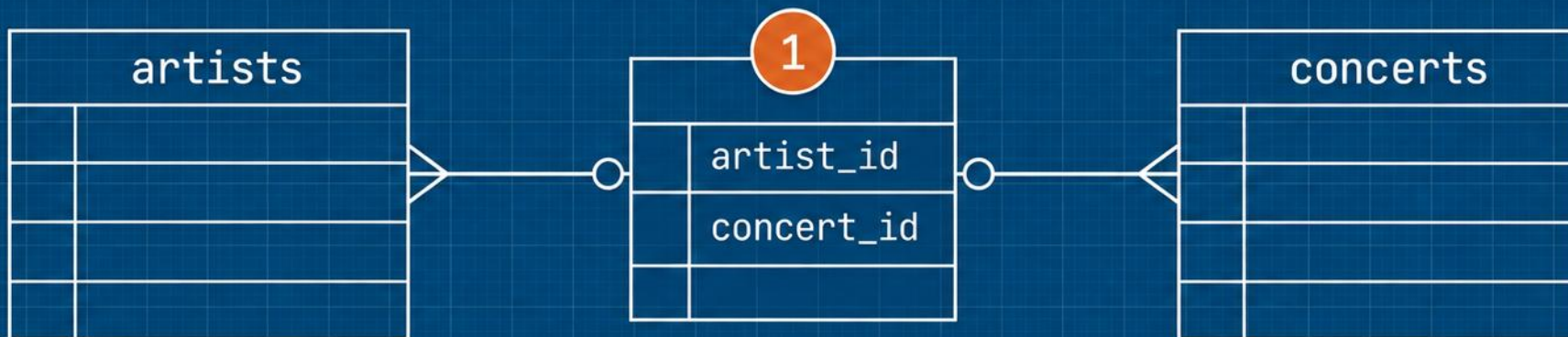


# Bloc Laravel · chapitre 12

## Modèles : plusieurs à plusieurs

Les invités reviennent avec leur table, artists, et une troisième table qui relie les artistes aux concerts.



1 • Deux listes qui se croisent

# Ana Lopes joue à deux concerts, et un concert reçoit deux invités

Au chapitre Views, les invités étaient une liste dans le tableau \$concerts. Ana Lopes est maintenant invitée aussi au concert de Duo Mistral.

première idée • une colonne de texte

| concerts |                    |                         |
|----------|--------------------|-------------------------|
| id       | artist             | guests                  |
| 1        | Duo Mistral        |                         |
| 2        | L'Orchestre du Lac | Ana Lopes, Karim Benali |
| 3        | Orage Mécanique    | Les Tambours du Parc    |

seconde idée • une clé étrangère

| artists |                      |            |
|---------|----------------------|------------|
| id      | name                 | concert_id |
| 1       | Ana Lopes            | 2          |
| 2       | Karim Benali         | 2          |
| 3       | Les Tambours du Parc | 3          |

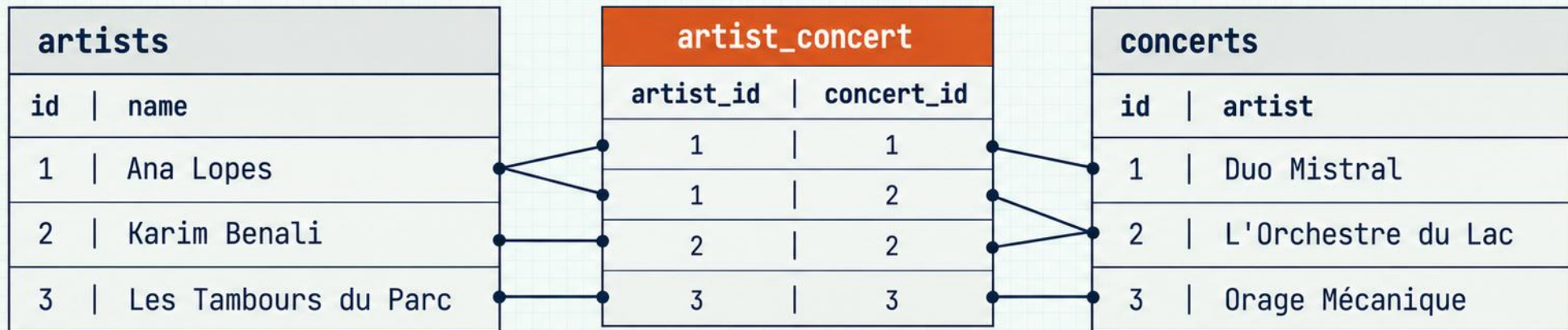
Dans les deux cas, ajouter Ana Lopes au concert 1 recopie son nom.

## 1 • Deux listes qui se croisent

# Une table pivot ne contient que des paires de numéros

Entre un concert et ses invités, la multiplicité vaut 0..\* des deux côtés : c'est la relation plusieurs à plusieurs du chapitre 4 de la POO.

trois tables



Chaque ligne du milieu se lit « cet artiste est invité à ce concert ». Le numéro 1 d'Ana Lopes y apparaît deux fois, son nom une seule fois.

# La table artists range un artiste par ligne

terminal

```
$ php artisan make:migration create_artists_table
```

database/migrations/...\_create\_artists\_table.php

```
1 Schema::create('artists', function (Blueprint $table) {  
2     $table->id();  
3     $table->string('name');  
4     $table->string('instrument');  
5     $table->timestamps();  
6 });
```

**artists**

id

name

instrument

timestamps

instrument contient l'instrument principal de l'artiste. Rien de neuf : c'est la migration du chapitre 10.

# La paire est la clé primaire de la table pivot

Dans le fichier créé par `make:migration create_artist_concert_table`, vous remplacez `id()` et `timestamps()`.

database/migrations/...\_create\_artist\_concert\_table.php

```
1 Schema::create('artist_concert', function (Blueprint $table) {  
2     $table->foreignId('artist_id')->constrained()->cascadeOnDelete();  
3     $table->foreignId('concert_id')->constrained()->cascadeOnDelete();  
4     $table->primary(['artist_id', 'concert_id']);  
5 });
```

1

Deux clés étrangères, comme `stage_id` au chapitre 11.

2

Une clé primaire composée : la même paire deux fois est refusée.

Pas d'`id()`, car une ligne se reconnaît à sa paire. Pas de `timestamps()`.

## artist\_concert: les deux modèles, au singulier, dans l'ordre alphabétique

Laravel attend ce nom pour une table pivot : les noms des deux modèles, en minuscules, séparés par `_`. Le `a` d'`artist` vient avant le `c` de `concert`.

terminal

```
$ php artisan db:table artist_concert
```

```
...
```

```
Index .....  
sqlite_autoindex_artist_concert_1 artist_id, concert_id .. compound, primary
```

```
Foreign Key ..... On Update / On Delete  
concert_id references id on concerts ..... no action / cascade  
artist_id references id on artists ..... no action / cascade
```

**compound, primary** : la clé primaire est faite des deux colonnes. Les deux clés étrangères suppriment en cascade.

## belongsToMany : un artiste est lié à plusieurs concerts

app/Models/Artist.php

```
1  #[Fillable(['name', 'instrument'])]
2  class Artist extends Model
3  {
4      public function concerts(): BelongsToMany
5      {
6          return $this->belongsToMany(Concert::class);
7      }
8  }
```

app/Models/Concert.php

```
1  public function guests(): BelongsToMany
2  {
3      return $this->belongsToMany(Artist::class);
4  }
```

Dans Concert, la méthode s'appelle `guests()`, et non `artists()`, parce qu'elle renvoie les invités du concert.

### 3 • Les deux modèles et belongsToMany

## Eloquent déduit trois noms des deux classes, jamais de la méthode

| Ce qu'Eloquent cherche              | Ce qu'il déduit             | De quoi                                          |
|-------------------------------------|-----------------------------|--------------------------------------------------|
| la table pivot                      | <code>artist_concert</code> | Artist et Concert, ordre alphabétique            |
| la colonne de la classe qui déclare | <code>artist_id</code>      | Artist en minuscules, suivi de <code>_id</code>  |
| la colonne de la classe visée       | <code>concert_id</code>     | Concert en minuscules, suivi de <code>_id</code> |

Au chapitre 11, `belongsToMany()` lisait le nom de la méthode `stage()` pour trouver `stage_id`. Ici, le nom `guests()` ne sert à aucune déduction.

## La relation envoie une jointure entre artists et artist\_concert

routes/web.php

```
1 $concert = Concert::find(2);  
2  
3 dump($concert->guests);  
4 dd($concert->guests()->toRawSql());
```

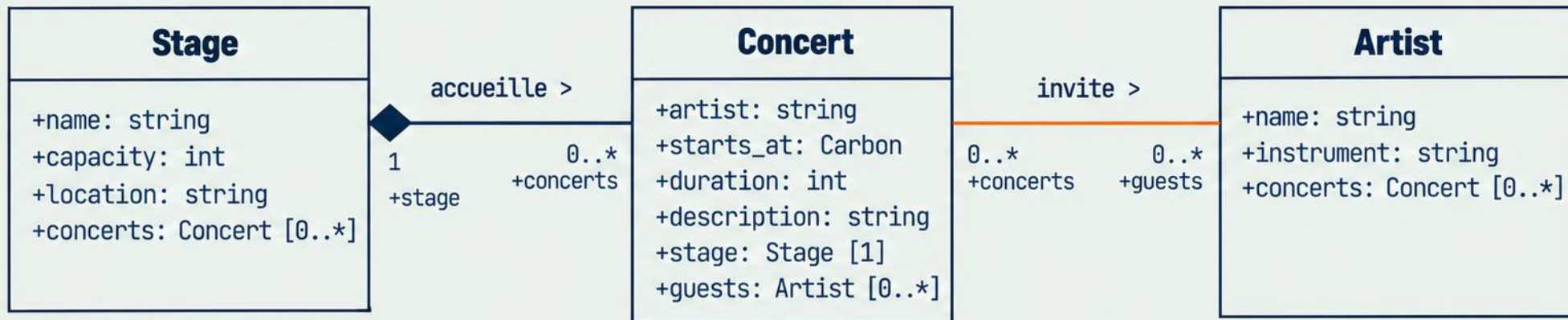
navigateur · /bac-a-sable

```
Illuminate\Database\Eloquent\Collection {#1102 ▼ // routes/web.php:188  
  #items: []  
  :::  
  "select * from "artists" inner join "artist_concert" on  
    "artists"."id" = "artist_concert"."artist_id" where "artist_concert"."concert_id" = 2" // routes/web.php:189
```

Une jointure, join en SQL, assemble les lignes de deux tables à partir d'une valeur commune.  
La collection est vide : la table pivot n'a encore aucune paire.

# Une association simple entre Concert et Artist, 0..\* des deux côtés

## diagramme de classes



Un artiste existe sans le concert : pas de losange. La table `artist_concert` n'apparaît pas, c'est le trait qui la représente.

# attach() insère une paire dans la table pivot

Vous enregistrez Ana Lopes et Karim Benali, puis vous les invitez.

routes/web.php

```
1 $ana = Artist::create(['name' => 'Ana Lopes', 'instrument' => 'chant']);
2 $karim = Artist::create(['name' => 'Karim Benali', 'instrument' => 'oud']);
3
4 Concert::find(1)->guests()->attach($ana->id);
5 Concert::find(2)->guests()->attach([$ana->id, $karim->id]);
```

table artist\_concert

| artist_id | concert_id |
|-----------|------------|
| 1         | 1          |
| 1         | 2          |
| 2         | 2          |

**guests()**, avec parenthèses, renvoie la relation. **attach()** reçoit un numéro d'artiste, seul ou dans un tableau, et insère une paire par numéro.

## \$concert->guests et \$artist->concerts lisent la même table pivot

routes/web.php

```
1 foreach (Concert::find(2)->guests as $guest) {  
2     dump($guest->name);  
3 }  
4  
5 $ana = Artist::where('name', 'Ana Lopes')->first();  
6  
7 foreach ($ana->concerts as $concert) {  
8     dump($concert->artist);  
9 }
```

navigateur · /bac-a-sable

```
"Ana Lopes" // routes/web.php:187  
"Karim Benali" // routes/web.php:187  
  
"Duo Mistral" // routes/web.php:193  
"L'Orchestre du Lac" // routes/web.php:193
```

**\$concert->guests** cherche dans `artist_concert` les paires du concert, et **\$ana->concerts** celles de l'artiste.

# La clé primaire composée refuse la même paire deux fois

Vous invitez une seconde fois Ana Lopes au concert 2.

routes/web.php

```
1 Concert::find(2)->guests()->attach($ana->id);
```

navigateur · /bac-a-sable

Illuminate\Database\UniqueConstraintViolationException

SQLSTATE[23000]: Integrity constraint violation: 19 UNIQUE constraint failed: artist\_concert.artist\_id, artist\_concert.concert\_id (...)

NON CONFORME

attach() envoie l'insertion sans vérifier que la paire existe. Sans cette clé, la fiche du concert afficherait Ana Lopes deux fois.

# detach() retire des paires, jamais l'artiste

routes/web.php

```
1 $concert = Concert::find(2);
2 $karim = Artist::where('name', 'Karim Benali')->first();
3
4 $concert->guests()->detach($karim->id);
5 dump($concert->guests()->count());
6
7 $concert->guests()->detach();
8 dd($concert->guests()->count());
```

1

Retire la paire de Karim Benali.

2

Sans argument : toutes les paires du concert.

navigateur · /bac-a-sable

```
1 // routes/web.php:190
0 // routes/web.php:193
```

Karim Benali existe toujours dans artists.

# sync() rend les paires identiques à la liste reçue

sync() compare la liste avec les paires du concert, ajoute celles qui manquent et retire celles qui n'y figurent pas.

routes/web.php

```
1 $concert = Concert::find(1);
2 $ana = Artist::where('name', 'Ana Lopes')->first();
3 $karim = Artist::where('name', 'Karim Benali')->first();
4
5 dd($concert->guests()->sync([$ana->id, $karim->id]));
```

navigateur • /bac-a-sable

```
array:3 [▼ // routes/web.php:190
  "attached" => array:1 [▼
    0 => 1
  ]
  "detached" => []
  "updated" => []
]
```

Karim Benali était déjà invité : sync() n'a ajouté qu'Ana Lopes, sans erreur.

**Écrivez attach() pour ajouter un invité, et sync() quand vous connaissez la liste complète des invités.**

## 4 · Relier : attach, detach et sync

# Le seeder invite les vrais artistes, puis deux artistes de fabrique par concert

database/seeder/ArtistSeeder.php

```
1 $ana = Artist::create(['name' => 'Ana Lopes', 'instrument' => 'chant']);
2 ...
3 Concert::where('artist', 'Duo Mistral')->first()->guests()->attach($ana->id);
4 ...
5 $artists = Artist::factory()->count(10)->create();
6
7 foreach (Concert::where('id', '>', 3)->get() as $concert) {
8     $concert->guests()->attach($artists->random(2));
9 }
```

1

Un opérateur de comparaison entre la colonne et la valeur : les concerts 4 à 23.

2

Deux artistes différents, pris au hasard : jamais la même paire deux fois.

Après `migrate:fresh --seed` : treize artistes et quarante-quatre paires.

# La liste du chapitre Views lit maintenant la relation

resources/views/concerts/show.blade.php

```
1 <h2>Invités</h2>
2 <ul>
3     @forelse ($concert->guests as $guest)
4         <li>{{ $guest->name }}, {{ $guest->instrument }}</li>
5     @empty
6         <li>Aucun invité annoncé</li>
7     @endforelse
8 </ul>
```

terminal

```
$ curl -s http://premier-test-laravel.test/concerts/2
...
<h2>Invités</h2>
<ul>
    <li>Ana Lopes, chant</li>
    <li>Karim Benali, oud</li>
</ul>
```

Le HTML est celui du chapitre Views. Chaque nom devient ensuite un lien vers `route('artists.show', $guest->id)`.

# La page `/artistes/{id}` liste les concerts d'un artiste

Le contrôleur suit le modèle de StageController, au chapitre 11.

app/Http/Controllers/ArtistController.php

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

o o o /artistes/1

routes/web.php

```
1 Route::get('/artistes/{id}', [ArtistController::class, 'show'])->whereNumber('id')->name('artists.show');
```

`$artist->concerts()` traverse la table pivot, et `orderBy('starts_at')` trie les concerts.  
`/artistes/99` répond 404.

# DB::getQueryLog() renvoie chaque requête envoyée à la base

DB::enableQueryLog() demande à Laravel de garder en mémoire chaque requête envoyée ensuite.

routes/web.php

```
1 DB::enableQueryLog();
2
3 foreach (Concert::all() as $concert) {
4     $stageName = $concert->stage->name;
5 }
6
7 dd(count(DB::getQueryLog()));
```

navigateur · /bac-a-sable

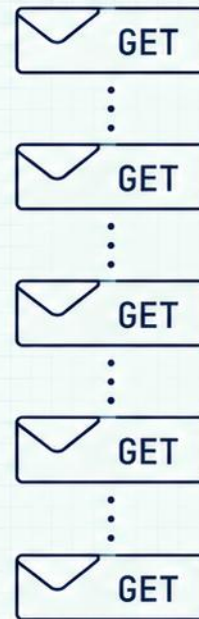
24 // routes/web.php:196

Vingt-trois concerts, vingt-quatre requêtes. C'est la boucle du composant concert-card, à chaque visite de /concerts.

# Une requête pour la liste, puis une requête de plus par élément

navigateur · /bac-a-sable

```
array:24 [▼ // routes/web.php:196
  0 => array:4 [▼
    "query" => "select * from "concerts""
    ...
  1 => array:4 [▼
    "query" => "select * from "stages" where "stages"."id" = ? limit 1"
    "bindings" => array:1 [▼
      0 => 1
    ]
    ...
  2 => array:4 [▼
    "query" => "select * from "stages" where "stages"."id" = ? limit 1"
    ...
  ...
```



Chaque lecture de `$concert->stage` envoie sa propre requête, et le `?` reçoit le `stage_id` du concert. C'est le **piège N+1** : avec cinq cents concerts, cinq cent une requêtes.

# with('stage') charge les scènes en une seule requête

routes/web.php

```
1  foreach (Concert::with('stage')->get() as $concert) {  
2      $stageName = $concert->stage->name;  
3  }
```

navigateur • /bac-a-sable

```
array:2 [▼ // routes/web.php:196  
  0 => array:4 [▼  
    "query" => "select * from "concerts""  
    ...  
  1 => array:4 [▼  
    "query" => "select * from "stages" where "stages"."id" in (1, 2)""  
    ...
```

Eloquent lit les concerts, relève leurs stage\_id, puis lit les deux scènes avec in (1, 2). Dans la boucle, `$concert->stage` ne demande plus rien à la base.

# with() accepte un tableau de relations

routes/web.php

```
1  foreach (Concert::with(['stage', 'guests'])->get() as $concert) {  
2      $stageName = $concert->stage->name;  
3  
4      foreach ($concert->guests as $guest) {  
5          $guestName = $guest->name;  
6      }  
7  }  
8  
9  dump(count(DB::getQueryLog()));
```

navigateur · /bac-a-sable

3 // routes/web.php:200

Trois requêtes, là où la même boucle sans with() en envoie quarante-sept.

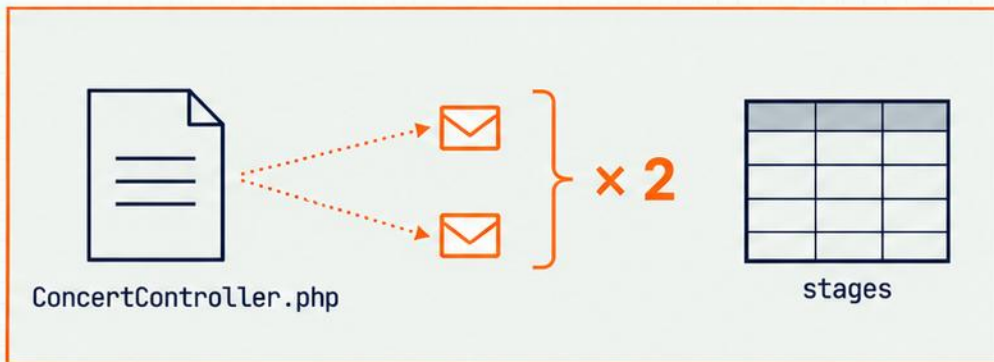
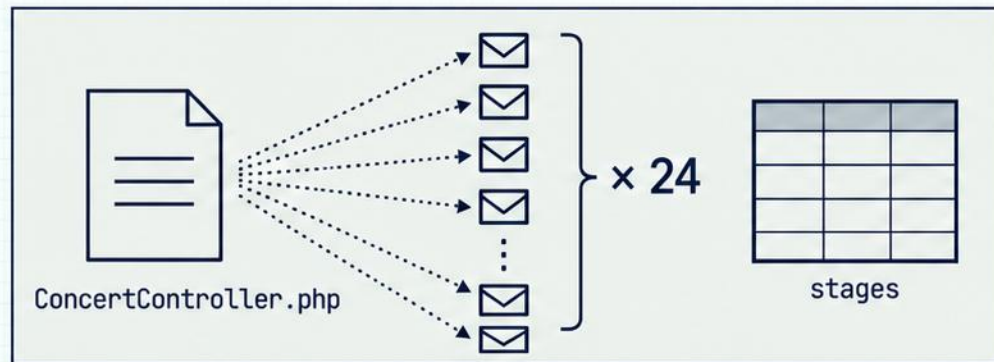
**Quand une boucle affiche une relation, chargez cette relation avec with().**

# La page /concerts passe de vingt-quatre requêtes à deux

app/Http/Controllers/ConcertController.php

```
1 $concerts = Concert::with('stage')->orderBy('starts_at')->get();
```

La page affiche le même HTML. La méthode `show()` ne change pas : la fiche d'un concert envoie trois requêtes, pour le concert, sa scène et ses invités, avec ou sans `with()`.



# Le nom d'un artiste n'est écrit qu'une fois

Ana Lopes se produit désormais sous le nom « Ana Lopes Ferreira ».

routes/web.php

```
1 Artist::find(1)->update(['name' => 'Ana Lopes Ferreira']);  
2  
3 return redirect()->route('concerts.show', 2);
```

navigateur · /concerts/2

```
<h2>Invités</h2>  
<ul>  
  <li><a href="http://premier-test-laravel.test/artistes/1">Ana Lopes Ferreira</a>, chant</li>  
  <li><a href="http://premier-test-laravel.test/artistes/2">Karim Benali</a>, oud</li>  
</ul>
```

Une seule ligne a changé, et la fiche du concert 1 affiche aussi le nouveau nom. C'est la réponse au rituel de la slide 2.

# Supprimer un artiste supprime ses paires, jamais ses concerts

La table pivot n'a pas de modèle : `DB::table('artist_concert')` l'interroge directement.

| routes/web.php |                                                             | navigateur · /bac-a-sable |                       |
|----------------|-------------------------------------------------------------|---------------------------|-----------------------|
| 1              | <code>dump(DB::table('artist_concert')-&gt;count());</code> |                           |                       |
| 2              |                                                             |                           |                       |
| 3              | <code>Artist::find(1)-&gt;delete();</code>                  | 44                        | // routes/web.php:191 |
| 4              |                                                             | 42                        | // routes/web.php:195 |
| 5              | <code>dump(DB::table('artist_concert')-&gt;count());</code> |                           |                       |
| 6              | <code>dd(Concert::count());</code>                          | 23                        | // routes/web.php:196 |

Ana Lopes était invitée à deux concerts : la base a supprimé ses deux paires, et les vingt-trois concerts restent.

## Une suppression se propage de clé étrangère en clé étrangère

Vous supprimez la Grande scène, comme au chapitre 11.

navigateur · /bac-a-sable

```
22 // routes/web.php:191
41 // routes/web.php:192
11 // routes/web.php:196
20 // routes/web.php:197
```

| Vous supprimez | La base supprime aussi          |
|----------------|---------------------------------|
| un artiste     | ses paires dans artist_concert  |
| un concert     | ses paires dans artist_concert  |
| une scène      | ses concerts, puis leurs paires |

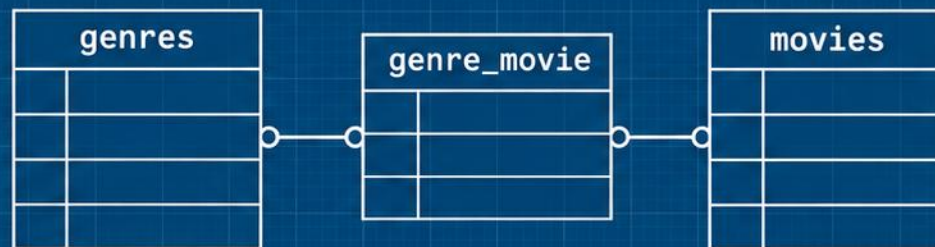
stage\_id fait supprimer les onze concerts de la scène, puis concert\_id fait supprimer leurs paires. Aucun artiste n'est supprimé.

# À vous : les genres du cinéma, dans eloquent-26

85 minutes, en séance ou chez vous, et 20 minutes de plus pour le bonus.

Les films reçoivent des genres : une table genres et une table pivot genre\_movie.

- Katas 12.1 et 12.2 : la table pivot, belongsToMany.
- Katas 12.3 et 12.4 : le seeder, la page genre.
- Kata 12.5 : les genres dans la liste, avec with().
- Bonus 12.6 : les salles. Exercice 12.7 : sur papier.



```
php artisan test --group=chapitre12
```