Laravel Models + Migrations — Database Se Rabta

The database connection: .env settings, a migration as the table's blueprint, php artisan migrate, and the Model that speaks for the table.

Introduction

Real apps remember. Students, marks, orders — the data must survive after the browser closes, and that home is the **database**. Laravel connects to it with two partners you will use in every project: **migrations**, which are versioned blueprints that build your tables, and **models**, PHP classes that speak for those tables so your code says `Student::all()` instead of writing raw SQL.

Why blueprints instead of clicking around in a database tool? Because a migration is code: it travels with the project, every teammate (and the live server, LV.8) runs the same file and gets the identical table. In this lesson you will connect the app to MySQL, design the students table as a migration, bring it to life, and read and write rows through the Student model — **Eloquent**, Laravel's object way of talking to tables.

(Roman Urdu: Migration table ka naksha hai jo code mein rehta hai; Model us table ka numainda hai jo aap ki zubaan bolta hai.)

Key points

- **Connection settings live in `.env`:** `DB_CONNECTION=mysql`, `DB_DATABASE=school_app`, `DB_USERNAME`, `DB_PASSWORD`. The app reads these at boot; change the file, change the database — no code edits. Create the empty database first (in phpMyAdmin or MySQL), Laravel fills it.
- **A migration is a table's biography in code:** `php artisan make:migration create_students_table` creates a file with two methods — `up()` builds, `down()` undoes. Laravel runs each migration once and remembers which ones ran.
- **`Schema::create` designs the table:** `$table->id()`, `$table->string('name')`, `$table->integer('marks')`, `$table->timestamps()` (created_at + updated_at, managed automatically).
- **`php artisan migrate` is the construction day:** it runs every pending `up()` and reports each table built. Changed your mind? `migrate:rollback` runs `down()` and the table is gone — cleanly.
- **A model represents one table:** `php artisan make:model Student` creates `app/Models/Student.php`. By convention the class `Student` speaks for the table `students` — plural, lowercase. No configuration needed when you follow the naming.

  (Roman Urdu: Model ka naam akela (Student), table ka naam jama (students) — ye chhota sa usool Laravel khud jor deta hai.)
- **Eloquent reads like English:** `Student::all()` every row, `Student::find(1)` the row with id 1, `Student::findOrFail(99)` the row — or an honest 404 page when it does not exist.
- **Saving is two honest steps:** set the fields on a new model, call `save()` — Laravel writes the INSERT and even fills the timestamps for you.

Code examples

Example 1 — Connect + design + build: .env, migration, migrate

```
# .env — the app's private settings
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=school_app
DB_USERNAME=root
DB_PASSWORD=
```

```
php artisan make:migration create_students_table
```

```
// the migration file's up() method
public function up(): void
{
    Schema::create('students', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('city');
        $table->integer('marks');
        $table->timestamps();
    });
}
```

```
php artisan migrate
```

What it does: `.env` tells Laravel where the database lives. The migration is the students table drawn as code — an id, two text columns, one number column, timestamps. `migrate` reads every blueprint and builds the real tables in MySQL.

What you will see: the terminal confirms each build:

```
   INFO  Running migrations.

  2026_01_01_000001_create_students_table ..... 42ms DONE
```

Open phpMyAdmin and the `students` table stands there, empty, with exactly the five columns you designed.

Example 2 — The Model speaks: save your first student

```
// app/Models/Student.php — Artisan's file, already enough
namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Student extends Model
{
    // One class, one table: Student <-> students
}
```

```
// anywhere in your app (a controller, for now)
$student = new Student;
$student->name = 'Ali Raza';
$student->city = 'Sahiwal';
$student->marks = 88;
$student->save();
```

What it does: The model class is nearly empty — the power comes from the `Model` it extends. Creating a record is plain object work: new model, set fields like properties, `save()`. Laravel translates that into an INSERT and fills `id`, `created_at` and `updated_at` itself.

What you will see: the students table now holds its first row:

```
id | name     | city    | marks | created_at          | updated_at
1  | Ali Raza | Sahiwal | 88    | 2026-01-01 09:00:00 | 2026-01-01 09:00:00
```

Example 3 — Reading back: all(), find(), findOrFail()

```
$students = Student::all();

foreach ($students as $student) {
    echo $student->name . ' — ' . $student->marks . '<br>';
}

$one = Student::find(1);
echo 'First student: ' . $one->name;

$missing = Student::findOrFail(99);
```

What it does: `all()` returns every row as a collection of Student objects — five students after you save Sara (92), Bilal (76), Hira (95) and Usman (64) the same way as Example 2. `find(1)` fetches one by id.

`findOrFail(99)` goes hunting for a student who does not exist — and instead of returning a dangerous null, it stops with a clean 404.

What you will see: the list prints all five names with marks; `First student: Ali Raza`; and the last line never prints — the browser shows Laravel's 404 page, because student 99 does not exist and `findOrFail` refused to pretend otherwise.

Quick recap

- `.env` holds the connection; a **migration** is the table's blueprint in code; `php artisan migrate` builds it.
- A **Model** (Student) speaks for its table (students) — that naming convention is the whole wiring.
- Eloquent: `all()`, `find()`, `findOrFail()`, and new-model + `save()` for writing.
- Rollback exists: migrations can be undone as cleanly as they were built.

Common mistakes

1. **Forgetting to create the empty database** — migrate fails with "Unknown database". Fix: create `school_app` in MySQL/phpMyAdmin first; Laravel builds tables, not the database itself.
2. **Editing a migration that already ran** — migrate skips files it has recorded, so your edit never applies and confusion follows. Fix: make a NEW migration for every change (`add_city_to_students_table`), or `migrate:fresh` on a practice app to rebuild everything (it deletes all data — practice apps only).
3. **Model/table name mismatch** — naming the model `Students` or the table `student` breaks the silent convention. Fix: model singular (`Student`), table plural (`students`) — or declare `protected $table` openly when you must differ.
4. **Raw SQL everywhere out of habit** — hand-written queries in controllers work, but throw away Eloquent's safety and readability. Fix: reach for the model first; drop to raw SQL only when a query truly demands it.

Practice

1. Add a `fees` table by migration (`student_id`, `amount`, `paid_at` nullable) and a `Fee` model. Run migrate, then roll back once and migrate again — write down what `down()` did.
2. Save all five series students through the model, then print a merit list with `Student::all()` sorted by hand in PHP (highest marks first). Confirm the order: Hira 95, Sara 92, Ali 88, Bilal 76, Usman 64.

Practice quiz

1. What are a migration's up() and down() methods?

- up() deletes, down() builds
- up() builds the table, down() undoes it; Laravel runs each migration once and remembers which ones ran
- up() and down() are both optional decoration

- up() is for data, down() is for code

Answer: up() builds the table, down() undoes it; Laravel runs each migration once and remembers which ones ran

2. By convention, which table does the model Student speak for?

- A table called Student
- The students table, plural and lowercase; model singular, table plural, no configuration needed when you follow the naming
- A table called student
- Any table, models have no convention

Answer: The students table, plural and lowercase; model singular, table plural, no configuration needed when you follow the naming

3. What is the difference between Student::find(99) and Student::findOrFail(99) when id 99 does not exist?

- There is none
- find returns null, a dangerous value to use blindly; findOrFail gives an honest 404 page
- find throws an error, findOrFail returns null
- Both return an empty model

Answer: find returns null, a dangerous value to use blindly; findOrFail gives an honest 404 page

---

Watch the Code Scope channel: https://www.youtube.com/@CodeScopePK — this lesson's own video joins the channel when it is recorded.

Read online: https://scopetreasure.com/notes/laravel-models-migrations · More free tools and notes at https://scopetreasure.com