Chapter 11 – Product Migration

Introduction to Laravel migrations

Now, we can just go to the phpMyAdmin application and create tables inside the *online_store* database (such as the product table). It can be done through the fill of a form. It is the **traditional way**. However, it has an issue, it does not allow us to have version control of our database tables and queries. If you have ever had to tell a teammate to manually add a table column to their local database schema after pulling in your changes from source control (such as GitHub), you have faced this issue.

Laravel **migrations** are like version control for our database. They solve the previous issue allowing us to define and share the application's database schema definition.

We won't create database tables in the traditional way. Instead, we will do it through Laravel migrations.

Product migration

Let's create our product migration. In the Terminal, go to the project directory, and execute the following:

Execute in Terminal

```
php artisan make:migration create_products_table
```

The previous command creates a products table migration file inside the *database/migrations* folder. Each migration file name contains a timestamp that allows Laravel to determine the order of the migrations. In our case, it created a file named *2022_02_11_153916_create_products_table.php* .

If you look at the generated file, it contains a migration class. Migration classes contain two methods: *up* and *down* . The *up* method is used to add new tables, columns, or indexes to your database. In contrast, the *down* method should reverse the operations performed by the *up* method.

Now, open the previously generated file. Delete all the existing code and fill it with the following code.

Replace Entire Code

```
<?php


use Illuminate \ Database \ Migrations \ Migration;
use Illuminate \ Database \ Schema \ Blueprint;
use Illuminate \ Support \ Facades \ Schema;


return new class extends Migration
{
	/**
	 * Run the migrations.
	 *
	 * @return void
	 */
	public function up()
	{
		Schema::create('products', function (Blueprint $table) {
			$table->id();
			$table->string('name');
			$table->text('description');
			$table->string('image');
			$table->integer('price');
			$table->timestamps();
		});
	}


	/**
	 * Reverse the migrations.
	 *
	 * @return void
	 */
	public function down()
	{
		Schema::dropIfExists('products');
	}
};
```

Let's analyze the previous code by parts.

Analyze Code

```
	public function up()
```

```
{
    Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->text('description');
            $table->string('image');
            $table->integer('price');
            $table->timestamps();
    });
}
```

The *up* method will create a new database table called *products* . By default, Laravel suggests creating the table names in plural. It is due to how the Laravel ORM "Eloquent" system works (discussed later). The *up* method defines the creation of the *products* table with five columns (*id* , *name* , *description* , *image* , and *price*). Therefore, a *timestamps* method will add two columns (*created_at* and *updated_at*). We also used five column types (*id* , *string* , *text* , *integer* , and *timestamps*). More information about available column types can be found here: https://laravel.com/docs/9.x/migrations#available-column-types.

Analyze Code

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

The *down* method contains the opposite of the *up* method. It drops the *products* table. We will see how to execute the migration classes *up* and *down* methods later. But first, we need to update our database credentials in our Laravel project.

TIP: Always define your database schema using migrations or similar approaches. Remember, they work as version control of your databases. Most web application frameworks provide these kinds of features. For example, Django provides Django Migrations which work like the previous approach.

Modifying the .env file

To execute the migrations, we need to modify the *.env* file (located at the project root folder). In the *.env* file, make the following changes in **bold**. You need to set the *DB_DATABASE* , *DB_USERNAME* , and *DB_PASSWORD* . If you have a different database name, username, or password, make the corresponding changes.

Modify Bold Code

```
...
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=online_store
DB_USERNAME=root
DB_PASSWORD=
...
```

Executing the migrations

To run the migrations, in the Terminal, go to the project directory, and execute the following:

Execute in Terminal

```
php artisan migrate
```

The previous command executes the migrations defined in the *database/migrations* folder. Five migrations are executed. The last one is our "create products table" migration. The other migrations correspond to Laravel default migrations. We will take advantage of some of them in upcoming chapters. If everything works as expected, you should see a result as presented in Fig. 11-1.

```
PS C:\xampp\htdocs\onlineStore> php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_000000_create_users_table
Migrated:  2014_10_12_000000_create_users_table (26.54ms)
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated:  2014_10_12_100000_create_password_resets_table (24.25ms)
Migrating: 2019_08_19_000000_create_failed_jobs_table
Migrated:  2019_08_19_000000_create_failed_jobs_table (24.61ms)
Migrating: 2019_12_14_000001_create_personal_access_tokens_table
Migrated:  2019_12_14_000001_create_personal_access_tokens_table (34.73ms)
Migrating: 2022_02_11_153916_create_products_table
Migrated:  2022_02_11_153916_create_products_table (11.84ms)
```

Figure 11-1. Execution of Laravel migrations.

Verifying the migrations in phpMyAdmin

It is time to verify that the migrations were properly applied in our database. Go to the phpMyAdmin application and click over the *online_store* database (see Fig. 11-2).

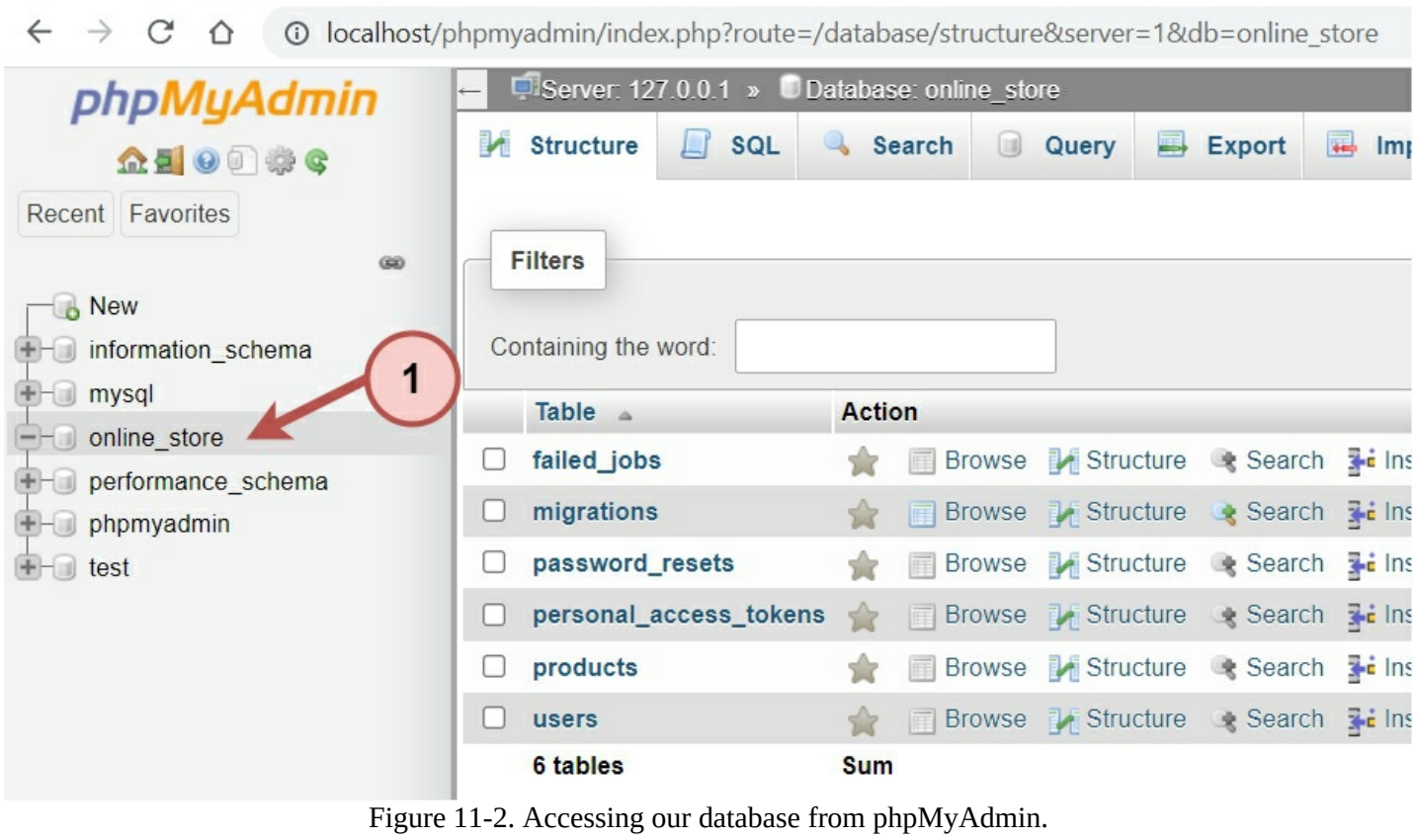

Figure 11-2. Accessing our database from phpMyAdmin.

As you can see, our *products* table appears listed. Migrations worked!

Inserting products

Let's insert four products into our database. For now, we will insert it manually (through SQL queries). Later, we will insert products through a form in an upcoming chapter.

In phpMyAdmin, click the *online_store* database, click the *SQL* tab, paste the following SQL queries, and click *go* (see Fig. 11-3).

Execute in Database

```
INSERT INTO products (id, name, description, image, price, created_at, updated_at) VALUES (NULL, 'TV', 'Best TV', 'game.png', '1000', '2021-10-01 00:00:00', '2021-10-01 00:00:00');
INSERT INTO products (id, name, description, image, price, created_at, updated_at) VALUES (NULL, 'iPhone', 'Best iPhone', 'safe.png', '999', '2021-10-01 00:00:00', '2021-10-01 00:00:00');
INSERT INTO products (id, name, description, image, price, created_at, updated_at) VALUES (NULL, 'Chromecast', 'Best Chromecast', 'submarine.png', '30', '2021-10-01 00:00:00', '2021-10-01 00:00:00');
INSERT INTO products (id, name, description, image, price, created_at, updated_at) VALUES (NULL, 'Glasses', 'Best Glasses', 'game.png', '100', '2021-10-01 00:00:00', '2021-10-01 00:00:00');
```

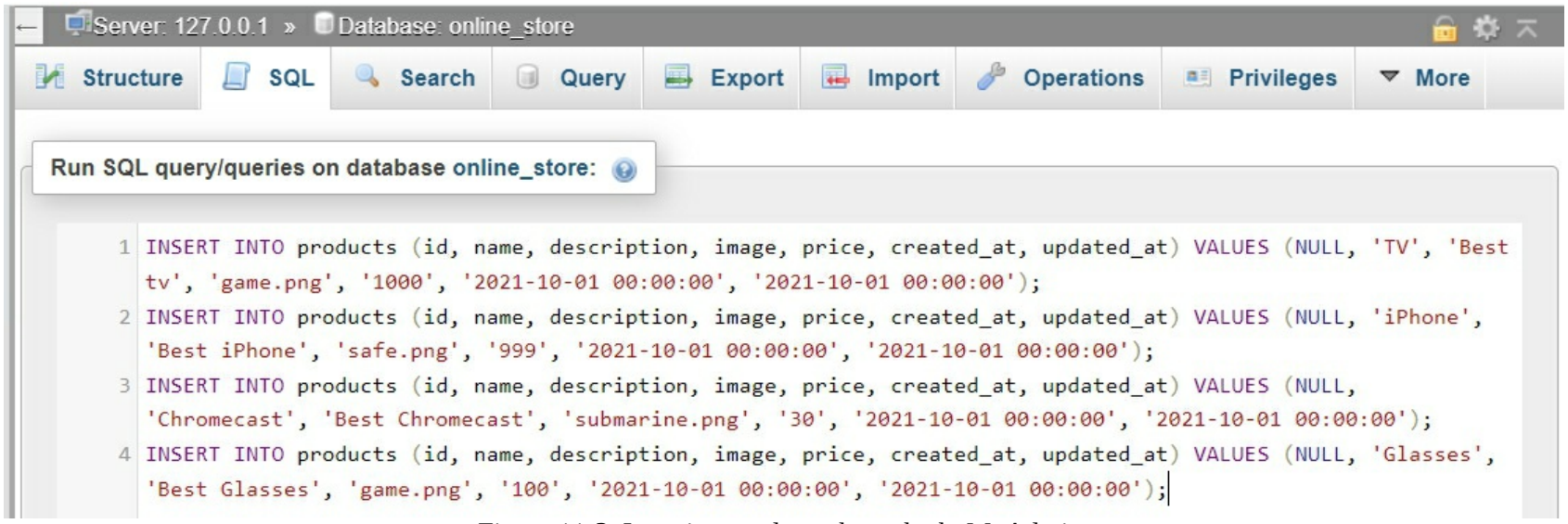

Figure 11-3. Inserting products through phpMyAdmin.

Let's check that the products were successfully inserted. In phpMyAdmin, click the *online_store* database, and click the *products* table. Hopefully, you will see the four products inserted (see Fig. 11-4).

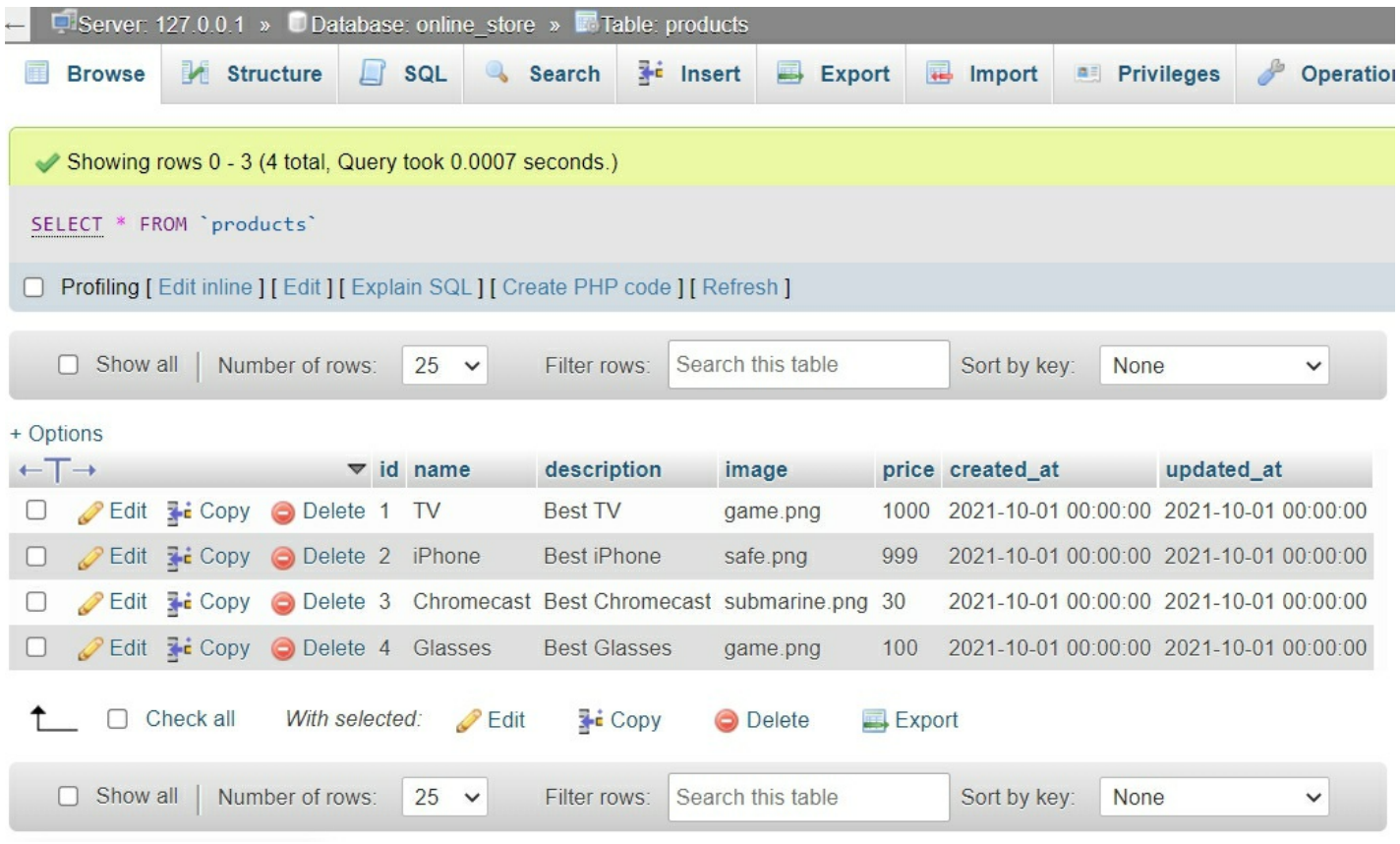

Figure 11-4. Products displayed in phpMyAdmin.