulltext_index.php
Schema::table('posts', function (Blueprint $table) {
    $table->fullText('content');
});
```
Code language: PHP (php)

This lets you query posts with natural language search, far faster than scanning every row. For complex searches, combine full-text with Laravel's query builder filtering techniques.

5 – Measuring Query Performance

Always measure performance before and after adding indexes using `EXPLAIN` in MySQL/Postgres.

```sql
EXPLAIN SELECT * FROM users WHERE email = 'test@example.com';
```
Code language: SQL (Structured Query Language) (sql)

Before adding the index, this query requires a full table scan. After the index, the DB uses an indexed lookup, which is dramatically faster on large datasets.

6 – Indexing Best Practices

- Index columns that are frequently used in `WHERE`, `JOIN`, and `ORDER BY`.
- Avoid indexing every column—indexes add write overhead.
- Use composite indexes for multi-column searches.
- Regularly analyze queries with `EXPLAIN`.

Indexing complements caching and eager loading. For more on optimizing queries, see Eager Loading vs Lazy Loading in Laravel: Best Practices.

Wrapping Up

Database indexing is one of the most effective optimizations for Laravel apps. By adding single, composite, unique, and full-text indexes, you can cut query times dramatically. Always measure queries with `EXPLAIN` and avoid over-indexing. Combined with caching and queues, indexes help Laravel handle millions of queries efficiently.

What’s Next

- 10 Proven Ways to Optimize Laravel for High Traffic — indexing is a key part of large-scale optimization.

- Caching Strategies in Laravel: Redis vs Database vs File — combine indexes with caching for maximum performance.
- Eager Loading vs Lazy Loading in Laravel: Best Practices — avoid unnecessary queries on top of optimized indexes.