Backend - Laravel

Migrációs és Seeder fájlok

Rostagni Csaba

2025. november 1.

Tartalom

Migráció bevezetés

- A aktuális adatbázist eltárolhatnánk egy sql fájlban
  - Az "aktuális" jelentése eltérhet fejlesztőkként
  - "A" programozó átnevezett egy oszlopot az adatbázisban.
  - "B" prorgamozónál nem működik a kód. Kitörli az adatbázisát, lefuttatja a kódot
  - "B" elvesztette a tesztadatait, írhatjua újra
- Tároljunk minden módosítást külön-külön fájlban
  - Számon kell tartanunk, hogy hol tartottunk
- A migráció lehetővé teszi az adatbázisunk szerkezetének verziókezelését
- Könnyen válthatunk a különböző állapotok között

A migráció működési elve

- Alapértelmezetten a migrációs fájlok a `database/migrations` mappában találhatóak
- A fájlok nevében a létrehozás pontos dátuma és ideje is megtalálható
- A fájlokban egy `up` és egy `down()` metódus található
  - Az `up`-ban azt adhatjuk meg, hogy milyen módosításokat szeretnénk
  - A `down()` azt adja meg hogyan lehet visszacsinálni az `up`-ot
- Az adatbázisban létrehoz magának egy `migrations` táblát, nyomonköveti mely migrációs fájlok lettek már lefuttatva, mely migrációs fájlok lettek egyszerre lefuttatva

Tartalom

Migrációs fájl generálása

CMD

```
php artisan make:migration create_cars_table
```

- Létrehoz egy fájlt a `database/migrations` mappában
- A fájl eleje a pontos dátum és idő, amit a megdott név követ
- A létrehozott fájl nevéből kitalálja, hogy mit szeretnénk
  - `create` Táblát szeretnénk létrehozni
  - `cars_table` A tábla neve `cars` lesz.

A legenerált migrációs fájl up() metódusa

PHP

```php
public function up(): void
{
    Schema::create('cars', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

- Az `Illuminate\Support\Facades\Schema` osztály segítségével hozhatunk létre, vagy módosíthatunk táblát
- Az osztály `create()` metódusával lehet táblát létrehozni
  - Az első paramétere a tábla neve
  - A második paramétere egy "tervrajz" típusú objektum (`Illuminate\Database\Schema\Blueprint`)

Linkek:

- Laravel API: Blueprint

A legenerált migrációs fájl down() metódusa

PHP
```php
public function down(): void
{
    Schema::dropIfExists('cars');
}
```

- A tábla létrehozásának a fordított művelete a tábla törlése
- A `Schema` osztály `dropIfExists()` metódusa törli a táblát, amennyiben létezik

Tartalom

Mező típusok

- `integer()` INT
- `bigInteger()` BIGINT
- `float()` FLOAT
- `string()` VARCHAR, a mérete paraméterként megadható
- `text()` TEXT
- `date()` DATE
- `time()` TIME
- `dateTime()` DATETIME
- `timestamp()` TIMESTAMP
- `boolean()` BOOLEAN (TINYINT(1))

Linkek:

- Laravel Dokumentáció: Mező típusok

Tanulók tábla

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

- Az elnevezés prefixe legyen `create_`, mert most hozzuk létre a táblát
- A tábla neve többesszámban legyen: `students`

PHP
```php
Schema::create('students', function (Blueprint $table) {
    $table->id();                       // UNSIGNED BIGINT
    $table->string("name",60);          // VARCHAR(60)
    $table->date("date_of_birth");      // DATE
    $table->float("average");           // DOUBLE
    $table->boolean("graduated");       // TINYINT(1)
});
```

- A tényleges típusok függenek az adatbázistól is
- Ha nem szükséges, a `timestamps()` törölhető

Tartalom

bigIncrements()

```php
$table->bigIncrements("student_id");
```

- Típusa: UNSIGNED BIGINT
- AUTO_INCREMENT (MySQL-ben automatikusan növekvő értéknek állítja be)
- Elsődleges kulcsnak megfelelő mező
- Az első paraméter a mező neve

Linkek:

- bigIncrements() - Laravel: Migrations

id()

PHP
```php
$table->id();
```

- Háttérben a `bigIncrements()`-t hívja meg
- Típusa: UNSIGNED BIGINT
- AUTO_INCREMENT (MySQL-ben automatikusan növekvő értéknek állítja be)
- alapértelmezetten `id` lesz a mező neve

PHP
```php
$table->id("student_id");
```

- Az alapértelemzett név felülbírálható

Linkek:

- id() - Laravel: Migrations

foreignId()

```php
$table->foreignId('student_id');
```

- Típusa: UNSIGNED BIGINT
- Az első paraméter a mező neve
- Idegenkulcsnak megfelelő, de nem hoz létre idegen kulcs megszorítást

Figyelem!

Az elsődleges kulcsnak és az idegen kulcsnak azonos típusúnak kell lennie! Az `id()` és `foreignId()` megfelelő használata ennek eleget tesz.

Linkek:

- foreignId() - Laravel: Migrations

timestamps()

```php
$table->timestamps();
```

- Két `TIMESTAMP` típusú mezőt hoz létre:
  - `created_at` és
  - `updated_at` néven

Figyelem!

Nem összekeverendő a `timestamp()` metódussal, ami csak egy TIMESTAMP típusú mezőt hoz létre!

Linkek:

- timestamps() - Laravel: Migrations

softDeletes()

```php
$table->softDeletes();
```

- Alapértelmezetten egy `TIMESTAMP` típusú mezőt hoz létre `deleted_at` néven

```php
$table->softDeletes("torles_ideje");
```

- Az első paraméter a mező neve, megadása opcionális

Figyelem!

Soft delete során az adatok nem kerülnek törlésre, az adatbázisban megmaradnak, csak egy jelzést kapnak, hogy törölve lettek, ezt majd a lekérdezésekkor figyelembe kell venni.

Linkek:

- softDeletes() - Laravel: Migrations

Tartalom

Mező módosítók

PHP
```php
Schema::table('users', function (Blueprint $table) {
    $table->string('email', 40)->nullable();
});
```

- `comment('megjegyzés')` A mezőhöz ad hozzá megjegyzést
- `nullable($value = true)` A mező értéke lehet null
- `default($value)` A mező alapértelmezett értéke
- `autoIncrement()` AI
- `charset('utf8mb4')` karakterkészlet
- `unsigned()` a szám nem tartalmazhat negatív értéket
- `useCurrent()` a dátum mező vegye fel az aktuális értéket
- `after('column')` a megadott mező után szúrja be (ahol támogatott)

Linkek:

- Laravel Dokumentáció: Mező módosítók

Tartalom

Migráció futtatása

CMD
```
php artisan migrate
```

- A még le nem futtatott migrációk futtatása

CMD
```
php artisan migrate:status
```

- A migráció állapota

CMD
```
php artisan migrate --force
```

- A migráció futtatása kikényszeríthető az adatvesztéssel járó migrációk esetén is

A migráció visszaállítása

CMD
```
php artisan migrate:rollback
```

- A legutóbbi migrálás során történt módosítások visszavonása
- Egyszerre többet is, ha azok egy lépésben futottak le

CMD
```
php artisan migrate:rollback --step=5
```

- A legutóbbi n darab lépés visszaállítása

CMD
```
php artisan migrate:reset
```

- Az összes migráció visszaállítása

Teljes adatbázis frissítése

CMD
```
php artisan migrate:refresh
```

- Az összes migrációt visszagörgeti,
- majd futtatja a migrációkat.

CMD
```
php artisan migrate:fresh
```

- Az adatbázis összes tábláját törli,
- majd futtatja a migrációkat.

Tanulók tábla

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

- Az elnevezés prefixe legyen `create_`, mert most hozzuk létre a táblát
- A tábla neve többesszámban legyen: `students`

PHP
```php
Schema::create('students', function (Blueprint $table) {
    $table->id();                       // UNSIGNED BIGINT
    $table->string("name",60);          // VARCHAR(60)
    $table->date("date_of_birth");      // DATE
    $table->float("average");           // DOUBLE
    $table->boolean("graduated");       // TINYINT(1)
});
```

- A tényleges típusok függenek az adatbázistól is
- Ha nem szükséges, a `timestamps()` törölhető

Mit fog lefuttatni a migráció?

CMD

```
php artisan migrate --pretend
```

- A `--pretend` kapcsolóval megkapjuk a futtatandó SQL kódot
- Valójában nem futtatja le, csak színleli
- Csak akkor ír ki bármit is, ha van mit csinálni

MySQL

```
create table `students` (
    `id` bigint unsigned not null auto_increment primary key,
    `name` varchar(60) not null,
    `date_of_birth` date not null,
    `average` float(53) not null,
    `graduated` tinyint(1) not null)
    default character set utf8mb4 collate 'utf8mb4_unicode_ci'
```

- Minden mezőt kötelező kitölteni
- A `FLOAT(53)` helyett az adatbázisban `DOUBLE` lesz (MySQL 8.0.37)

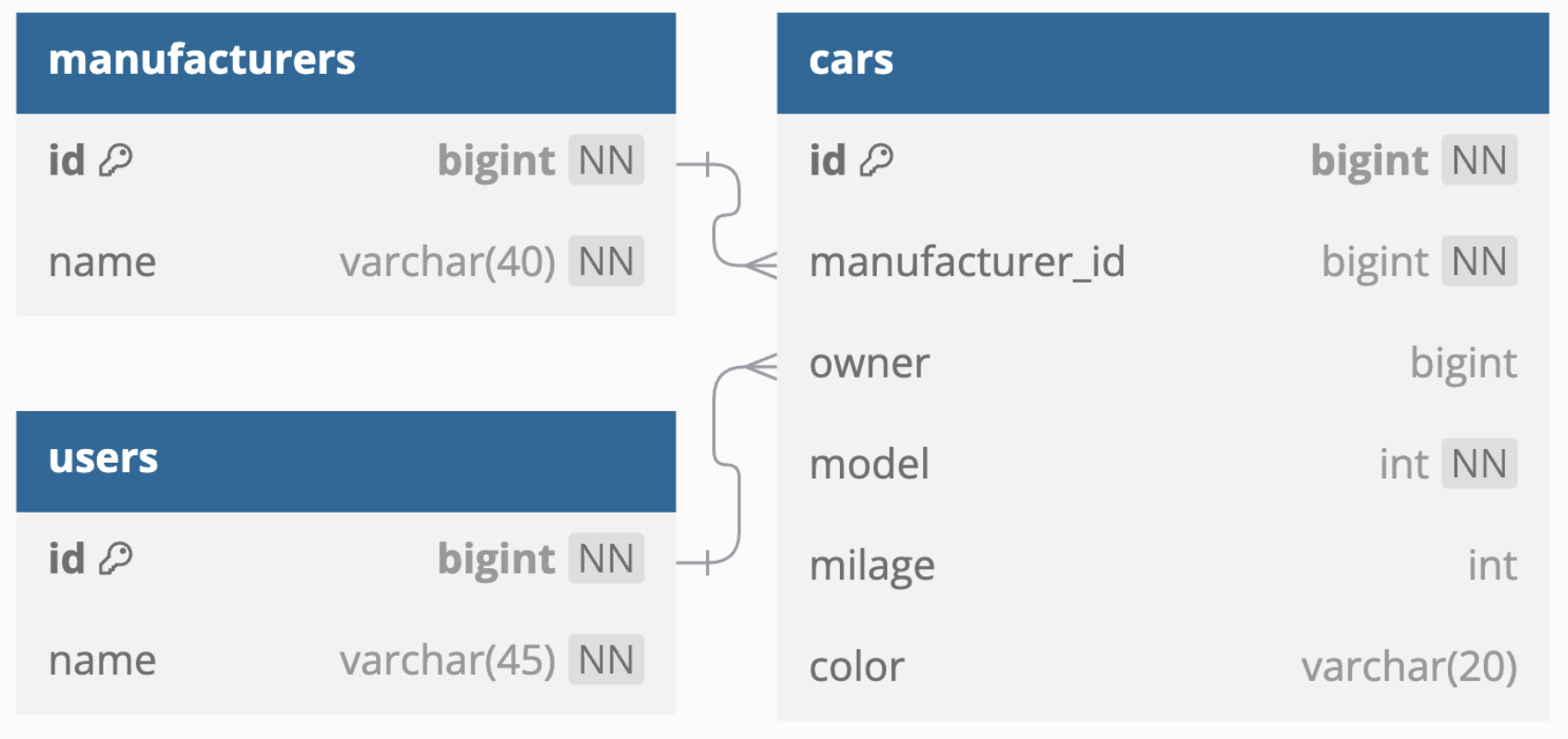
Tartalom

Autók és tulajdonosok DBML

```
Table "manufacturers" {
    "id" bigint [pk, not null, increment]
    "name" varchar(40) [not null]
}
Table "cars" {
    "id" bigint [pk, not null, increment]
    "manufacturer_id" bigint [not null]
    "owner" bigint
    "model" int [not null]
    "milage" int
    "color" varchar(20)
}
Table "users" {
    "id" bigint [pk, not null, increment]
    "name" varchar(45) [not null]
}

Ref "FK_cars_manufacturers_1":
    "cars"."manufacturer_id" > "manufacturers"."id" [update: cascade, delete: casca
Ref "FK_cars_users_1":
    "cars"."owner" > "users"."id" [update: cascade, delete: cascade]
```

Autók és tulajdonosok UML



Idegen kulcs mező létrehozása

PHP
```php
Schema::table('cars', function (Blueprint $table) {
    $table->unsignedBigInteger('owner')->nullable();

    $table->foreign('owner')->references('id')->on('users');
});
```

- Az idegen kulcs létrehozása történhet külön migrációs fájl által
- Elsőként a szülő táblát kell létrehozni, utána a gyerek táblánál megadható az idegen kulcs
- A `Schema::table()` metódusa egy már meglévő táblát módosít
- Az `unsignedBigInteger()` létrehoz idegen kulcsnak megfelelő mezőt, **aminek a típusa megegyezik a hivatkozott tábla elsődleges kulcsával**!

Linkek:

- Idegenkulcs megszorítás - Laravel: Migrations

Idegen kulcs megszorítás

```php
Schema::table('cars', function (Blueprint $table) {
    $table->unsignedBigInteger('owner')->nullable();

    $table->foreign('owner')->references('id')->on('users');
});
```

- A `foreign()` metódus megadja, hogy a (már korábban létrehozott) mező egy idegen kulcs
- A `references()` megadja, hogy a másik táblának melyik oszlopára hivatkozik
- Végül az `on()` a táblát határozza meg

Idegen kulcs megszorítás - constrained("tabla","mezo")

PHP

```php
Schema::table('cars', function (Blueprint $table) {
    $table->foreignId('owner')->nullable()
          ->constrained('users','id');
});
```

- A `foreigId()` metódus egy `UNSIGNED BIGINT`-nek megfelelő típust hoz létre
- A `constrained()` metódus meghívható a mező létrehozásakor, így egy lépésben létrehozható a mező és az idegen kulcs megszorítás is
  - Az első paramétere a tábla neve amire hivatkozik az idegen kulcs
  - A második paraméter a hivatkozott mező
- A `constrained()` a módosítók (`charset()`, `unsigned()`, `nullable()`, stb. . .) **után** álljon!

Idegen kulcs megszorítás - constrained()

PHP

```php
Schema::table('cars', function (Blueprint $table) {
    $table->foreignId('manufacturer_id')
          ->constrained();
});
```

- A tábla nevét megfelelő elnevezés esetén ki tudja találni
  - `manufacturer_id` mező név esetén a tábla `manufacturers` lesz
- Alapértelmezetten `id`-t veszi hivatkozandó mezőnek

ON DELETE, ON UPDATE

PHP
```php
Schema::table('cars', function (Blueprint $table) {
    $table->foreignId('owner')->nullable()
          ->constrained('users','id')
          ->onUpdate('cascade')
          ->onDelete('cascade');
});
```

- `CASCADE`: A "fő" tábla módosításakor a "gyerekben" is végrehajtódik a módosítás
- Amennyiben a `users` táblában az `id` mező értéke megváltozna, úgy az őrá hivatkozó `owner` mező is módosul a `cars` táblában
- A fenti példában módosítás és törlés esetén is hasonlóan viselkedik
- Továbbá: `RESTRICT`, `NO ACTION`, `SET NULL`, `SET DEFAULT`

Linkek:

- MySQL 8 dokumentáció: Foreig key contraints

Idegen kulcs törlése

PHP
```php
Schema::table('cars', function (Blueprint $table) {
    $table->dropForeign('cars_owner_foreign');
}
```

- A laravel által generált nevet használhatjuk
  - A tábla nevével kezdődik
  - Aláhúzás követi, majd a mező neve
  - Ezt is aláhúzás követi, végül a `foreign` szó

PHP
```php
Schema::table('cars', function (Blueprint $table) {
    $table->dropForeign(['owner']);
}
```

- Tömbként megadhatjuk az idegenkulcs mezőjét (itt `owner`)
- A mező nevéből kitalálja az idegenkulcsot

Tartalom

Seeder intro

Bizonyos adatokra szükségünk lehet egy alklamazásban, amiket nem szeretnénk egyesével felvinni az adatbázisba.

- Kategórák
- Menüpontok
- Jogosultsági szintek
- Címkék

Seeder generálása

CMD

```
php artisan make:seeder CarSeeder
```

- Létrehoz egy fájlt a `database/seeders` mappában

Seeder példa

PHP

```php
class CarSeeder extends Seeder
{
    public function run()
    {
        DB::table("cars")->insert([
            'plate_number' => 'AAA-555',
            'manufacturer_id' => 1,
            'model' => 'Jazz',
            'category' => 'M1',
            'fuel_type' => 'hibrid',
            'consumption' => 8.00,
        ]);
    }
}
```

Seeder használata

CMD
```
php artisan db:seed
```

- Lefuttatja a `Database\Seeders\DatabaseSeeder` fájlban lévő seedereket

CMD
```
php artisan db:seed  --class=CarSeeder
```

- Lefuttatja a `CarSeeder` seedert

DatabaseSeeder

PHP
```php
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call(CarSeeder::class);
    }
}
```

- Az `artisan db:seed` csak a `DatabaseSeeder`-nek a `run()` metódusát futtatja le.
- Benne a `$this->call()` metódussal hívható meg másik migreációs osztály
- Mivel a seederek azonos névtérben vannak nem szükséges a `use` használata

Migárció futtatása seederrel

CMD

```
php artisan migrate:refresh --seed
```

- Az összes **migrációt visszagörgeti**,
- majd futtatja a migrációkat.
- Végül futtatja a seedereket is.

CMD

```
php artisan migrate:fresh --seed
```

- Az adatbázis **összes tábláját törli**,
- majd futtatja a migrációkat.
- Végül futtatja a seedereket is.