Eloquent avancé

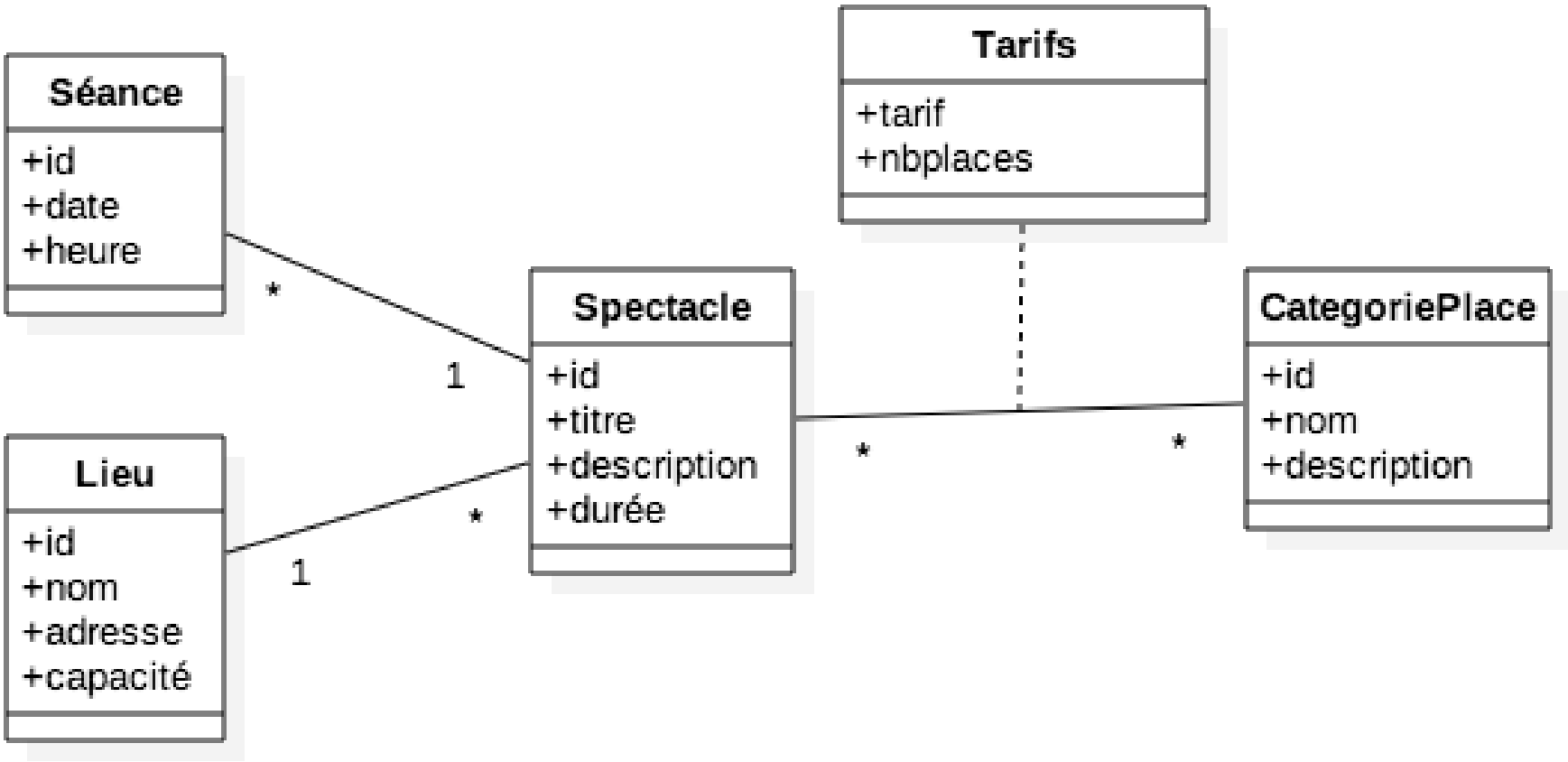
- associations n-n
- requêtes sur les associations
- soft deletes
- clés primaires non auto-increment

la gestion des associations avec Eloquent

- **Gérer les associations entre objets**
  - récupérer dans la bd les objets associés à un objet
  - insérer/modifier/supprimer les associations entre objets dans la bd

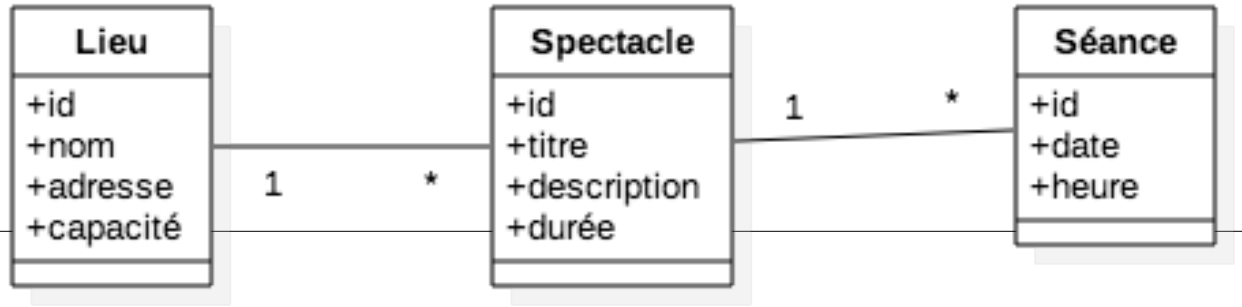
- **Eloquent** permet de définir des associations entre **modèles** pour les gérer comme des méthodes/propriétés permettant de parcourir des liens entre objets.

exemple




```
spectacle(id,nom, descr, durée, lieu_id)
lieu(id, nom, adresse, capacité)
seance(id, date, heure, spect_id)
categorie_place(id, nom, descr)
tarif(spect_id, cat_id, tarif, nbplaces)
```

associations 1-n (rappel)




```php
namespace tikenet;

class Spectacle extends \Illuminate\Database\Eloquent\Model
{
    protected $table='spectacle';
    protected $primaryKey='id'

   public function lieu() {
       return $this->belongsTo('tikenet\Lieu',
                               'lieu_id');
   }
   public function seances() {
       return $this->hasMany('tikenet\Seance',
                             'spect_id')
   }
}
```

utilisation

```php
$s= Spectacle::find(73);

$seances = $s->seances()->get();
$lieu = $s->lieu()->get();

/* raccourci */
$seances = $s->seances ;
$lieu = $s->lieu ;

/* avec des conditions */
$seances = $s->seances()
              ->where('date', '=', '2017-12-24')
              ->get();

/* chargement lié */
$sp = Spectacle::where('titre', 'like', '%stones%')
                    ->with('seances', 'lieu')
                    ->get();
```

création d'associations et d'objets associés

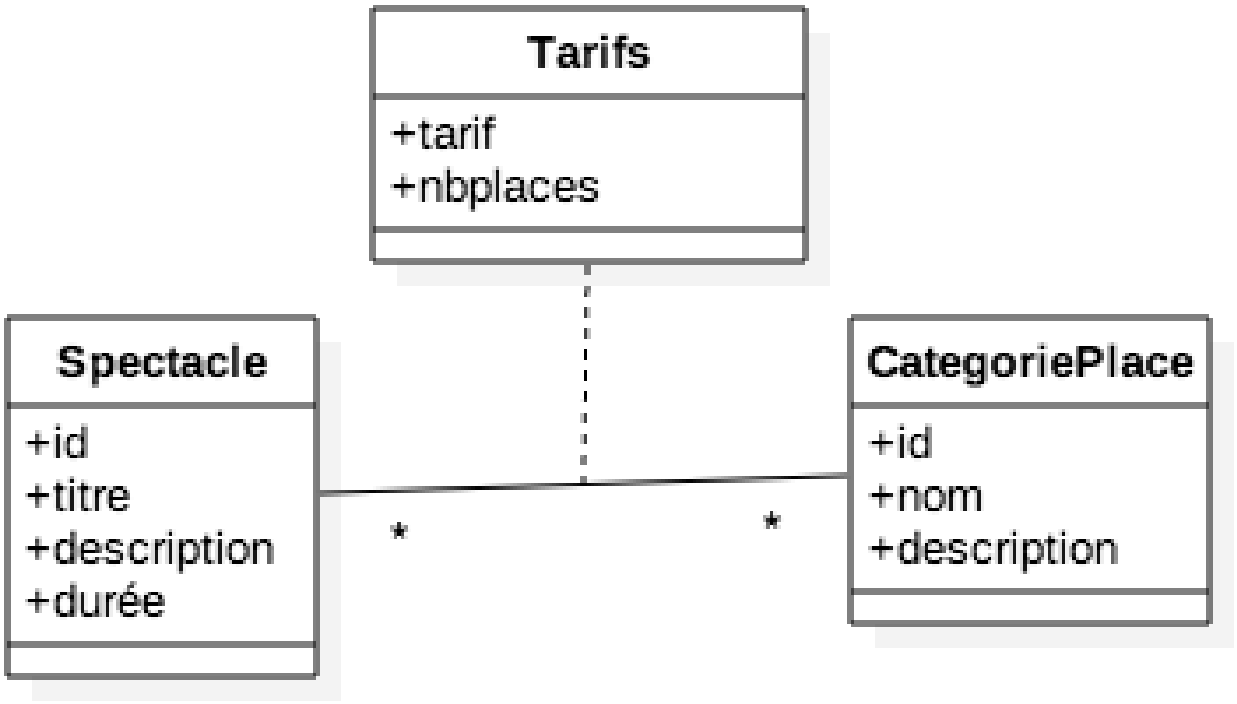
```php
/**  créer une association belongsTo entre 2 objets
 *   existants
 **/

$spect = Spectacle::find(73);
$lieu = Lieu::find(12);

$spect->lieu()->associate($lieu);
```

```php
/* créer un objet associé  */

$seance = new Seance();
$seance->date = '2017-12-24';
$seance->heure= '14:00';

$spect->seances()->save($seance);
```

associations n-n

- utilisation d'une table **pivot** contenant 2 clés étrangères
- méthode **belongsToMany()** utilisée de chaque côté
- il faut préciser :
  - le nom de la table pivot
  - le nom des clés étrangères

exemple




```
spectacle(id,nom, descr, durée, lieu_id)
categorie_place(id, nom, descr)
tarif(spect_id, cat_id, tarif, nbplaces)
```

réalisation

```php
namespace tikenet;

class Spectacle extends \Illuminate\Database\Eloquent\Model{
    protected $table='spectacle';
    protected $primaryKey='id'

   public function lieu() { … }
   public function seances() { … }

   public function categories() {

     return
      $this->belongsToMany('tikenet\CategoriePlace',
                           'tarifs',
                           'spect_id',
                           'cat_id');
    }
}
```

modèle cible

table pivot

FK vers source

FK vers cible

association inverse

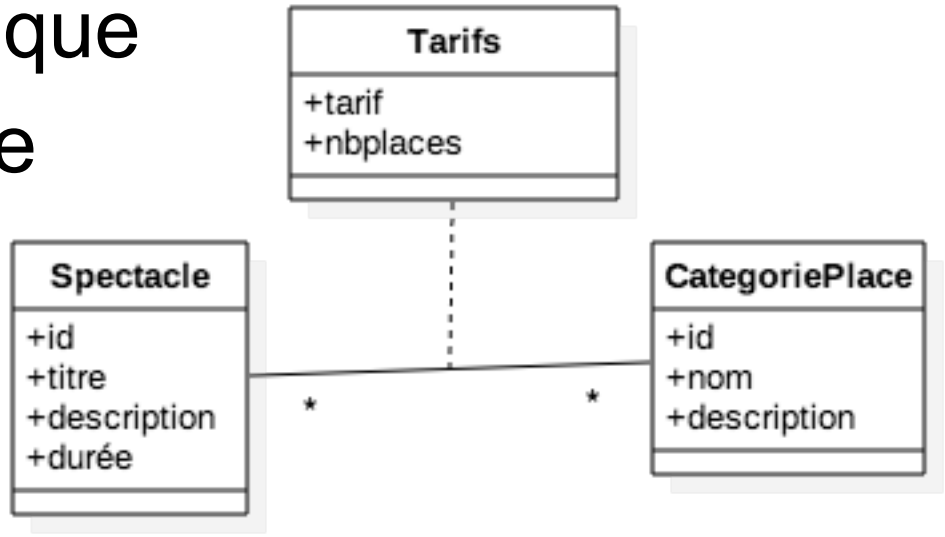
```
class CategoriePlace extends
                \Illuminate\Database\Eloquent\Model
{

  public function spectacles() {

    return $this->belongsToMany('tikenet\Spectacle',
                                'tarifs',
                                'cat_id',
                                'spect_id');
    }
}
```

utilisation : comme les associations 1-n

```php
$s= Spectacle::find(73) ;

/* catégories du spectacle 42 */
$cat = $s->categories()->get();

/* raccourci */
$cat = $s->categories;

/* condition */
$cat = $s->categories()
                ->where('nom', 'like', 'balcon')
                ->get();

/* chargement lié */
$sps = Spectacle::where('titre', 'like', '%stones%')
                 ->with('seances', 'lieu', 'categories')
                 ->get();
```

oui, mais, c'est pas fini

- comment manipuler les attributs de l'association ?
  - tarif et nbplaces de chaque catégorie d'un spectacle



1) Déclarer les attributs de l'association :

```
public function categories() {

     return
      $this->belongsToMany('tikenet\CategoriePlace',
                           'tarifs',
                           'spect_id',
                           'cat_id')
        ->withPivot( ['tarif', 'nbplaces' ] );
    }
```

2) Accéder aux attributs :

```
$s = Spectacle::with('categories')
     ->where('id','=',73)->first();

foreach ($s->categories as $cat) {

     print $cat->pivot->tarif;
     print $cat->pivot->nbplaces;
}
```

- pour personnaliser la **dénomination du pivot** :

```php
public function categories() {

    return
     $this->belongsToMany('tikenet\CategoriePlace',
                          'tarifs',
                          'spect_id',
                          'cat_id')
       ->withPivot( ['tarif', 'nbplaces' ] )
       ->as( 'tarification' );
   }
```

```php
$s = Spectacle::with('categories')
     ->where('id','=',73)->first();

foreach ($s->categories as $cat) {

    print $cat->tarification->tarif;
    print $cat->tarification->nbplaces;
}
```

créer des objets associés

```php
/* créer un objet associé  */

$cat = new CategoriePlace();
$cat->nom = 'gradins debout';
$cat->descrition= 'place debout en haut des gradins';

$spect->categories()->save($cat,
                           ['tarif'=>28.0,
                            'nbplaces'=>250]
);
```

- création d'une catégorie associée à un spectacle
- le tableau d'attributs est optionnel

créer/supprimer des associations entre objets existants

- création/suppression d'associations :

```php
/* associer le spectacle 42 aux catégories 1,3,5,7 */

$spect = Spectacle::find(42);
$spect->categories()->attach( [1,3,5,7] );

/* dissocier le spectacle 42 des catégories 1,7 */

$spect->categories()->detach( [1,7] );


/* modification globale de l'association  */

$spect->categories()->sync( [5,8] );
/* 3 : retirée
 * 5 : conservée
 * 8 : ajoutée
 */
```

gérer les attributs d'association

- les méthodes `attach()` et `sync()` permettent d'indiquer les valeurs des attributs portés par l'association :

```
$spect = Spectacle::find(42);
$spect->categories()->attach( [
      1=>[ 'tarif'=>28.0, 'nbplaces'=>250],
      3=>[ 'tarif'=>35.0, 'nbplaces'=>150] ]
);

$spect->categories()->sync( [
      3,
      8=>[ 'tarif'=>55.0, 'nbplaces'=>175]]
);
```

Requêtes sur les associations

1) requêtes portant sur des **objets liés**, avec des conditions sur les objets liés
   - pour le spectacle 22, liste des séances à 20h
   - => on cherche des **séances**
2) requêtes sur des **objets sources** avec une condition portant sur des objets liés
   - les spectacles ayant une séance à 20h
   - => on cherche des **spectacles**

1er cas : objets cibles avec condition

```php
$s= Spectacle::find(73);

$seances = $s->seances()
              ->where('heure', '=', 20)->get();

$cat = $s->categories()
              ->where('nom', 'like', 'gradin')
              ->get();


/* condition sur un attribut de l'association */

$cat = $s->categories()
              ->where('nom', 'like', 'gradin')
              ->wherePivot('tarif', '<', 20)
              ->get();
```

2ème cas : objets sources avec des condition sur les objets associés

```php
/* spectacles ayant un lieu défini :
 * test l'existence de l'association
 */
$sps = Spectacle::has('lieu')->get();

/* spectacles avec plus de 3 séances : */
$sps = Spectacle::has('seances', '>', 3)->get();
;
/* spectacles ayant des séances à 20h */
$sps = Spectacle::whereHas('seances', function($q) {
    $q->where('heure', '=', 20);
})->get();

/* condition sur les attributs d'une association */
$sps = Spectacle::whereHas('categories',
  function($q) {
      $q->where('tarification.tarif', '<', 30);
})->get();
```

- exemple complet : spectacle dont le titre contient "stones" ayant des séances à 20h et des places gradins à moins de 50€ à Nancy

```
Spectacle::where('titre', 'like', '%stones%')
    ->whereHas('seances', function($q) {
        $q->where('heure', '=', 20);
     })
    ->whereHas('lieu', function($q) {
        $q->where('adresse', 'like', '%nancy%');
     })
    ->whereHas('categories', function($q) {
        $q->where('nom', 'like', '%gradin%')
          ->where('tarification.tarif', '<=', 50);
     })
    ->orderBy('titre')
    ->get();
```

suppressions logiques

- Eloquent permet de gérer des suppressions logiques (soft delete) :
  - données marquées comme supprimées
  - n'apparaissent plus dans les résultats des requêtes
  - mais conservées dans la base

  - **technique** : ajouter une colonne "`deleted_at`" dans la table concernée
  - toutes les lignes avec 1 valeur non nulle pour cette colonne sont éliminées des résultats
  - la méthode `delete()` gère la colonne `deleted_at`

exemple : profils d'utilisateurs, on ne les supprime jamais

```php
namespace tikenet;

use Illuminate\Database\Eloquent\Model ;
use Illuminate\Database\Eloquent\SoftDeletes;

class Profile extends Model {

    use SoftDeletes;

    protected $table = 'user_profile';
    protected $primaryKey = 'id';

    protected $dates = ['deleted_at'];
}
```

```php
$profile = \tikenet\Profile::find(73);
$profile->delete();
```

- On peut malgré tout faire des requêtes
  - incluant les lignes supprimées logiquement
  - portant sur les lignes supprimées logiquement

```
$profiles = \tikenet\Profile::withTrashed()
                ->where('city', 'like', 'nancy')
                ->get();


$profiles = \tikenet\Profile::onlyTrashed()
                ->where('city', 'like', 'paris')
                ->get();
```

clés primaires non auto-increment

- Dans certains cas, il est utile de pouvoir déclarer une clé primaire dont la valeur est gérée par l'application et non par la base de données
- La valeur de la clé doit être fournie avant l'insertion dans la base

```php
namespace catalogue\models;
use \Illuminate\Database\Eloquent\Model ;


class Product extends Model {


    protected $table = 'product';
    protected $primaryKey = 'reference';

    public  $incrementing = false;;
    public $keyType='string';


}
```

```php
use catalogue\models\Product;
$p = new Product ;
$p->reference = 'P17456-453' ;
$p->nom = 'vélo électrique ville' ;

$p->save() ;
```