



❖ Jointures

- ◆ Les exemples qui suivent expliquent comment faire des jointures dans LINQ
- ◆ C'est complexe et en général, en BD First, on fait autrement, avec une vue sql ou une procédure stockée pour simplifier le tout.
- ◆ De sorte qu'on ne vous demandera pas de faire des jointures LINQ dans ce cours.
- ◆ Les explications qui suivent sont pour les curieux et/ou pour ceux qui font des projets en CodeFirst et n'ont pas la chance de simplifier leur vie en faisant des vues SQL ou des procédures stockées....



❖ Jointures

- ◆ Exemple 1 : La liste d'acteurs pour une seule série particulière
 - Deux types de syntaxes:

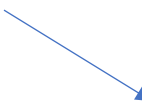
LINQ « Query Syntax »

```
IEnumerable<Acteur> acteursDeLaSerie =  
    from a in acteurs  
    join s in acteurSeries  
    on a.ActeurId equals s.ActeurId  
    where s.SerieId == serie.SerieId  
    select a;
```

LINQ « Method Syntax »

```
List<Acteur> acteursDeLaSerie = acteurs  
    .Join(acteurSeries,  
        acteur => acteur.ActeurId,  
        acteurSerie => acteurSerie.ActeurId,  
        (acteur, acteurSerie) => new { Acteur = acteur, ActeurSerie = acteurSerie })  
    .Where(resultat => resultat.ActeurSerie.SerieId == serie.SerieId)  
    .Select(resultat => resultat.Acteur).ToList();
```

- LINQ donne accès à ces deux syntaxes. N'hésitez pas à utiliser la **Query Syntax**, elle est parfois beaucoup plus intuitive pour des opérations qui sont plus sémantiquement proches de ce qu'on ferait avec SQL. (Jointures, agrégation, etc)
- La **Query Syntax** ne correspond pas parfaitement à la **syntaxe SQL** (L'ordre des instructions peut déstabiliser), mais c'est proche !



```
SELECT A.ActeurID, Prenom, Nom, DateNaissance, DateDeces  
FROM Acteurs.Acteur A  
INNER JOIN Series.ActeurSerie acS  
ON A.ActeurID = acS.ActeurID  
WHERE acS.SerieID = 1
```



❖ Jointures AVEC LINQ

- ◆ Exemple 1 : La liste d'acteurs pour une seule série particulière
 - Dans le contrôleur, on fait donc notre jointure avec LINQ pour créer un ViewModel à envoyer à la vue Razor.
 - Ici on va utiliser LINQ « Method Syntax »

```
public async Task<IActionResult> ActeursSerie(int id)
{
    Serie? serie = await _serieService.Get(id);
    if (serie == null)
    {
        return NotFound();
    }

    IEnumerable<Acteur> acteurs = await _acteurService.GetAll();
    IEnumerable<ActeurSerie> acteurSeries = await _acteurSerieService.GetAll();

    var requete = acteurs
        .Join(acteurSeries,
            acteur => acteur.ActeurId,
            acteurSerie => acteurSerie.ActeurId,
            (acteur, acteurSerie) => new { Acteur = acteur, ActeurSerie = acteurSerie })
        .Where(resultat => resultat.ActeurSerie.SerieId == serie.SerieId)
        .Select(resultat => resultat).ToList();

    List<Acteur> acteursDeLaSerie = new List<Acteur>();

    foreach (var r in requete)
    {
        acteursDeLaSerie.Add(new Acteur {
            ActeurId = r.Acteur.ActeurId,
            Prenom = r.Acteur.Prenom,
            Nom = r.Acteur.Nom,
            DateNaissance = r.Acteur.DateNaissance,
            DateDeces = r.Acteur.DateDeces});
    }

    return View(new ActeursSerieViewModel(serie, acteursDeLaSerie));
}
```



❖ Jointures AVEC LINQ

- ◆ Exemple 1 : La liste d'acteurs pour une seule série particulière
 - Dans le contrôleur, on fait donc notre jointure avec LINQ pour créer un ViewModel à envoyer à la vue Razor.
 - Ici on va utiliser LINQ « Method Syntax »

```
public async Task<IActionResult> ActeursSerie(int id)
{
    Serie? serie = await _serieService.Get(id);
    if (serie == null)
    {
        return NotFound();
    }

    IEnumerable<Acteur> acteurs = await _acteurService.GetAll();
    IEnumerable<ActeurSerie> acteurSeries = await _acteurSerieService.GetAll();

    var requete = acteurs
        .Join(acteurSeries,
            acteur => acteur.ActeurId,
            acteurSerie => acteurSerie.ActeurId,
            (acteur, acteurSerie) => new { Acteur = acteur, ActeurSerie = acteurSerie })
        .Where(resultat => resultat.ActeurSerie.SerieId == serie.SerieId)
        .Select(resultat => resultat).ToList();

    List<Acteur> acteursDeLaSerie = new List<Acteur>();

    foreach (var r in requete)
    {
        acteursDeLaSerie.Add(new Acteur {
            ActeurId = r.Acteur.ActeurId,
            Prenom = r.Acteur.Prenom,
            Nom = r.Acteur.Nom,
            DateNaissance = r.Acteur.DateNaissance,
            DateDeces = r.Acteur.DateDeces});
    }

    return View(new ActeursSerieViewModel(serie, acteursDeLaSerie));
}
```



❖ Jointures

- ◆ Exemple 1 : La liste d'acteurs pour une seule série particulière
 - Pas-à-pas de l'action du contrôleur

- On commence par récupérer la série qui nous intéresse.

- Pour faire une jointure avec les tables Acteur et ActeurSerie, nous aurons besoin de ces deux DbSet

```
public async Task<IActionResult> ActeursSerie(int id)
{
    Serie? serie = await _context.Series.FindAsync(id);
    if (serie == null)
    {
        return NotFound();
    }

    IEnumerable<Acteur> acteurs = await _context.Acteurs.ToListAsync();
    IEnumerable<ActeurSerie> acteurSeries = await _context.ActeurSeries.ToListAsync();
}
```



❖ Jointures

- ◆ Exemple 1 : La liste d'acteurs pour une seule série particulière
 - Pas-à-pas de l'action du contrôleur

- Oof ! Les `Join()` avec LINQ ne sont pas jolis jolis.
- Cette ligne détermine la structure du résultat. (Ici, une liste de duos d'entités `Acteur` + `ActeurSerie`)
- Le `Select()` n'est pas obligatoire. C'est si on avait voulu limiter les données conservées.
- Le type est « `var` », car nous n'avons pas de Model qui correspond au type « Listes de Acteur + ActeurSerie », bien entendu. Le résultat est donc une liste d'**objets anonymes** qui ressemble un peu au tableau en bas à droite de cette diapo.

```
var requete = acteurs
    .Join(acteurSeries,
        acteur => acteur.ActeurId,
        acteurSerie => acteurSerie.ActeurId,
        (acteur, acteurSerie) => new { Acteur = acteur, ActeurSerie = acteurSerie })
    .Where(resultat => resultat.ActeurSerie.SerieId == serie.SerieId)
    .Select(resultat => resultat).ToList();
```

Table 1

Table 2

« ON Id = Id »

Juste garder les acteurs qui ont une correspondance avec la série qui nous intéresse

```
SELECT * FROM Series.ActeurSerie A_S
INNER JOIN Acteurs.Acteur A
ON A.ActeurID = A_S.ActeurID
WHERE A_S.SerieID = 1
```

Équivalent SQL

ActeurSerieID	ActeurID	SerieID	ActeurID	Prenom	Nom	DateNaissance	DateDeces
1	1	1	1	Emma	D'Arcy	1992-06-27	NULL
2	2	1	2	Matt	Smith	1982-10-28	NULL
3	3	1	3	Olivia	Cooke	1993-12-27	NULL



❖ Jointures

- ◆ Exemple 1 : La liste d'acteurs pour une seule série particulière
 - Pas-à-pas de l'action du contrôleur

- Finalement, puisque `var requete` contient tous les acteurs qui nous intéressent, MAIS que sa structure ne correspond pas à `IEnumerable<Acteur>` (Ou `List<Acteur>`), on doit parcourir `requete` pour reconstruire tous les acteurs.

- Ensuite, il nous reste à utiliser notre `ViewModel` pour envoyer la `Serie` et la liste d'`Acteur` à la vue.

```
List<Acteur> acteursDeLaSerie = new List<Acteur>();

foreach(var r in requete)
{
    acteursDeLaSerie.Add(new Acteur {
        ActeurId = r.Acteur.ActeurId,
        Prenom = r.Acteur.Prenom,
        Nom = r.Acteur.Nom,
        DateNaissance = r.Acteur.DateNaissance,
        DateDeces = r.Acteur.DateDeces});
}

return View(new ActeursSerieViewModel(serie, acteursDeLaSerie));
```



❖ Jointures

◆ Exemple 1 : La liste d'acteurs pour une seule série particulière

○ Des alternatives pour simplifier

- Dans ce cas-ci, **seules les informations des acteurs nous intéressent** une fois qu'on a éliminé les acteurs qui ne font pas partie de la série choisie. (Grâce au **.Where**)

- On peut donc utiliser le **.Select()** pour réorganiser la structure du résultat obtenu pour ne garder QUE la portion « **Acteur** » (Et se débarrasser de la portion « **ActeurSerie** » dont on avait besoin pour la jointure)

- Cela permet d'immédiatement insérer le résultat dans une **List<Acteur>**. (Plus besoin de la boucle d'après)

```
var requete = acteurs
    .Join(acteurSeries,
    acteur => acteur.ActeurId,
    acteurSerie => acteurSerie.ActeurId,
    (acteur, acteurSerie) => new { Acteur = acteur, ActeurSerie = acteurSerie })
    .Where(resultat => resultat.ActeurSerie.SerieId == serie.SerieId)
    .Select(resultat => resultat).ToList();

List<Acteur> acteursDeLaSerie = new List<Acteur>();

foreach(var r in requete)
{
    acteursDeLaSerie.Add(new Acteur {
        ActeurId = r.Acteur.ActeurId,
        Prenom = r.Acteur.Prenom,
        Nom = r.Acteur.Nom,
        DateNaissance = r.Acteur.DateNaissance,
        DateDeces = r.Acteur.DateDeces});
}
```

Devient

- C'est comme si on avait juste gardé les infos de la table **Acteur** après la jointure avec **ActeurSerie**.

Prenom	Nom	DateNaissance	DateDeces
Emma	D'Arcy	1992-06-27	NULL
Matt	Smith	1982-10-28	NULL
Olivia	Cooke	1993-12-27	NULL

```
List<Acteur> acteursDeLaSerie = acteurs
    .Join(acteurSeries,
    acteur => acteur.ActeurId,
    acteurSerie => acteurSerie.ActeurId,
    (acteur, acteurSerie) => new { Acteur = acteur, ActeurSerie = acteurSerie })
    .Where(resultat => resultat.ActeurSerie.SerieId == serie.SerieId)
    .Select(resultat => resultat.Acteur).ToList();
```


❖ Select (Retrieve)

◆ GroupBy() avec Query Syntax (Plus intuitif qu'avec la Method Syntax)

Dans le segment « **group x by y into result** » :

- **y** est ce qu'on aurait mis dans notre GROUP BY en SQL.
- **x** est le « restant » qui sera absorbé par le groupement.
- **result** est le résultat. La / les données de groupement sont accessibles avec **result.Key** (Ici, on groupe par série, alors **result.Key** est une **série entière**) Avec **result**, on peut faire une fonction d'agrégation comme Count(), Sum(), Max(), etc.

Nom	AnneeDebut	AnneeFin	NbActeurs
House Of The Dragon	2022	NULL	3
The Crown	2016	NULL	3
The Tudors	2007	2010	3
The Witcher	2019	NULL	2
White Lotus	2021	NULL	3

- Comme on a généré une liste d'objets anonymes, on utilise ensuite une boucle pour restructurer en une liste de notre **ViewModel**.

```
SELECT S.Nom, S.AnneeDebut, S.AnneeFin, COUNT(A_S.ActeurID) AS 'NbActeurs'
FROM Series.Serie S
INNER JOIN Series.ActeurSerie A_S
ON A_S.SerieID = S.SerieID
GROUP BY S.Nom, S.AnneeDebut, S.AnneeFin
```

```
public async Task<IActionResult> NbActeursParSerie()
{
    IEnumerable<Serie> series = await _context.Series.ToListAsync();
    IEnumerable<ActeurSerie> acteurSeries = await _context.ActeurSeries.ToListAsync();

    /*Query-syntax*/
    var requete = from s in series
                  join a_s in acteurSeries
                    on s.SerieId equals a_s.SerieId
                  group a_s.ActeurId by s into result
                  select new { serie = result.Key, nbActeurs = result.Count()};
    

    List<SerieAvecNbActeursViewModel> seriesAvecNbActeurs = new List<SerieAvecNbActeursViewModel>();

    foreach(var r in requete)
    {
        seriesAvecNbActeurs.Add(new SerieAvecNbActeursViewModel(r.serie, r.nbActeurs));
    }

    return View(seriesAvecNbActeurs);
}
```

```
SELECT S.Nom, S.AnneeDebut, S.AnneeFin, COUNT(A_S.ActeurID) AS 'NbActeurs'
FROM Series.Serie S
INNER JOIN Series.ActeurSerie A_S
ON A_S.SerieID = S.SerieID
GROUP BY S.Nom, S.AnneeDebut, S.AnneeFin
```

❖ Select (Retrieve)

◆ GroupBy() avec Method Syntax

- Le résultat est totalement identique, donc le reste de la méthode ne change pas.

Nom	AnneeDebut	AnneeFin	NbActeurs
House Of The Dragon	2022	NULL	3
The Crown	2016	NULL	3
The Tudors	2007	2010	3
The Witcher	2019	NULL	2
White Lotus	2021	NULL	3

- Le premier paramètre est l'ensemble de colonne(s) utilisé pour le groupement. (Une **série en entier**)
- La deuxième paramètre est la ou les valeurs groupées. (**ActeurId**)

```
var requete = series
    .Join(
        acteurSeries,
        s => s.SerieId,
        a_s => a_s.SerieId,
        (s, a_s) => new { s, a_s }
    )
    .GroupBy(
        sa => sa.s,
        sa => sa.a_s.ActeurId
    )
    .Select(g => new {
        serie = g.Key,
        nbActeurs = g.Count()
    });
```