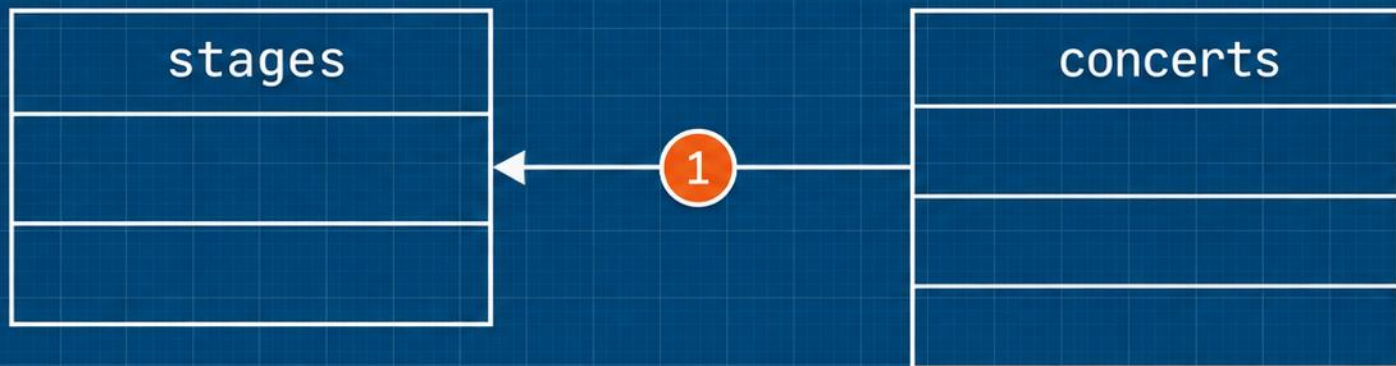


Bloc Laravel • chapitre 11

Modèles : deux tables

Les scènes reçoivent leur propre table, stages. Chaque concert ne garde que le numéro de sa scène.



Une majuscule de trop, et le concert change de scène

Au chapitre 10, chaque concert recopie le nom de sa scène dans la colonne stage. Vous ajoutez un quatrième concert, sur la « Grande Scène ».

routes/web.php		navigateur · /bac-a-sable	
1	<code>Concert::create([</code>		
2	<code> 'artist' => 'Les Cordes Sauvages',</code>	4	<code>// routes/web.php:190</code>
3	<code> 'stage' => 'Grande Scène',</code>		
4	<code> ...</code>	2	<code>// routes/web.php:191</code>
5	<code>]);</code>		
6			
7	<code>dump(Concert::count());</code>		
8	<code>dd(Concert::where('stage', 'Grande scène')->count());</code>		

La table contient quatre concerts, mais `where()` n'en trouve toujours que deux sur la Grande scène : pour SQLite, S n'est pas s.

Une scène a ses propres données : elle a besoin de sa table



L'organisatrice a deux demandes.

- Renommer la Grande scène en « Scène du Lac » : il faut modifier chaque concert qui porte ce nom.
- Afficher la capacité de la scène, 3 000 places : il faudrait la recopier dans chaque concert.

Une scène accueille plusieurs concerts, et chaque concert se joue sur une seule scène.

La table stages range une scène par ligne

Vous créez d'abord la table des scènes, avec une migration comme celle du chapitre 10.

terminal

```
$ php artisan make:migration create_stages_table
```

database/migrations/..._create_stages_table.php

```
1 Schema::create('stages', function (Blueprint $table) {  
2     $table->id();  
3     $table->string('name');  
4     $table->unsignedSmallInteger('capacity');  
5     $table->string('location');  
6     $table->timestamps();  
7 });
```

1

Un nombre de places.

2

L'endroit de la scène dans le parc.

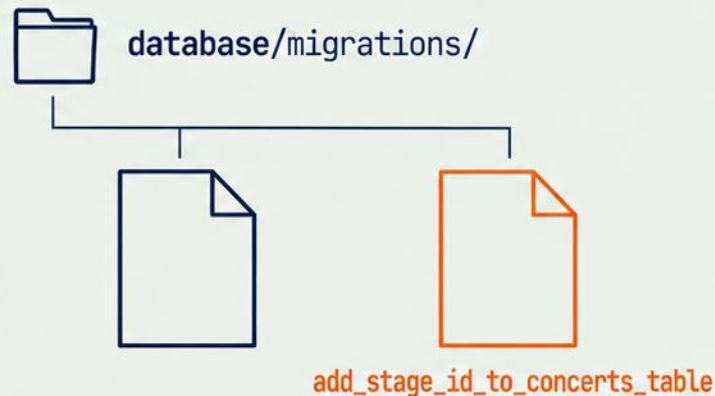
Pour changer une table existante, écrivez une nouvelle migration

La table concerts existe déjà. L'option `--table=concerts` dit à Artisan quelle table la migration modifie.

terminal

```
$ php artisan make:migration add_stage_id_to_concerts_table  
--table=concerts
```

INFO Migration [`database/migrations/2026_09_28_220038_add_stage_id_to_concerts_table.php`] created successfully.



Le fichier créé appelle `Schema::table('concerts', ...)`, qui modifie une table existante. `Schema::create()` en crée une.

foreignId('stage_id') ajoute une clé étrangère

Une **clé étrangère** est une colonne qui contient l'identifiant d'une ligne d'une autre table.

database/migrations/..._add_stage_id_to_concerts_table.php

```
1 public function up(): void
2 {
3     Schema::table('concerts', function (Blueprint $table) {
4         $table->foreignId('stage_id')->constrained()->cascadeOnDelete();
5         $table->dropColumn('stage');
6     });
7 }
```

1

Une colonne d'entiers :
l'id d'une ligne de stages.

2

La base refuse un stage_id
qui ne désigne aucune scène.

3

Supprimer une scène
supprime ses concerts.

dropColumn('stage') supprime la colonne de texte, remplacée par la clé.

Sur une table remplie, l'ajout de `stage_id` échoue

terminal

```
$ php artisan migrate
```

```
2026_09_28_220036_create_stages_table ..... 7.71ms DONE
```

```
2026_09_28_220038_add_stage_id_to_concerts_table ..... 4.15ms FAIL
```

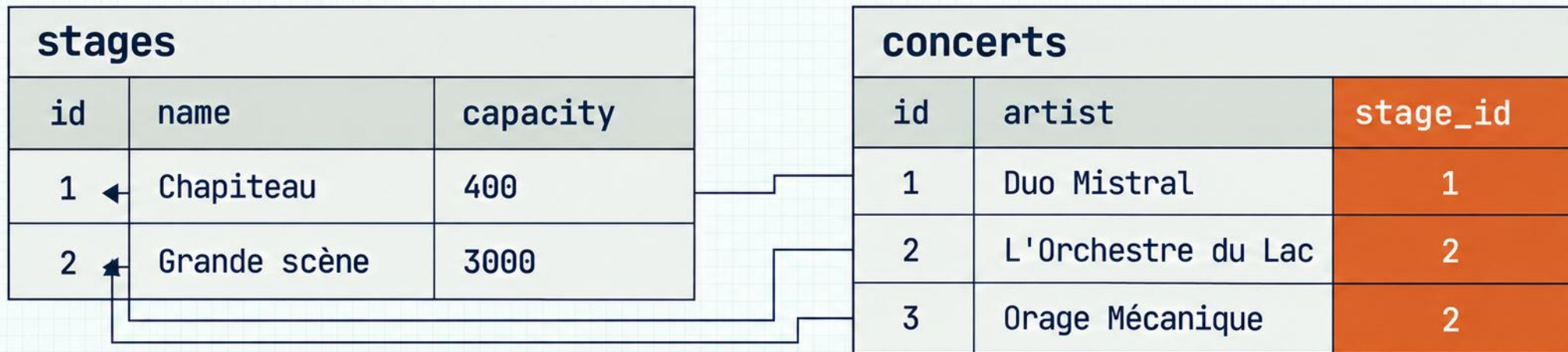
NON CONFORME

```
SQLSTATE[HY000]: General error: 1 Cannot add a NOT NULL column with default value NULL (...)
```

Les quatre concerts de la table recevraient `null` dans la nouvelle colonne, et `stage_id` refuse `null`.

Dans ce chapitre, videz la base avec `php artisan migrate:fresh` : sur une table vide, l'ajout réussit. Jamais sur une base dont vous gardez les données.

stage_id contient l'id d'une ligne de stages



terminal

```
$ php artisan db:table concerts
```

```
...
```

```
Foreign Key ..... On Update / On Delete  
stage_id references id on stages ..... no action / cascade
```

La colonne stage a disparu. Chaque concert désigne sa scène par un numéro.

hasMany : une scène a plusieurs concerts

Une **relation** Eloquent est une méthode de modèle qui décrit comment retrouver, dans une autre table, les lignes liées à l'objet.

app/Models/Stage.php

```
1  #[Fillable(['name', 'capacity', 'location'])]
2  class Stage extends Model
3  {
4      public function concerts(): HasMany
5      {
6          return $this->hasMany(Concert::class);
7      }
8  }
```

1

Au pluriel : la méthode renvoie plusieurs concerts.

2

Cherche dans concerts les lignes dont stage_id vaut l'id de la scène.

belongsTo : un concert appartient à une scène

Dans Concert, 'stage' devient 'stage_id' dans #[Fillable], et la classe reçoit la méthode stage().

app/Models/Concert.php

```
1  #[Fillable(['artist', 'stage_id', 'starts_at', 'duration', 'description'])]
2  class Concert extends Model
3  {
4      ...
5      public function stage(): BelongsTo
6      {
7          return $this->belongsTo(Stage::class);
8      }
9  }
```

1

Au singulier : la méthode renvoie une seule scène.

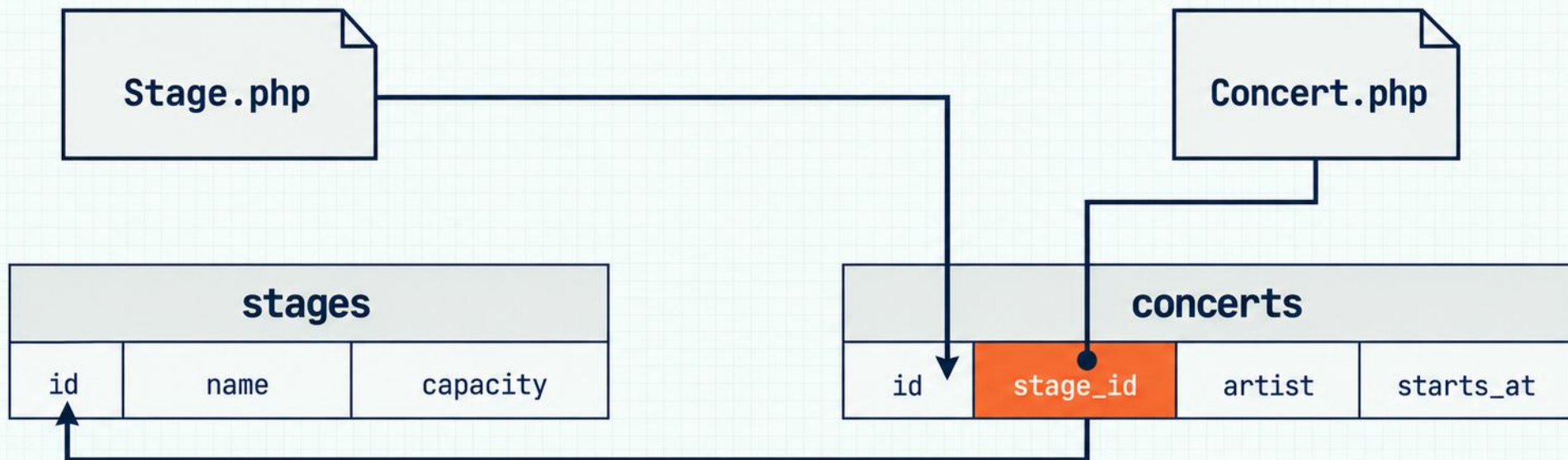
2

Lit le stage_id du concert, puis cherche cette ligne dans stages.

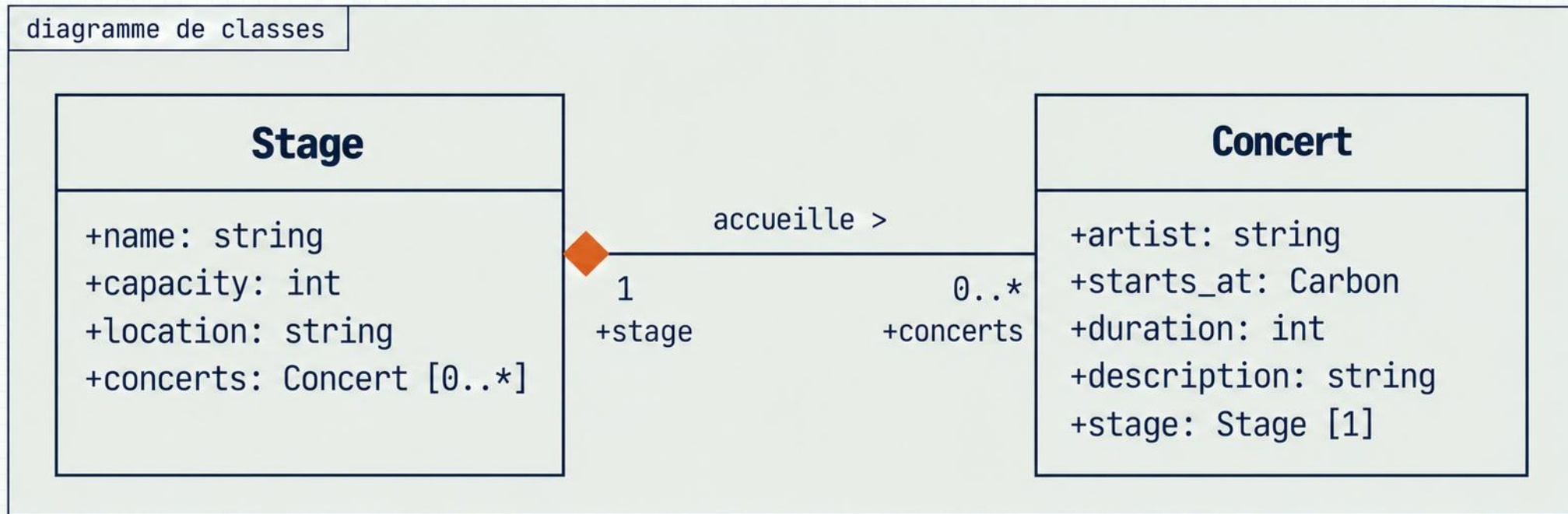
Les deux relations passent par la colonne `stage_id`

Vous n'écrivez aucun nom de colonne : Eloquent le déduit.

- `hasMany` part du nom de la classe `Stage` : `stage`, suivi de `_id`.
- `belongsTo` part du nom de la méthode `stage()` : `stage`, suivi de `_id`.
- Une méthode appelée `scene()` chercherait une colonne `scene_id`, qui n'existe pas.



Dans le diagramme de classes, c'est une composition



Le losange plein, collé à Stage, marque une **composition** : un concert n'existe pas sans sa scène et disparaît avec elle. `cascadeOnDelete()` est sa traduction en base.

La base refuse un concert sur une scène qui n'existe pas

La table stages est encore vide. Vous essayez d'enregistrer un concert sur la scène 1.

routes/web.php	
1	<code>Concert::create([</code>
2	<code> 'stage_id' => 1,</code>
3	<code> 'artist' => 'Les Cordes Sauvages',</code>
4	<code> ...</code>
5	<code>]);</code>

navigateur · /bac-a-sable	
<code>Illuminate\Database\QueryException</code> <code>SQLSTATE[23000]: Integrity constraint violation: 19 FOREIGN KEY constraint failed (...)</code>	

NON CONFORME

C'est `constrained()` qui a posé cette règle. **Une clé étrangère désigne toujours une ligne qui existe.**

Une fabrique produit des objets avec des valeurs inventées

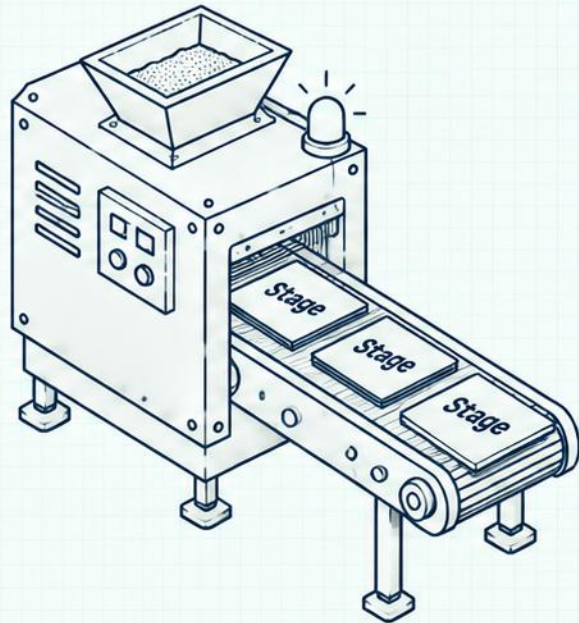
Un vrai programme compte une vingtaine de concerts. Une **fabrique** est une classe qui produit des objets d'un modèle avec des valeurs inventées.

terminal

```
$ php artisan make:factory StageFactory --model=Stage
```

database/factories/StageFactory.php

```
1 public function definition(): array
2 {
3     return [
4         'name' => fake()->randomElement(['Kiosque', 'Clairière', 'Verger', 'Serre']),
5         'capacity' => fake()->numberBetween(100, 2000),
6         'location' => fake()->streetName(),
7     ];
8 }
```



definition() renvoie un tableau avec une clé par colonne. La fabrique l'appelle une fois pour chaque objet à produire.

fake() invente une nouvelle valeur à chaque appel

fake() renvoie un objet de **Faker**, une bibliothèque qui invente des données, installée dans chaque projet Laravel.

Vous écrivez	Faker renvoie
fake()->randomElement([...])	un élément du tableau, pris au hasard
fake()->numberBetween(100, 2000)	un entier entre les deux bornes comprises
fake()->streetName()	un nom de rue
fake()->name()	un prénom et un nom
fake()->dateTimeBetween('...', '...')	une date et une heure entre deux moments
fake()->sentence()	une phrase de faux latin

Sans le trait HasFactory, Stage::factory() n'existe pas

navigateur • /bac-a-sable

BadMethodCallException

Call to undefined method App\Models\Stage::factory()

NON CONFORME

La méthode statique factory() vient du trait HasFactory. make:model ne l'ajoute pas : collez-le dans Stage, et de même dans Concert.

app/Models/Stage.php

```
1  use Illuminate\Database\Eloquent\Factories\HasFactory;
2
3  #[Fillable(['name', 'capacity', 'location'])]
4  class Stage extends Model
5  {
6      use HasFactory;
7      ...
8  }
```

factory()->count(3)->create() enregistre trois scènes

routes/web.php

```
1 Route::get('/bac-a-sable', function () {  
2     dd(Stage::factory()->count(3)->create());  
3 });
```

navigateur · /bac-a-sable

```
0 => App\Models\Stage {#1548 ▼  
...  
    #attributes: array:6 [▼  
        "name" => "Verger"  
        "capacity" => 472  
        "location" => "Lane Harbors"  
    ...
```

count(3) demande trois objets, et create() les enregistre puis les renvoie dans une collection. Chez vous, les valeurs sont différentes.

4 • Les fabriques et Faker

APP_FAKER_LOCALE fixe la langue des données inventées

« Lane Harbors » est une rue américaine : la ligne vaut **en_US** dans un projet neuf. Passez au français de Belgique.

.env

1 APP_FAKER_LOCALE=fr_BE

navigateur • /bac-a-sable

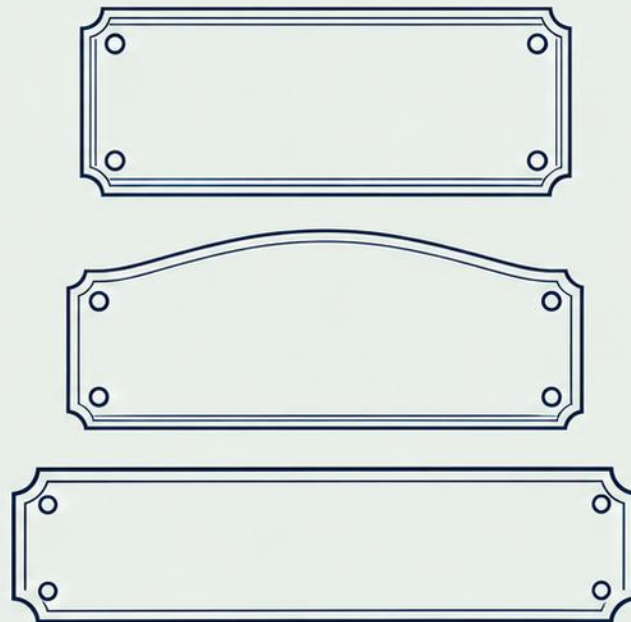
"location" => "chemin Lecomte"

...

"location" => "impasse Claeys"

...

"location" => "place Dierckx"



Les noms de personnes suivent aussi ce réglage, mais **sentence()** reste en faux latin.

'stage_id' => Stage::factory() crée la scène du concert

Un concert a besoin d'une scène. La fabrique des concerts en demande une à la fabrique des scènes.

database/factories/ConcertFactory.php

```
1 public function definition(): array
2 {
3     return [
4         'stage_id' => Stage::factory(), 1
5         'artist' => fake()->name(),
6         'starts_at' => fake()->dateTimeBetween('2026-10-16 17:00', '2026-10-16 23:00')->format('Y-m-d H:00:00'), 2
7         'duration' => fake()->randomElement([45, 60, 75, 90]),
8         'description' => fake()->sentence(),
9     ];
10 }
```

1 Laravel enregistre une nouvelle scène, puis place son id ici.

2 Une heure ronde du vendredi soir, entre 17 h et 23 h.

Le seeder écrit les vrais concerts, la fabrique invente les autres

StageSeeder enregistre les deux vraies scènes. ConcertSeeder retrouve chaque scène par son nom.

database/seeder/ConcertSeeder.php

```
1  $tent = Stage::where('name', 'Chapiteau')->first();
2  ...
3  Concert::create([
4      'stage_id' => $tent->id,
5      'artist' => 'Duo Mistral',
6      ...
7  ]);
8  ...
9  foreach (Stage::all() as $stage) {
10     Concert::factory()->count(10)->create(['stage_id' => $stage->id]);
11 }
```

1

Le numéro du Chapiteau.

2

Le tableau remplace les valeurs de definition().

Les scènes d'abord, les concerts ensuite

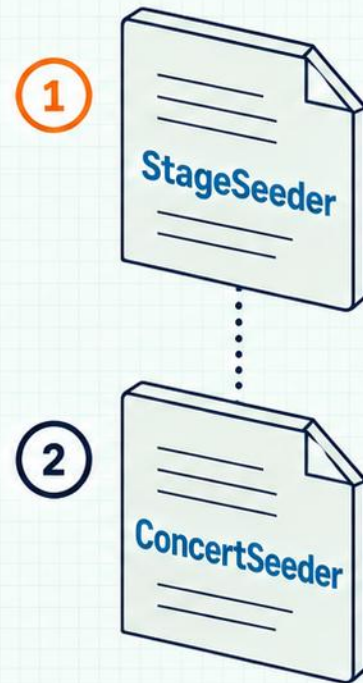
```
database/seeder/DatabaseSeeder.php
```

```
1 $this->call(StageSeeder::class);  
2 $this->call(ConcertSeeder::class);
```

Dans l'ordre inverse, stages serait vide, et first() renverrait null.
Après php artisan migrate:fresh --seed :

```
navigateur · /bac-a-sable
```

```
2 // routes/web.php:183  
  
23 // routes/web.php:184
```



Deux scènes et vingt-trois concerts : trois écrits à la main, et dix de fabrique par scène.

**Écrivez à la main les concerts que les pages doivent toujours montrer,
et laissez une fabrique inventer les lignes dont les valeurs important peu.**

\$concert->stage renvoie la scène du concert

routes/web.php		navigateur • /bac-a-sable	
1	<code>\$concert = Concert::find(1);</code>	1	// routes/web.php:185
2			
3	<code>dump(\$concert->stage_id);</code>	"Chapiteau"	// routes/web.php:186
4	<code>dump(\$concert->stage->name);</code>		
5	<code>dd(\$concert->stage()->toRawSql());</code>	"select * from "stages" where "stages"."id" = 1"	// routes/web.php:189

Sans parenthèses, `$concert->stage` envoie la requête et renvoie un objet Stage. Avec parenthèses, `$concert->stage()` renvoie la relation, qui décrit la recherche sans l'envoyer.

\$stage->concerts renvoie les concerts de la scène

De l'autre côté, la relation renvoie une collection. Avec parenthèses, vous la complétez comme une requête du chapitre 10.

routes/web.php

```
1 $stage = Stage::find(1);
2
3 dump($stage->concerts->count());
4
5 foreach ($stage->concerts()->orderBy('starts_at')->get() as $concert) {
6     dump($concert->starts_at->format('G \h') . ' ' . $concert->artist);
7 }
```

navigateur · /bac-a-sable

```
11 // routes/web.php:186

"17 h Léa Bodart" // routes/web.php:190
...
"18 h Duo Mistral" // routes/web.php:190
...
```

Le Chapiteau a onze concerts : Duo Mistral et dix concerts de fabrique.

Sans parenthèses, les données. Avec parenthèses, la relation.

Vous écrivez	Vous recevez
<code>\$concert->stage</code>	un objet Stage
<code>\$concert->stage()</code>	une relation BelongsTo, sans requête envoyée
<code>\$stage->concerts</code>	une collection d'objets Concert, peut-être vide
<code>\$stage->concerts()</code>	une relation HasMany, à compléter avec <code>orderBy()</code> , <code>get()</code> ou <code>count()</code>

Lisez la propriété pour afficher une donnée, et appelez la méthode pour compléter la requête.

Supprimer une scène supprime ses concerts

Vous vérifiez la cascade choisie dans la migration, avec `cascadeOnDelete()`.

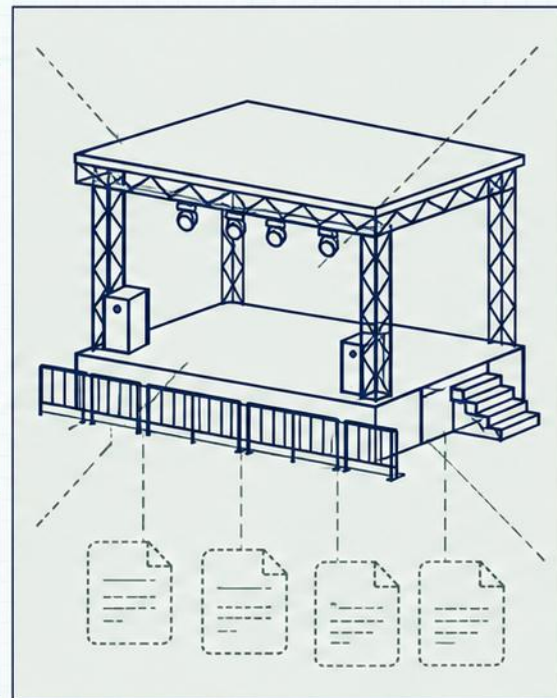
routes/web.php

```
1 dump(Concert::count());
2
3 Stage::where('name', 'Grande scène')->first()->delete();
4
5 dd(Concert::count());
```

navigateur · /bac-a-sable

```
23 // routes/web.php:183
```

```
11 // routes/web.php:187
```



Aucune ligne de PHP ne supprime de concert. La base a supprimé elle-même les douze concerts de la Grande scène.

{{ \$concert->stage }} affiche un objet : écrivez ->name

Le composant **concert-card** écrit toujours `{{ $concert->stage }}`, qui renvoie maintenant un objet **Stage**.

terminal

```
$ curl -s http://premier-test-laravel.test/concerts
```

...

```
<p>{"id":1,"name":"Chapiteau","capacity":400,...}, 18 h</p>
```



resources/views/components/concert-card.blade.php

```
1 <p>{{ $concert->stage->name }}, {{ $concert->starts_at->format('G \h') }}</p>
```

La page affiche de nouveau **Chapiteau**, **18 h**. **ConcertController** n'a pas changé.

La page /scenes/{id} liste les concerts d'une scène

app/Http/Controllers/StageController.php

```
1 public function show(int $id): View
2 {
3     $stage = Stage::findOrFail($id);
4     $concerts = $stage->concerts()->orderBy('starts_at')->get();
5
6     return view('stages.show', ['stage' => $stage, 'concerts' => $concerts]);
7 }
```

1 Répond 404 pour un numéro inconnu.

2 Les concerts de cette scène, triés par heure.

routes/web.php

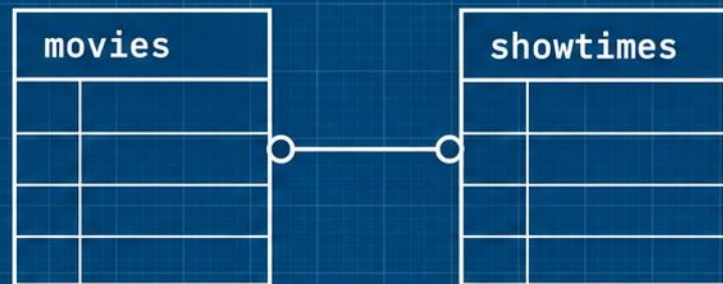
```
1 Route::get('/scenes/{id}', [StageController::class, 'show'])->whereNumber('id')->name('stages.show');
```

La vue stages/show.blade.php parcourt \$concerts avec @forelse, et affiche « Aucun concert sur cette scène » dans @empty.

À vous : les séances du cinéma, dans eloquent-26

90 minutes, en séance ou chez vous. Les films reçoivent leurs séances : une table `showtimes` reliée à `movies`.

- Katas 11.1 et 11.2 : la table `showtimes`, les deux relations.
- Katas 11.3 et 11.4 : les fabriques, le seeder.
- Katas 11.5 et 11.6 : la page `/seances`, la fiche complétée.
- Exercice 11.7 : prédire cinq lectures, sur papier.



```
php artisan test --group=chapitre11
```