Миграции LARAVEL 12

Гайд по миграциям в **Laravel 12**

> Laravel 12 вышел 6 февраля 2025 г. и привнёс несколько аккуратных улучшений к системе миграций — от обработчика `shouldRun()` до опции `--isolated` для безопасного выката. Ниже — выжимка шаг-за-шагом, с примерами кода и советами из практики.

---

1. Концепция и подготовка окружения

Миграции — это контроль версий для вашей БД (laravel.com).

1. **Требования**: PHP ≥ 8.3, Composer ≥ 2.8, любая из поддерживаемых СУБД (MySQL 8, MariaDB 10.6, PostgreSQL 13, SQLite 3, SQL Server 2019).
2. **Настройка** `.env`: заполняем `DB_CONNECTION`, `DB_HOST`, `DB_PORT`, `DB_DATABASE`, `DB_USERNAME`, `DB_PASSWORD`.
3. **Запуск проекта**:

```
composer create-project laravel/laravel:^12 myapp
cd myapp && php artisan serve
```

---

2. Создаём миграции

Задача	Команда	Что делает
Новая таблица	`php artisan make:migration create_posts_table`	Laravel сам подставит `Schema::create()` в файле (laravel.com)
Изменить таблицу	`php artisan make:migration add_status_to_posts_table --table=posts`	Сгенерирует `Schema::table()`

Задача	Команда	Что делает
Указать путь	`--path=database/migrations/admin`	Полезно при модулярной архитектуре
Создать, но не запускать пока	добавить в файл метод `shouldRun()`	Вернёт `false`, миграция пропустится (laravel.com)

Минимальный пример

```
<?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::create('posts', function (Blueprint $table) {
            $table->id();
            $table->string('title')->index();
            $table->text('body');
            $table->boolean('published')->default(false);
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('posts');
    }
};
```

---

3. Запуск миграций

Команда	Для чего нужна	Где пригодится
`php artisan migrate`	Применить все «висящие» миграции (laravel.com)	Локальная разработка

Команда	Для чего нужна	Где пригодится
`php artisan migrate --pretend`	Показать SQL без выполнения (laravel.com)	Code-review, CI
`php artisan migrate --isolated`	Заблокирует одновременный запуск на нескольких нодах (laravel.com)	Blue-/green deploy
`php artisan migrate --force`	Подавить подтверждение в prod (laravel.com)	CI/CD pipeline
`php artisan migrate:status`	Таблица применённых/неприменённых (laravel.com)	Аудит

---

4. Откат и перезапуск

Команда	Что делает
`php artisan migrate:rollback`	Откатить последний *batch* (laravel.com)
`php artisan migrate:rollback --step=2`	Откатить ровно 2 файла
`php artisan migrate:reset`	Откатить **всё** (laravel.com)
`php artisan migrate:refresh --seed`	Полный откат → заново → сиды (laravel.com)
`php artisan migrate:fresh`	Drop all tables → заново (laravel.com)

---

5. Оптимизация: squash-дампы

С ростом проекта сотни файлов замедляют CI.
Используйте:

```
php artisan schema:dump --prune      # создаст dump.sql и удалит старые миграции
```

На чистой базе Laravel сначала прогонит `dump.sql`, а затем только новые миграции (laravel.com).

---

6. Лайфхаки и лучшие практики

1. **Атомарность**: тяжёлые изменения (переименование столбцов, удаление индексов) выполняйте в отдельных мелких миграциях.
2. **Idempotency**: проверяйте `Schema::hasTable()` / `hasColumn()` перед изменениями для совместимости между ветками (laravel.com).
3. **Транзакции**: `public bool $withinTransaction = true;` в классе миграции гарантирует откат при ошибке (по умолчанию true для поддерживаемых СУБД).
4. **Страницируйте деплой**: для «живых» таблиц используйте «параллельную» стратегию — добавьте новый столбец, заполните его бэкграунд-джобой, переключайтесь, потом удаляйте старый.
5. **CI-тесты**: в PHPUnit подключите трейт `RefreshDatabase` — он вызывает `migrate:fresh` и откатывает после тестов.
6. **Stub publishing**:

   ```
   php artisan stub:publish --only=migration
   resources/stubs/migration.stub
   ```

   Настройте свои шаблоны под корпоративные стандарты.

---

7. Отладка типичных ошибок

Симптом	Причина	Решение
`SQLSTATE[42S01]: Table already exists`	Миграция не в `down()` или путаются имена	Проверить `down()`, убедиться в правильном порядке launch-batch
`Class ... not found`	Переименовали файл/класс	Очистить кэш `composer dump-autoload`
Ошибка кодировки/сортировки	MySQL 8, несовпадение `charset/collation`	Явно задать `$table->charset('utf8mb4');` (laravel.com)

---

8. Чек-лист перед продакшен-выкатом

- [ ] Миграции проходят `php artisan migrate --pretend` на staging.
- [ ] Включён `--isolated` или ручная блокировка.
- [ ] Тесты используют ту же версию СУБД, что и prod.
- [ ] Резервная копия БД лежит в безопасном месте.

☐
☐ Для разрушительных изменений используется «expand-migrate-contract» паттерн.

---

9. Ссылки для углубления

- Официальная документация «Database → Migrations» Laravel 12 (laravel.com)
- Ченджлог 12.x (`shouldRun`, `isolated`) GitHub (laravel.com)
- Release Notes 12.x для общего понимания цикла релизов (laravel.com)

Краткий итог

Миграции в Laravel 12 — зрелый инструмент: **генерируем** `make:migration`, **описываем** `up()/down()`, **запускаем** `migrate`, **откатываем** `rollback`. Новинка `shouldRun()` помогает «прятать» эксперименты, а `--isolated` — уберечь прод от гонок. Пользуйтесь «мелкими, атомарными» паттернами, держите *schema dump* в репозитории и внедряйте миграции в CI — тогда база данных будет версионироваться так же уверенно, как и код.

Revision #1
Created 22 June 2025 03:07:14 by Admin
Updated 22 June 2025 03:08:05 by Admin