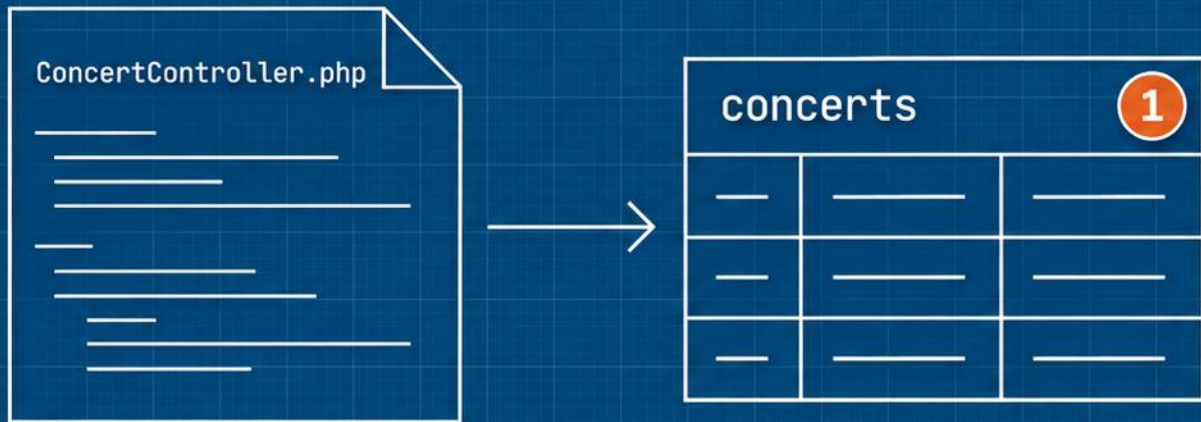


Bloc Laravel · chapitre 10

Modèles : une table

Les concerts quittent le tableau PHP du contrôleur. Ils s'installent dans une table de la base, lue par la classe Concert.



Les concerts sont écrits dans le code du contrôleur

Au chapitre Views, les trois concerts sont une propriété de ConcertController.

app/Http/Controllers/ConcertController.php

```
1  private array $concerts = [  
2      1 => [  
3          'artist' => 'Duo Mistral',  
4          'stage' => 'Chapiteau',  
5          'time' => '18 h',  
6          ...  
7      ],  
8      ...  
9  ];
```

1 Une autre page ne peut pas lire cette propriété.

2 Huit lignes par concert : 240 lignes pour trente concerts.

Les données sont mélangées au code : pour changer une heure, il faut modifier le programme.

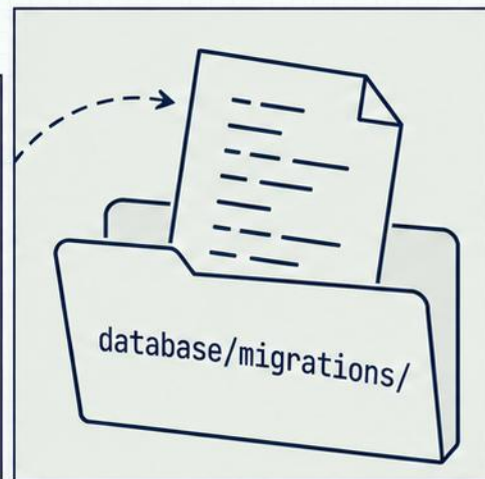
make:migration crée le fichier qui décrit la table

Une migration est un fichier PHP qui crée ou modifie une table. Artisan reconnaît la forme `create_..._table` et prépare la création de la table concerts.

terminal

```
$ php artisan make:migration create_concerts_table
```

```
INFO Migration [database/migrations/2026_09_28_213249_create_concerts_table.php]  
created successfully.
```



Le nom du fichier commence par la date et l'heure de la commande. Laravel exécute les migrations dans l'ordre de leurs noms.

up() crée la table, down() la supprime

Voici les deux méthodes du fichier créé par Artisan.

database/migrations/2026_09_28_213249_create_concerts_table.php

```
1  public function up(): void
2  {
3      Schema::create('concerts', function (Blueprint $table) {
4          $table->id();
5          $table->timestamps();
6      });
7  }
8  ...
9  public function down(): void
10 {
11     Schema::dropIfExists('concerts');
12 }
```

1

La clé primaire.

2

Les colonnes created_at et updated_at.

3

L'inverse de up().

Chaque appel sur \$table ajoute une colonne

Entre `id()` et `timestamps()`, vous ajoutez les colonnes d'un concert.

database/migrations/..._create_concerts_table.php

```
1 Schema::create('concerts', function (Blueprint $table) {  
2     $table->id();  
3     $table->string('artist');  
4     $table->string('stage');  
5     $table->dateTime('starts_at');  
6     $table->unsignedSmallInteger('duration');  
7     $table->timestamps();  
8 });
```

1

La méthode donne le type, l'argument donne le nom.

2

2026-10-16 18:00:00, que la base sait trier.

3

Des minutes.

Six types de colonnes suffisent pour ce cours

Méthode	La colonne contient	Exemple
<code>string('artist')</code>	un texte court, 255 caractères au plus	un nom
<code>text('description')</code>	un texte long	une description
<code>unsignedSmallInteger('duration')</code>	un entier de 0 à 65 535	des minutes
<code>date('released_on')</code>	une date sans heure	une sortie de film
<code>dateTime('starts_at')</code>	une date et une heure	un début de concert
<code>decimal('price', 5, 2)</code>	un nombre exact, 5 chiffres dont 2 décimales	un prix

Une colonne refuse une case vide, null en PHP. Pour l'accepter, ajoutez `->nullable()` après le type.

migrate applique les migrations pas encore appliquées

migrate exécute la méthode `up()` de chaque migration nouvelle, puis inscrit son nom dans la table **migrations** de la base.

terminal

```
$ php artisan migrate
```

```
INFO Running migrations.
```

```
2026_09_28_213249_create_concerts_table ..... 7.72ms DONE
```

```
$ php artisan migrate
```

```
INFO Nothing to migrate.
```

migrations



create_concerts_table

La seconde fois, la migration est déjà inscrite : **migrate** ne la rejoue pas.

migrate:status affiche l'état de chaque migration

terminal

```
$ php artisan migrate:status
```

```
Migration name ..... Batch / Status
0001_01_01_000000_create_users_table ..... [1] Ran
...
2026_09_28_211118_create_posts_table ..... [2] Ran
2026_09_28_213249_create_concerts_table ..... [3] Ran
```



Le numéro entre crochets est le **lot** : les migrations lancées par la même commande forment un lot.

Ran signifie que la migration est appliquée.

db:table décrit les colonnes d'une table

Pour vérifier ce que la migration a créé, demandez la description de la table.

terminal

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

```
...
```

Column	Type
id integer, autoincrement	integer
artist varchar	varchar
stage varchar	varchar
starts_at datetime	datetime
duration integer	integer
created_at datetime, nullable	datetime
updated_at datetime, nullable	datetime

SQLite nomme varchar un texte court et range unsignedSmallInteger parmi les integer.

migrate:rollback annule le dernier lot

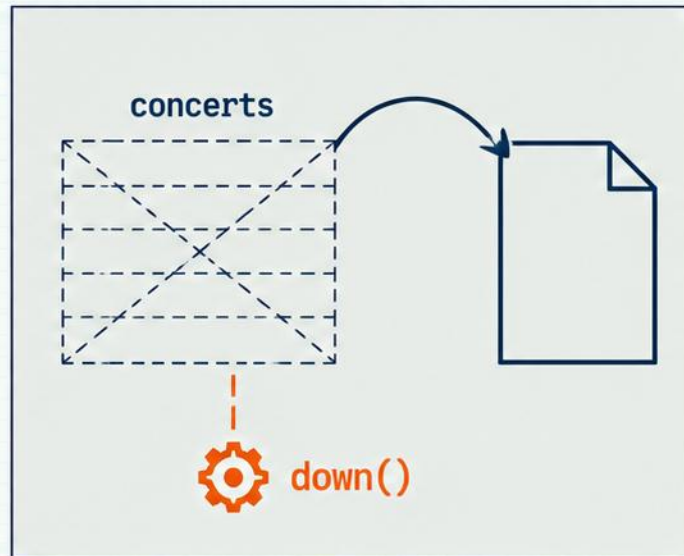
Il manque la colonne description. Or migrate ne rejoue pas une migration déjà inscrite : annulez-la d'abord.

terminal

```
$ php artisan migrate:rollback
```

INFO Rolling back migrations.

```
2026_09_28_213249_create_concerts_table ..... 10.37ms DONE
```



Laravel exécute `down()`, qui supprime la table `concerts`. Vous ajoutez `$table->text('description');` sous `duration`, puis vous relancez `migrate`.

Chaque ligne de `up()` devient une colonne de la table

database/migrations/..._create_concerts_table.php

```
1 Schema::create('concerts', function (Blueprint $table) {  
2     $table->id();  
3     $table->string('artist');  
4     $table->string('stage');  
5     $table->dateTime('starts_at');  
6     $table->unsignedSmallInteger('duration');  
7     $table->text('description');  
8     $table->timestamps();  
9 });
```

concerts							
id	artist	stage	starts_at	duration	description	created_at	updated_at

Une migration appliquée par d'autres ne se modifie plus : écrivez une nouvelle migration qui change la table.

Un modèle est une classe liée à une table

La classe interroge la table, et chaque objet de la classe représente une ligne.

terminal

```
$ php artisan make:model Concert
```

```
INFO Model [app/Models/Concert.php] created successfully.
```

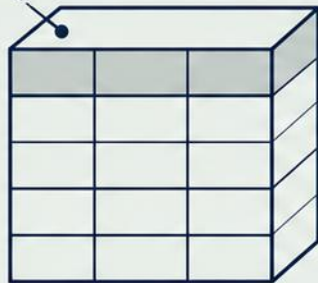
app/Models/Concert.php

```
1 class Concert extends Model
2 {
3     //
4 }
```

Concert



concerts



La classe est vide. Par héritage, elle reçoit de `Model` les méthodes qui lisent et écrivent dans la base.

Eloquent déduit le nom de la table du nom de la classe

Eloquent est la partie de Laravel qui relie les classes aux tables. Il met le nom de la classe au pluriel anglais, en minuscules, avec un `_` entre les mots.

Classe	Table
Concert	concerts
ParkingRate	parking_rates
Person	people
Cheval	chevals

La table chevaux ne serait jamais trouvée. C'est pourquoi tout le code du cours porte des noms anglais.

create() enregistre une ligne à partir d'un tableau

Une **route bac à sable** est une route réservée à vos essais. Elle remplace une page tant que vous testez le modèle.

routes/web.php

```
1 Route::get('/bac-a-sable', function () {  
2     Concert::create([  
3         'artist' => 'Duo Mistral',  
4         'stage' => 'Chapiteau',  
5         'starts_at' => '2026-10-16 18:00:00',  
6         'duration' => 45,  
7         'description' => 'Guitare et violon, <strong>entrée libre</strong>.',  
8     ]);  
9  
10    return 'Concert enregistré';  
11 });
```

1

Chaque clé porte le nom d'une colonne.

Sans #[Fillable], create() est refusé

```
navigateur · /bac-a-sable
```

```
Illuminate\Database\Eloquent\MassAssignmentException
```

```
Add [artist] to fillable property to allow mass assignment on [App\Models\Concert].
```

NON CONFORME

Remplir plusieurs colonnes avec un tableau s'appelle une **affectation de masse**. Le modèle doit donner la liste des colonnes qu'un tableau peut remplir.

```
app/Models/Concert.php
```

```
1  #[Fillable](['artist', 'stage', 'starts_at', 'duration', 'description'])
```

```
2  class Concert extends Model
```

Avec la liste, un visiteur ne peut pas ajouter un champ `is_admin` à un formulaire.

Un seeder ajoute les données de départ

Un **seeder** est une classe qui ajoute des données de départ dans la base. Sa méthode `run()` crée les trois concerts du festival.

terminal

```
$ php artisan make:seeder ConcertSeeder
```

database/seeder/ConcertSeeder.php

```
1 public function run(): void
2 {
3     Concert::create([
4         'artist' => 'Duo Mistral',
5         ...
6     ]);
7     ...
8 }
```

DatabaseSeeder



ConcertSeeder



concerts



Dans `DatabaseSeeder`, la ligne `$this->call(ConcertSeeder::class);` appelle votre seeder.

4 • Le seeder

db:seed ajoute, migrate:fresh --seed remet à neuf

Un seeder ajoute des lignes à celles qui existent. Relancé, db:seed s'arrête sur l'utilisateur de test, crée une seconde fois :

terminal

```
$ php artisan db:seed
```

```
...
```

```
SQLSTATE[23000]: Integrity constraint violation: 19 UNIQUE constraint failed: users.email (Connection: sqlite, ...)
```

NON CONFORME

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

```
Dropping all tables ..... 4.77ms DONE
```

```
...
```

migrate:fresh --seed supprime toutes les tables, rejoue les migrations, puis lance les seeders. Ne lancez jamais cette commande sur une base dont vous gardez les données.

dd() affiche une variable et arrête le programme

dump() affiche le contenu d'une variable, puis le code continue. **dd()** affiche le contenu, puis arrête le programme.

routes/web.php

```
1 Route::get('/bac-a-sable', function () {  
2     dd(Concert::all());  
3 });
```

navigateur · /bac-a-sable

```
Illuminate\Database\Eloquent\Collection {#1522 ▼ // routes/web.php:182  
  #items: array:3 [▶]  
  #escapeWhenCastingToString: false  
}
```

Concert::all() renvoie une **collection** de trois éléments. La première ligne donne aussi le fichier et la ligne du **dd()**.

Chaque objet Concert représente une ligne de la table

Dépliez #items, puis le premier élément, puis #attributes.

navigateur · /bac-a-sable

```
0 => App\Models\Concert {#1534 ▼  
  ...  
  #attributes: array:8 [▼  
    "id" => 1  
    "artist" => "Duo Mistral"  
    "stage" => "Chapiteau"  
    "starts_at" => "2026-10-16 18:00:00"  
    "duration" => 45  
    ...  
    "created_at" => "2026-09-28 21:35:32"  
    "updated_at" => "2026-09-28 21:35:32"  
  ]
```



concerts						
1	Duo Mistral	Chapiteau	2026-10-16 18:00:00	45	2026-09-28 21:35:32	2026-09-28 21:35:32

#attributes contient une valeur par colonne. Vous lisez ces valeurs avec une flèche : `$concert->artist`.

find() renvoie un objet ou null, findOrFail() répond 404

routes/web.php		navigateur · /bac-a-sable	
1	<code>Route::get('/bac-a-sable', function () {</code>		
2	<code> dump(Concert::count());</code>	3	<code>// routes/web.php:182</code>
3	<code> dump(Concert::find(2)->artist);</code>		<code>"L'Orchestre du Lac" // routes/web.php:183</code>
4	<code> dd(Concert::find(99));</code>		<code>null // routes/web.php:184</code>
5	<code>});</code>		

Aucun concert ne porte l'id 99. Dans une vue, `$concert->artist` sur `null` donnerait une erreur 500. `findOrFail(99)` lance une exception, que Laravel transforme en réponse 404.

where() filtre, orderBy() trie, get() envoie la requête

where() et orderBy() préparent la requête sans l'envoyer à la base. get(), first() ou count() l'envoient.

routes/web.php

```
1  foreach (Concert::where('stage', 'Grande scène')->get() as $concert) {  
2      dump($concert->artist);  
3  }  
4  
5  dd(Concert::where('stage', 'Grande scène')->orderBy('starts_at')->toRawSql());
```

navigateur · /bac-a-sable

```
"L'Orchestre du Lac" // routes/web.php:183  
"Orage Mécanique" // routes/web.php:183  
"select * from "concerts" where "stage" = 'Grande scène' order by "starts_at" asc" // routes/web.php:187
```

toRawSql() montre le SQL que la requête enverrait.

Collection, objet, null ou nombre : ce que chaque méthode renvoie

Vous écrivez	Vous recevez
<code>Concert::all(), ...->get()</code>	une collection d'objets Concert, peut-être vide
<code>Concert::find(2), ...->first()</code>	un objet Concert, ou null
<code>Concert::findOrFail(2)</code>	un objet Concert, ou une réponse 404
<code>Concert::count(), ...->count()</code>	un entier



Écrivez `findOrFail()` dans un contrôleur quand l'id vient de l'adresse, `find()` quand null vous convient, et `where()` pour chercher selon une autre colonne.

Le contrôleur lit les concerts dans la base

La propriété \$concerts disparaît. Les routes et les adresses ne changent pas.

app/Http/Controllers/ConcertController.php

```
1 public function index(): View
2 {
3     $concerts = Concert::orderBy('starts_at')->get();
4
5     return view('concerts.index', ['concerts' => $concerts]);
6 }
7
8 public function show(int $id): View
9 {
10    $concert = Concert::findOrCreate($id);
11
12    return view('concerts.show', ['concert' => $concert]);
13 }
```

1

Les concerts, triés par heure.

2

Remplace isset() et abort(404).

Dans les vues, `$concert['artist']` devient `$concert->artist`

Un concert était un tableau. C'est maintenant un objet : les trois vues lisent ses colonnes avec une flèche.

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

1  @props(['concert'])
2
3  <article class="concert-card">
4      <h2>{{ $concert->artist }}</h2>
5      <p>{{ $concert->stage }}, {{ $concert->starts_at->format('G \h') }}</p>
6      ...
7  </article>
```

1

Remplace l'ancienne
clé time. Affiche 18 h.

Dans `concerts/index.blade.php`, le lien utilise la colonne `id` : `route('concerts.show', $concert->id)`.

casts() transforme starts_at en objet date

La base envoie starts_at sous forme de texte, et une chaîne n'a pas de méthode format():

```
navigateur · /concerts
```

```
Call to a member function format() on string (View: .../resources/views/components/concert-card.blade.php)
```

Une **conversion**, *cast* en anglais, transforme la valeur d'une colonne à chaque lecture.

```
app/Models/Concert.php
```

```
1  protected function casts(): array
2  {
3      return [
4          'starts_at' => 'datetime',
5      ];
6  }
```

\$concert->starts_at renvoie maintenant un objet Carbon, et la page affiche Chapiteau, 18 h.

save() et update() modifient une ligne, delete() la supprime

Pour modifier un concert, lisez-le, changez une propriété, puis appelez save().

routes/web.php

```
1 $concert = Concert::find(1);
2 $concert->duration = 60;
3 $concert->save();
4
5 $concert->update(['stage' => 'Kiosque', 'duration' => 30]);
6
7 Concert::find(3)->delete();
8
9 dd(Concert::count());
```

1

Envoie un UPDATE et met updated_at à l'heure.

2

Passer par #[Fillable], comme create().

3

Affiche 2.

Ensuite, `php artisan migrate:fresh --seed` remet les trois concerts.

À vous : le cinéma, dans le dépôt eloquent-26

85 minutes, en séance ou chez vous. Le dépôt ne contient que des tests : vous créez tout avec Artisan.

- **Katas 10.1 à 10.3** : la table `movies`, le modèle `Movie`, le seeder.
- **Katas 10.4 à 10.6** : la liste `/films`, la fiche, le filtre par année.
- **Exercice 10.7** : prédire cinq requêtes, sur papier.

ooo

```
php artisan test --group=chapitre10
```

movies			
—	—	—	—
—	—	—	—
—	—	—	—
—	—	—	—

