

# Contents

[Entity Framework](#)

[EF Core & EF6](#)

[Comparer EF Core et EF6](#)

[Déplacer d'EF6 vers EF Core](#)

[Vue d'ensemble](#)

[Portage d'un modèle basé sur le format EDMX](#)

[Portage d'un modèle basé sur du code](#)

[Utiliser EF6 et EF Core dans la même application](#)

[Entity Framework Core](#)

[Vue d'ensemble](#)

[Nouveautés](#)

[Vue d'ensemble](#)

[EF Core 3.0](#)

[Nouvelles fonctionnalités](#)

[Modifications avec rupture](#)

[EF Core 2.2](#)

[EF Core 2.1](#)

[EF Core 2.0](#)

[EF Core 1.1](#)

[EF Core 1.0](#)

[Mise à niveau à partir de versions antérieures](#)

[De 1.0 RC1 vers RC2](#)

[De 1.0 RC2 vers RTM](#)

[De 1.x vers 2.0](#)

[Bien démarrer](#)

[Tutoriel EF Core](#)

[Installation d'EF Core](#)

[Tutoriel ASP.NET Core >>](#)

[Notions de base](#)

- [Chaînes de connexion](#)
- [Journalisation](#)
- [Résilience des connexions](#)
- [Test](#)
  - [Vue d'ensemble](#)
  - [Test avec SQLite](#)
  - [Test avec InMemory](#)
- [Configuration d'un DbContext](#)
- [Types références Nullables](#)
- [Création d'un modèle](#)
  - [Vue d'ensemble](#)
  - [Types d'entités](#)
  - [Propriétés d'entité](#)
  - [Touches](#)
  - [Valeurs générées](#)
  - [Jetons d'accès concurrentiel](#)
  - [Propriétés cachées](#)
  - [Relations](#)
  - [Index](#)
  - [Héritage](#)
  - [Séquences](#)
  - [Champs de stockage](#)
  - [Conversions de valeurs](#)
  - [Amorçage des données](#)
  - [Constructeurs de type d'entité](#)
  - [Fractionnement de table](#)
  - [Types d'entité détenus](#)
  - [Types d'entité sans clé](#)
  - [Alternance de modèles avec un même DbContext](#)
  - [Données spatiales](#)
- [Gestion des schémas de base de données](#)
  - [Vue d'ensemble](#)

## Migrations

- Vue d'ensemble
- Environnements d'équipe
- Opérations personnalisées
- Utilisation d'un projet distinct
- Plusieurs fournisseurs
- Table d'historique personnalisée

## Créer et supprimer des API

- Ingénierie à rebours (génération de modèles automatique)

## Interrogation des données

- Vue d'ensemble
- Évaluation sur le client ou le serveur
- Requêtes avec suivi ou sans suivi
- Opérateurs de requête complexes
- Chargement des données associées
- Requêtes asynchrones
- Requêtes SQL brutes
- Filtres de requête globale
- Balises de requête
- Fonctionnement d'une requête

## Enregistrement de données

- Vue d'ensemble
- Enregistrement de base
- Données associées
- Suppression en cascade
- Conflits d'accès concurrentiel
- Transactions
- Enregistrement asynchrone
- Entités déconnectées
- Valeurs explicites des propriétés générées
- Implémentations de .NET prises en charge
- Fournisseurs de bases de données

[Vue d'ensemble](#)

[Microsoft SQL Server](#)

- [Vue d'ensemble](#)
- [Tables à mémoire optimisée](#)
- [Spécification des options Azure SQL Database](#)

[SQLite](#)

- [Vue d'ensemble](#)
- [Limitations de SQLite](#)

[Cosmos](#)

- [Vue d'ensemble](#)
- [Utilisation de données non structurées](#)
- [Limitations de Cosmos](#)

[InMemory \(pour les tests\)](#)

[Écriture d'un fournisseur de base de données](#)

[Modifications ayant un impact sur le fournisseur](#)

[Outils et extensions](#)

[Référence de la ligne de commande](#)

- [Vue d'ensemble](#)
- [Console du Gestionnaire de package \(Visual Studio\)](#)
- [CLI .NET Core](#)
- [Création de DbContext au moment de la conception](#)
- [Services à la conception](#)

[Informations de référence sur l'API EF Core >>](#)

[Entity Framework 6](#)

- [Vue d'ensemble](#)
- [Nouveautés](#)
- [Vue d'ensemble](#)
- [Versions précédentes](#)
- [Mise à niveau vers EF6](#)
- [Versions de Visual Studio](#)

[Bien démarrer](#)

[Notions de base](#)

- Obtenir Entity Framework
- Utilisation de DbContext
- Présentation des relations
- Interrogation et enregistrement en mode asynchrone
- Configuration
  - Basée sur le code
  - Fichier config
  - Chaînes de connexion
  - Résolution des dépendances
- Gestion des connexions
- Résilience des connexions
  - Logique de nouvelle tentative
  - Échecs de validation de transaction
- Liaison de données
  - WinForms
  - WPF
- Entités déconnectées
  - Vue d'ensemble
  - Entités de suivi automatique
    - Vue d'ensemble
    - Procédure pas à pas
- Journalisation et interception
- Performances
  - Considérations sur les performances (livre blanc)
  - Utilisation de NGEN
  - Utilisation des vues prégénérées
- Fournisseurs
  - Vue d'ensemble
  - Modèle de fournisseur EF6
  - Prise en charge spatiale dans les fournisseurs
- Utilisation de proxys
- Test avec EF6

- Utilisation de la simulation
- Écriture de vos propres doubles de test
- Testabilité avec EF4 (Article)
- Création d'un modèle
  - Vue d'ensemble
  - Utilisation de Code First
    - Flux de travail
      - Avec une nouvelle base de données
      - Avec une base de données existante
    - Annotations de données
    - DbSets
    - Types de données
      - Enums
      - Spatial
    - Conventions
      - Conventions intégrées
      - Conventions personnalisées
      - Conventions de modèle
    - Configuration de Fluent
      - Relations
      - Type et propriétés
      - Utilisation dans Visual Basic
      - Mappage de procédures stockées
    - Migrations
      - Vue d'ensemble
      - Migrations automatiques
      - Utilisation de bases de données existantes
      - Personnalisation de l'historique des migrations
      - Utilisation de Migrate.exe
      - Migrations dans les environnements d'équipe
    - Utilisation d'EF Designer
    - Flux de travail

[Model-First](#)

[Database-First](#)

[Types de données](#)

[Types complexes](#)

[Enums](#)

[Spatial](#)

[Fractionner les mappages](#)

[Fractionnement d'entité](#)

[Fractionnement de table](#)

[Mappages d'héritage](#)

[Table par hiérarchie](#)

[Table par type](#)

[Mappage de procédures stockées](#)

[Query](#)

[Mise à jour](#)

[Mappage des relations](#)

[Diagrammes multiples](#)

[Choix de la version de runtime](#)

[Génération de code](#)

[Vue d'ensemble](#)

[ObjectContext hérité](#)

[Avancé](#)

[Format de fichier EDMX](#)

[Requête de définition](#)

[Jeux de résultats multiples](#)

[Fonctions table](#)

[Raccourcis clavier](#)

[Interrogation des données](#)

[Vue d'ensemble](#)

[Load, méthode](#)

[Données locales](#)

[Requêtes avec et sans suivi](#)

- Utilisation de requêtes SQL brutes
- Interrogation de données associées
- Enregistrement de données
  - Vue d'ensemble
  - Suivi des modifications
    - Déetecter automatiquement les changements
    - État d'entité
    - Valeurs de propriété
  - Gestion de conflits d'accès concurrentiel
  - Utilisation de transactions
  - Validation de données
- Ressources supplémentaires
  - Blogs
  - Études de cas
  - Collaboration
  - Obtenir de l'aide
  - Glossaire
  - Exemple de base de données School
  - Outils et extensions
  - Licences
  - EF5
    - Chinois simplifié
    - Chinois traditionnel
    - Allemand
    - Anglais
    - Espagnol
    - Français
    - Italien
    - Japonais
    - Coréen
    - Russe
  - EF6

[Version préliminaire](#)

[Chinois simplifié](#)

[Chinois traditionnel](#)

[Allemand](#)

[Anglais](#)

[Espagnol](#)

[Français](#)

[Italien](#)

[Japonais](#)

[Coréen](#)

[Russe](#)

[Informations de référence sur les API EF6 >>](#)

# Documentation d'Entity Framework

## Entity Framework

Entity Framework est un mapeur objet-relationnel (ORM) qui permet aux développeurs .NET de travailler avec une base de données en utilisant des objets .NET. Du coup, ils n'ont plus à écrire une grande partie du code d'accès aux données qu'ils doivent généralement écrire.

□

## Entity Framework Core

EF Core est une version légère, extensible et multiplateforme d'Entity Framework.

□

## Entity Framework 6

EF6 est une technologie d'accès aux données éprouvée, connue depuis de nombreuses années pour sa stabilité et la richesse de ses fonctionnalités.

□

## Choix

Déterminez la version d'EF qui vous convient.

□

## Porter vers EF Core

Conseils pour le portage d'une application EF 6 vers EF Core.

EF Core

tout

EF Core est une version légère, extensible et multiplateforme d'Entity Framework.

□

## Bien démarrer

Vue d'ensemble

Créer un modèle

Interroger les données

Enregistrer les données

□

## Didacticiels

Plus...

□

## Fournisseurs de bases de données

[SQL Server](#)

[MySQL](#)

[PostgreSQL](#)

[SQLite](#)

[Cosmos](#)

[Plus...](#)

□

## **Informations de référence sur l'API**

[DbContext](#)

[DbSet< TEntity >](#)

[Plus...](#)

[EF 6](#)

EF6 est une technologie d'accès aux données éprouvée, connue depuis de nombreuses années pour sa stabilité et la richesse de ses fonctionnalités.

□

## **Bien démarrer**

Découvrez comment accéder aux données avec Entity Framework 6.

□

## **Informations de référence sur les API**

Accédez à l'API Entity Framework 6, organisée par espace de noms.

# Comparer EF Core et EF6

10/01/2020 • 9 minutes to read • [Edit Online](#)

Entity Framework est un mapeur objet-relationnel (ORM) pour .NET. Cet article compare les deux versions : Entity Framework 6 et Entity Framework Core.

## Entity Framework 6

Entity Framework 6 (EF6) est une technologie d'accès aux données éprouvée. Il a été publié pour la première fois en 2008, dans le cadre de .NET Framework 3.5 SP1 et de Visual Studio 2008 SP1. À compter de la version 4.1, il est fourni en tant que package NuGet [EntityFramework](#). EF6 s'exécute sur .NET Framework 4.x et .NET Core à partir de 3.0.

EF6 continue d'être un produit pris en charge et de bénéficier de correctifs de bogues et d'améliorations mineures.

## Entity Framework Core

Entity Framework Core (EF Core) est une réécriture complète d'EF6 publiée en 2016. Il est fourni dans des packages Nuget, le principal étant [Microsoft.EntityFrameworkCore](#). EF Core est un produit multiplateforme qui s'exécute sur .NET Core.

EF Core a été conçu pour fournir une expérience de développement similaire à EF6. La plupart des API de niveau supérieur restent les mêmes ; EF Core semblera donc familier aux développeurs EF6.

## Comparaison des fonctionnalités

EF Core offre de nouvelles fonctionnalités qui ne seront pas implémentées dans EF6 (telles que les [clés secondaires](#), les [mises à jour par lot](#) et l'[évaluation mixte client/base de données dans les requêtes LINQ](#)). Mais comme il s'agit d'une nouvelle base de code, il manque également certaines fonctionnalités présentes dans EF6.

Les tableaux suivants comparent les fonctionnalités disponibles dans EF Core et EF6. Il s'agit d'une comparaison générale qui ne répertorie pas toutes les fonctionnalités et n'explique pas les différences entre la même fonctionnalité dans les différentes versions d'EF.

La colonne EF Core indique la version du produit dans laquelle la fonctionnalité a été introduite pour la première fois.

### Création d'un modèle

| FONCTIONNALITÉ                      | EF 6 | EF CORE |
|-------------------------------------|------|---------|
| Mappage des classes de base         | Oui  | 1.0     |
| Constructeurs avec des paramètres   |      | 2.1     |
| Conversions de valeurs de propriété |      | 2.1     |
| Types mappés sans clé               |      | 2.1     |
| Conventions                         | Oui  | 1.0     |

| FONCTIONNALITÉ   | EF 6    | EF CORE       |
|--|---------|---------------|
| Conventions personnalisées   | Oui     | 1.0 (partiel) |
| Annotations de données   | Oui     | 1.0           |
| API Fluent   | Oui     | 1.0           |
| Héritage : table par hiérarchie (TPH)                              | Oui     | 1.0           |
| Héritage : table par type (TPT)                                    | Oui     |               |
| Héritage : table par classe concrète (TPC)                         | Oui     |               |
| Propriétés d'état de clichés instantanés                           |         | 1.0           |
| Clés secondaires   |         | 1.0           |
| Plusieurs-à-plusieurs sans entité de jonction                      | Oui     |               |
| Génération de la clé : Base de données                             | Oui     | 1.0           |
| Génération de la clé : Client                                      |         | 1.0           |
| Types complexes/détenus  | Oui     | 2.0           |
| Données spatiales  | Oui     | 2.2           |
| Visualisation graphique de modèle                                  | Oui     |               |
| Éditeur de modèle graphique  | Oui     |               |
| Format de modèle : Code  | Oui     | 1.0           |
| Format de modèle : EDMX (XML)                                      | Oui     |               |
| Créer un modèle à partir d'une base de données : Ligne de commande | Oui     | 1.0           |
| Créer un modèle à partir d'une base de données : Assistant VS      | Oui     |               |
| Mise à jour d'un modèle à partir d'une base de données             | Partial |               |
| Filtres de requête globale   |         | 2.0           |
| Fractionnement de table  | Oui     | 2.0           |
| Fractionnement d'entité  | Oui     |               |

| FONCTIONNALITÉ                                    | EF 6     | EF CORE |
|---|----------|---------|
| Mappage de fonctions scalaires de base de données | Médiocre | 2.0     |
| Mappage de champs                                 |          | 1.1     |
| Types références Nullables (C# 8.0)               |          | 3.0     |

### Interrogation des données

| FONCTIONNALITÉ   | EF6      | EF CORE                                    |
|--|----------|--|
| Requêtes LINQ  | Oui      | 1.0 (en cours pour les requêtes complexes) |
| Code SQL généré lisible  | Médiocre | 1.0  |
| Traduction GroupBy   | Oui      | 2.1  |
| Chargement des données associées : hâtif                                   | Oui      | 1.0  |
| Chargement des données associées : Chargement hâtif pour les types dérivés |          | 2.1  |
| Chargement des données associées : Différé                                 | Oui      | 2.1  |
| Chargement des données associées : Explicit                                | Oui      | 1.1  |
| Requêtes SQL brutes : Types d'entité                                       | Oui      | 1.0  |
| Requêtes SQL brutes : Types d'entité sans clé                              | Oui      | 2.1  |
| Requêtes SQL brutes : Composition avec LINQ                                |          | 1.0  |
| Requêtes compilées explicitement   | Médiocre | 2.0  |
| Langage de requête textuel (Entity SQL)                                    | Oui      |  |
| await foreach (C# 8.0)   |          | 3.0  |

### Enregistrement de données

| FONCTIONNALITÉ                         | EF6 | EF CORE |
|--|-----|---------|
| Suivi des modifications : Instantané   | Oui | 1.0     |
| Suivi des modifications : Notification | Oui | 1.0     |

| FONCTIONNALITÉ                           | EF6      | EF CORE       |
|--|----------|---------------|
| Suivi des modifications : Serveurs proxy | Oui      |               |
| État du suivi de l'accès                 | Oui      | 1.0           |
| Accès concurrentiel optimiste            | Oui      | 1.0           |
| Transactions                             | Oui      | 1.0           |
| Traitement par lot d'instructions        |          | 1.0           |
| Mappage de procédure stockée             | Oui      |               |
| API de bas niveau à graphes déconnectés  | Médiocre | 1.0           |
| De bout en bout à graphes déconnectés    |          | 1.0 (partiel) |

### Autres fonctionnalités

| FONCTIONNALITÉ   | EF6 | EF CORE |
|--|-----|---------|
| Migrations   | Oui | 1.0     |
| API de création/suppression de base de données           | Oui | 1.0     |
| Données seed   | Oui | 2.1     |
| Résilience de la connexion                               | Oui | 1.1     |
| Raccordements de cycle de vie (événements, interception) | Oui |         |
| Journalisation simple (Database.Log)                     | Oui |         |
| Regroupement DbContext                                   |     | 2.0     |

### Fournisseurs de bases de données

| FONCTIONNALITÉ | EF6 | EF CORE |
|----------------|-----|---------|
| SQL Server     | Oui | 1.0     |
| MySQL          | Oui | 1.0     |
| PostgreSQL     | Oui | 1.0     |
| Oracle         | Oui | 1.0     |
| SQLite         | Oui | 1.0     |

| FONCTIONNALITÉ             | EF6 | EF CORE            |
|----------------------------|-----|--------------------|
| SQL Server Compact         | Oui | 1.0 <sup>(1)</sup> |
| DB2                        | Oui | 1.0                |
| Firebird                   | Oui | 2.0                |
| Jet (Microsoft Access)     |     | 2.0 <sup>(1)</sup> |
| Cosmos DB                  |     | 3.0                |
| In-memory (pour les tests) |     | 1.0                |

<sup>1</sup> Les fournisseurs SQL Server Compact et Jet fonctionnent uniquement sur le .NET Framework (et non sur .NET Core).

## Implémentations de .NET

| FONCTIONNALITÉ | EF6                   | EF CORE                 |
|----------------|-----------------------|-------------------------|
| .NET Framework | Oui                   | 1.0 (supprimé dans 3.0) |
| .NET Core      | Oui (ajouté dans 6.3) | 1.0                     |
| Mono & Xamarin |                       | 1.0 (en cours)          |
| UWP            |                       | 1.0 (en cours)          |

## Conseils pour les nouvelles applications

Utilisez plutôt EF Core pour une nouvelle application si les deux conditions suivantes sont remplies :

- L'application a besoin des fonctionnalités de .NET Core. Pour plus d'informations, consultez [Choix entre .NET Core et .NET Framework pour les applications serveur](#).
- EF Core prend en charge toutes les fonctionnalités requises par l'application. Si une fonctionnalité souhaitée est manquante, consultez la [feuille de route EF Core](#) pour savoir si sa prise en charge est prévue à l'avenir.

Utilisez plutôt EF6 si les deux conditions suivantes sont remplies :

- L'application s'exécutera sur Windows et .NET Framework 4.0 ou version ultérieure.
- EF6 prend en charge toutes les fonctionnalités requises par l'application.

## Conseils pour les applications EF6 existantes

En raison des modifications importantes apportées à EF Core, nous vous déconseillons de migrer une application EF6 vers EF Core, à moins d'avoir une raison justifiant réellement ce changement. Si vous souhaitez passer à EF Core pour utiliser de nouvelles fonctionnalités, vérifiez bien ses limitations. Pour plus d'informations, consultez [Portage d'EF6 vers EF Core. Le déplacement d'EF6 vers EF Core est plus un portage qu'une mise à niveau.](#)

## Étapes suivantes

Pour plus d'informations, consultez la documentation :

- [Vue d'ensemble d'Entity Framework Core - EF Core](#)
- [Vue d'ensemble d'Entity Framework 6 - EF6](#)

# Portage depuis EF6 vers EF Core

11/10/2019 • 6 minutes to read

En raison des modifications importantes apportées à EF Core, nous vous déconseillons de migrer une application EF6 vers EF Core, à moins d'avoir une raison justifiant réellement ce changement. Il est préférable de considérer la migration d'EF6 vers EF Core comme un portage plutôt qu'une mise à niveau.

## IMPORTANT

Avant de commencer le processus de portage, il est important de vérifier qu'EF Core répond aux exigences d'accès aux données de votre application.

## Fonctionnalités manquantes

Assurez-vous qu'EF Core dispose de toutes les fonctionnalités que vous devez utiliser dans votre application. Consultez [Comparaison des fonctionnalités](#) pour une comparaison détaillée de la façon dont la fonctionnalité définie dans EF Core est comparée à EF6. Si des fonctionnalités requises sont manquantes, assurez-vous que vous pouvez compenser leur absence avant de procéder au portage vers EF Core.

## Changements de comportement

Il s'agit d'une liste non exhaustive de certaines modifications de comportement entre EF6 et EF Core. Il est important de les garder à l'esprit lors du port de votre application, car elles peuvent modifier son comportement, mais ne s'afficheront pas en tant qu'erreurs de compilation après le passage à EF Core.

### DbSet. Add/Attach et le comportement du graphique

Dans EF6, appeler `DbSet.Add()` sur une entité entraîne une recherche récursive pour toutes les entités référencées dans ses propriétés de navigation. Toutes les entités qui sont détectées et ne sont pas déjà suivies par le contexte sont également marquées comme ajoutées. `DbSet.Attach()` se comporte de la même manière, sauf que toutes les entités sont marquées comme inchangées.

### EF Core effectue une recherche récursive similaire, mais avec des règles légèrement différentes.

- L'entité racine est toujours dans l'état demandé (ajouté pour `DbSet.Add` et inchangé pour `DbSet.Attach`).
- Pour les entités qui sont détectées lors de la recherche récursive des propriétés de navigation :
  - Si la clé primaire de l'entité est générée par le magasin
    - Si la clé primaire n'est pas définie sur une valeur, l'état est défini sur ajouté. La valeur de clé primaire est considérée comme « non définie » si elle est affectée à la valeur CLR par défaut pour le type de propriété (par exemple, `0` pour `int`, `null` pour `string`, etc.).
    - Si la clé primaire n'est pas définie sur une valeur, l'état est défini sur inchangé.
  - Si la clé primaire n'est pas générée par la base de données, l'entité est mise dans le même état que la racine.

### Initialisation de la base de données Code First

EF6 dispose d'un grand nombre de commandes magic exécutées dans le contexte de la sélection de la connexion de base de données et de l'initialisation de la base de données. Voici quelques-unes de ces règles :

- Si aucune configuration n'est effectuée, EF6 sélectionne une base de données sur SQL Express ou sur une base de données locale.
- Si une chaîne de connexion portant le même nom que le contexte se trouve dans le fichier `App/Web.config` des applications, cette connexion est utilisée.
- Si la base données n'existe pas, elle est créée.
- Si aucune des tables du modèle n'existe dans la base de données, le schéma du modèle actuel est ajouté à la base de données. Si les migrations sont activées, elles sont utilisées pour créer la base de données.
- Si la base de données existe et qu'EF6 a créé le schéma précédemment, la compatibilité du schéma avec le modèle actuel est vérifiée. Une exception est levée si le modèle a changé depuis la création du schéma.

#### **EF Core n'exécute aucun de ces magics.**

- La connexion de base de données doit être configurée de manière explicite dans le code.
- Aucune initialisation n'est effectuée. Vous devez utiliser `DbContext.Database.Migrate()` pour appliquer des migrations (ou `DbContext.Database.EnsureCreated()` et `EnsureDeleted()` pour créer/supprimer la base de données sans utiliser les migrations).

#### **Convention d'affectation de noms de la table Code First**

EF6 exécute le nom de la classe d'entité via un service de pluralisation pour calculer le nom de la table par défaut à laquelle l'entité est mappée.

EF Core utilise le nom de la propriété `DbSet` dans laquelle l'entité est exposée dans le contexte dérivé. Si l'entité n'a pas de propriété `DbSet`, le nom de la classe est utilisé.

# Portage d'un modèle basé sur EF6 EDMX vers EF Core

11/10/2019 • 2 minutes to read

EF Core ne prend pas en charge le format de fichier EDMX pour les modèles. La meilleure option pour porter ces modèles consiste à générer un nouveau modèle basé sur du code à partir de la base de données de votre application.

## Installer EF Core des packages NuGet

Installez le package NuGet `Microsoft.EntityFrameworkCore.Tools`.

## Régénérer le modèle

Vous pouvez maintenant utiliser la fonctionnalité d'ingénierie à rebours pour créer un modèle basé sur votre base de données existante.

Exécutez la commande suivante dans la console du gestionnaire de package (outils –> Gestionnaire de package NuGet –> console du gestionnaire de package). Consultez [console du gestionnaire de package \(Visual Studio\)](#) pour obtenir des options de commande permettant d'effectuer une génération de modèles automatique d'un sous-ensemble de tables.

```
Scaffold-DbContext "<connection string>" <database provider name>
```

Par exemple, voici la commande pour générer un modèle de structure à partir de la base de données de blogs sur votre instance SQL Server de base de données locale.

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer
```

## Supprimer le modèle EF6

Vous pouvez maintenant supprimer le modèle EF6 de votre application.

Il est parfait de conserver le package NuGet EF6 (EntityFramework) installé, car EF Core et EF6 peuvent être utilisés côte à côte dans la même application. Toutefois, si vous n'avez pas l'intention d'utiliser EF6 dans les zones de votre application, la désinstallation du package permet d'obtenir des erreurs de compilation sur des portions de code nécessitant votre attention.

## Mettre à jour votre code

À ce stade, il s'agit de traiter les erreurs de compilation et de consulter le code pour voir si les changements de comportement entre EF6 et EF Core ont un impact sur vous.

## Tester le port

Simplement parce que votre application est compilée, ne signifie pas qu'elle est correctement reportée vers EF Core. Vous devrez tester toutes les zones de votre application pour vous assurer qu'aucune des modifications de

comportement n'a un impact négatif sur votre application.

# Portage d'un modèle basé sur du code EF6 vers EF Core

11/10/2019 • 4 minutes to read

Si vous avez lu tous les avertissements et que vous êtes prêt à utiliser le port, voici quelques conseils pour vous aider à démarrer.

## Installer EF Core des packages NuGet

Pour utiliser EF Core, vous installez le package NuGet pour le fournisseur de base de données que vous souhaitez utiliser. Par exemple, lorsque vous ciblez SQL Server, vous devez installer `Microsoft.EntityFrameworkCore.SqlServer`. Pour plus d'informations, consultez [fournisseurs de bases de données](#).

Si vous envisagez d'utiliser des migrations, vous devez également installer le package `Microsoft.EntityFrameworkCore.Tools`.

Il est parfait de conserver le package NuGet EF6 (`EntityFramework`) installé, car EF Core et EF6 peuvent être utilisés côté à côté dans la même application. Toutefois, si vous n'avez pas l'intention d'utiliser EF6 dans les zones de votre application, la désinstallation du package permet d'obtenir des erreurs de compilation sur des portions de code nécessitant votre attention.

## Permuter les espaces de noms

La plupart des API que vous utilisez dans EF6 se trouvent dans l'espace de noms `System.Data.Entity` (et les sous-espaces de noms associés). La première modification du code consiste à basculer vers l'espace de noms `Microsoft.EntityFrameworkCore`. En général, vous démarrez avec votre fichier de code de contexte dérivé, puis vous travaillez à partir de là, en résolvant les erreurs de compilation à mesure qu'elles se produisent.

## Configuration du contexte (connexion, etc.)

Comme décrit dans [la section s'assurer EF Core fonctionne pour votre application](#), EF Core a moins de magie pour détecter la base de données à laquelle se connecter. Vous devez remplacer la méthode `OnConfiguring` sur votre contexte dérivé et utiliser l'API spécifique du fournisseur de base de données pour configurer la connexion à la base de données.

La plupart des applications EF6 stockent la chaîne de connexion dans le fichier des applications `App/Web.config`. Dans EF Core, vous lisez cette chaîne de connexion à l'aide de l'API  `ConfigurationManager`. Vous devrez peut-être ajouter une référence à l'assembly de Framework `System.Configuration` pour pouvoir utiliser cette API.

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(ConfigurationManager.ConnectionStrings["BloggingDatabase"].ConnectionString);
    }
}
```

## Mettre à jour votre code

À ce stade, il s'agit de traiter les erreurs de compilation et de consulter le code pour voir si les changements de comportement ont un impact sur vous.

## Migrations existantes

Il n'existe pas vraiment un moyen pratique de porter des migrations EF6 existantes vers EF Core.

Si possible, il est préférable de supposer que toutes les migrations précédentes de EF6 ont été appliquées à la base de données, puis de commencer à migrer le schéma à partir de ce point à l'aide de EF Core. Pour ce faire, vous devez utiliser la commande `Add-Migration` pour ajouter une migration une fois que le modèle est porté sur EF Core. Vous supprimez ensuite tout le code des méthodes `Up` et `Down` de la migration par génération de modèles automatique. Les migrations suivantes seront comparées au modèle lors de la génération de modèles automatique de la migration initiale.

## Tester le port

Simplement parce que votre application est compilée, ne signifie pas qu'elle est correctement reportée vers EF Core. Vous devrez tester toutes les zones de votre application pour vous assurer qu'aucune des modifications de comportement n'a un impact négatif sur votre application.

# Utilisation conjointe d'EF Core et d'EF6 dans la même application

20/09/2019 • 2 minutes to read • [Edit Online](#)

Il est possible d'utiliser EF Core et EF6 dans la même application ou bibliothèque en installant les deux packages NuGet.

Certains types portent les mêmes noms dans EF Core et EF6 et se distinguent uniquement par l'espace de noms, ce qui peut compliquer l'utilisation conjointe d'EF Core et d'EF6 dans le même fichier de code. L'ambiguïté peut être facilement supprimée en utilisant des directives d'alias d'espace de noms. Par exemple :

```
using Microsoft.EntityFrameworkCore; // use DbContext for EF Core
using EF6 = System.Data.Entity; // use EF6.DbContext for the EF6 version
```

Si vous déplacez une application existante qui possède plusieurs modèles EF, vous pouvez, si vous le souhaitez, déplacer spécifiquement certains d'entre eux dans EF Core et continuer à utiliser EF6 pour les autres.

# Entity Framework Core

07/11/2019 • 3 minutes to read • [Edit Online](#)

Entity Framework (EF) Core est une version légère, extensible, [open source](#) et multiplateforme de la très connue technologie d'accès aux données Entity Framework.

EF Core peut servir de mappeur relationnel/objet (O/RM), permettant aux développeurs .NET de travailler avec une base de données à l'aide d'objets .NET, et éliminant la nécessité de la plupart du code d'accès aux données qu'ils doivent généralement écrire.

EF Core prend en charge de nombreux moteurs de base de données ; consultez [Fournisseurs de base de données](#) pour plus d'informations.

## Modèle

Avec EF Core, l'accès aux données est effectué à l'aide d'un modèle. Un modèle est composé de classes d'entités et d'un objet de contexte représentant une session avec la base de données, ce qui permet d'interroger et d'enregistrer des données. Pour en savoir plus, consultez [Création d'un modèle](#).

Vous pouvez générer un modèle à partir d'une base de données existante, coder manuellement un modèle en fonction de votre base de données ou utiliser [EF Migrations](#) pour créer une base de données à partir de votre modèle et la faire évoluer au même rythme que celui-ci.

```

using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

namespace Intro
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer(
                @"Server=(localdb)\mssqllocaldb;Database=Blogging;Integrated Security=True");
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }
        public int Rating { get; set; }
        public List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}

```

## Interrogation

Les instances de vos classes d'entité sont récupérées de la base de données à l'aide de LINQ (Language Integrated Query). Pour en savoir plus, consultez [Interrogation des données](#).

```

using (var db = new BloggingContext())
{
    var blogs = db.Blogs
        .Where(b => b.Rating > 3)
        .OrderBy(b => b.Url)
        .ToList();
}

```

## Enregistrement de données

Les données sont créées, supprimées et modifiées dans la base de données à l'aide d'instances de vos classes d'entité. Pour en savoir plus, consultez [Enregistrement de données](#).

```

using (var db = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    db.Blogs.Add(blog);
    db.SaveChanges();
}

```

## Étapes suivantes

Pour des tutoriels d'introduction, consultez [Bien démarrer avec Entity Framework Core](#).

# Nouveautés dans EF Core

25/09/2019 • 8 minutes to read • [Edit Online](#)

## Versions récentes

- **EF Core 3.0** (dernière version stable)
  - Nouvelles fonctionnalités
  - Changements cassants dont vous devez être conscient lors de la mise à niveau
- **EF Core 2.2**
- **EF Core 2.1** (dernière version de support à long terme)

## Feuille de route du produit

### IMPORTANT

Notez que les fonctionnalités et les plannings des versions ultérieures sont susceptibles de changer à tout moment, et même si cette page est régulièrement mise à jour, elle risque de ne pas toujours refléter nos projets les plus récents.

### Versions futures

- **EF Core 3.1**
  - En développement actif
  - Contiendra de [petites améliorations en matière de performances, de qualité et de stabilité](#)
  - Planifiée en tant que version de support à long terme (LTS)
  - Actuellement planifiée pour décembre 2019
- **EF Core « vNext »**
  - Prochaine version majeure d'EF Core à être fournie avec .NET 5
  - La planification de cette version n'a pas encore été définie et aucune fonctionnalité n'a été annoncée.

### Planification

La planification de la version d'EF Core est synchronisée avec la [planification de la version de .NET Core](#).

### Backlog

Le [jalon Backlog](#) dans notre suivi de problème contient les problèmes sur lesquels nous prévoyons de travailler un jour ou susceptibles selon nous d'être traités par la communauté. Les clients sont invités à envoyer leurs commentaires et leurs votes concernant ces problèmes. Les collaborateurs qui souhaitent travailler sur un de ces problèmes sont encouragés à lancer dans un premier temps une discussion sur la façon d'approcher un problème.

Il n'est jamais garanti que nous allons travailler sur une fonctionnalité donnée dans une version spécifique d'EF Core. Comme dans tous les projets de logiciels, les priorités, les calendriers de publication et les ressources disponibles peuvent changer à tout moment. Mais si nous avons l'intention de résoudre un problème dans un laps de temps défini, nous lui attribuerons un jalon de version plutôt qu'un jalon de type backlog. Nous déplaçons régulièrement des problèmes entre les jalons backlog et de mise en production dans le cadre de notre [processus de planification de mise en production](#).

Nous fermons généralement un problème si nous ne prévoyons pas de le résoudre. Mais nous pouvons reconSIDéRer un problème que nous avions fermé si nous obtenons de nouvelles informations à ce sujet.

### Processus de planning des versions

On nous demande souvent comment nous choisissons les fonctionnalités qui seront intégrées dans une version donnée. Notre backlog ne se traduit pas du tout automatiquement en plannings de versions. De plus, la présence d'une fonctionnalité dans EF6 ne signifie pas automatiquement qu'elle doit être implémentée dans EF Core.

Il est difficile de détailler l'ensemble du processus que nous mettons en place pour planifier une mise en production. Une grande partie de ce processus se résume à étudier les fonctionnalités, les opportunités et les priorités spécifiques, et le processus lui-même évolue également avec chaque nouvelle version. Mais il est relativement facile de récapituler les questions courantes auxquelles nous essayons de répondre quand nous choisissons les éléments sur lesquels nous allons travailler :

1. **Combien de développeurs vont utiliser la fonctionnalité selon nous et dans quelle mesure va-t-elle améliorer leurs applications ou leur expérience ?** Pour répondre à cette question, nous recueillons les commentaires provenant de nombreuses sources (les commentaires et votes sur les problèmes en sont une).
2. **Quelles sont les solutions de contournement possibles si nous n'implémentons pas encore cette fonctionnalité ?** Par exemple, de nombreux développeurs peuvent mapper une table de jointure afin de pallier l'absence de prise en charge native du mappage plusieurs-à-plusieurs. Évidemment, tous les développeurs ne peuvent pas le faire, mais beaucoup en sont capables, ce qui constitue un facteur important dans notre décision.
3. **L'implémentation de cette fonctionnalité fait-elle évoluer l'architecture d'EF Core de telle manière que l'implémentation d'autres fonctionnalités s'en voit facilitée ou plus probable ?** Nous avons tendance à privilégier les fonctionnalités qui agissent comme blocs de construction pour d'autres fonctionnalités. Par exemple, les entités de sac de propriétés peuvent faciliter le passage à la prise en charge du mappage plusieurs-à-plusieurs et les constructeurs d'entité permettent la prise en charge de notre chargement différé.
4. **La fonctionnalité est-elle un point d'extensibilité ?** Nous avons tendance aussi à favoriser les points d'extensibilité au détriment des fonctionnalités standard, car ils permettent aux développeurs de raccorder leurs propres comportements et d'obtenir ainsi une compensation pour certaines fonctionnalités manquantes.
5. **Quelle est la synergie de la fonctionnalité quand elle est utilisée conjointement avec d'autres produits ?** Nous favorisons les fonctionnalités qui permettent ou améliorent considérablement l'utilisation d'EF Core avec d'autres produits, comme .NET Core, la dernière version de Visual Studio, Microsoft Azure et ainsi de suite.
6. **Quelles sont les compétences des personnes disponibles pour travailler sur une fonctionnalité, et comment exploiter au mieux ces ressources ?** Chaque membre de l'équipe EF et nos collaborateurs de la Communauté ont différents niveaux d'expérience dans différents domaines, et nous devons donc planifier en conséquence. Même si nous souhaitons faire appel en même temps à toutes ces ressources pour travailler sur une fonctionnalité spécifique telle que les traductions GroupBy ou le mappage plusieurs-à-plusieurs, ce ne serait pas pratique.

Comme mentionné précédemment, le processus évolue à chaque version. À l'avenir, nous essaierons d'ajouter davantage d'opportunités pour les membres de la communauté de fournir des entrées dans nos plans de mise en production. Par exemple, nous aimerais faciliter le processus de révision de nos ébauches de fonctionnalités et du plan de mise en production lui-même.

# Nouvelles fonctionnalités d'Entity Framework Core 3.0

06/12/2019 • 13 minutes to read • [Edit Online](#)

La liste suivante présente les nouvelles fonctionnalités majeures d'EF Core 3.0.

En tant que version majeure, EF Core 3.0 contient également plusieurs [changements cassants](#), qui sont des améliorations d'API pouvant avoir un impact négatif sur les applications existantes.

## Révision de LINQ

LINQ permet d'écrire des requêtes de base de données dans le langage .NET de prédilection, en tirant parti de riches informations de type pour proposer IntelliSense et le contrôle de type au moment de la compilation. Toutefois, LINQ vous permet également d'écrire un nombre illimité de requêtes complexes contenant des expressions arbitraires (appels de méthode ou opérations). La gestion de toutes ces combinaisons est le principal défi des fournisseurs LINQ.

Dans EF Core 3.0, nous avons remanié notre fournisseur LINQ pour permettre la traduction d'un plus grand nombre de modèles de requête en SQL, en générant des requêtes efficaces dans des cas plus nombreux et en empêchant les requêtes inefficaces de passer inaperçues. Le nouveau fournisseur LINQ est la fondation qui nous permettra d'offrir de nouvelles capacités de requête et des améliorations de performances dans les versions futures, sans arrêter les applications et les fournisseurs de données existants.

### Évaluation restreinte du client

Le changement de conception le plus important porte sur la façon dont nous manipulons les expressions LINQ qui ne peuvent pas être converties en paramètres ou traduites en SQL.

Dans les versions précédentes, EF Core identifiait les parties d'une requête pouvant être traduites en SQL et exécutait le reste de la requête sur le client. Ce type d'exécution côté client est souhaitable dans certains cas, mais dans de nombreux autres cas, il peut entraîner des requêtes inefficaces.

Par exemple, si EF Core 2.2 ne pouvait pas traduire un prédictat en appel `Where()`, il exécutait une instruction SQL sans filtre, transférait toutes les lignes depuis la base de données, puis les filtrait dans la mémoire :

```
var specialCustomers = context.Customers
    .Where(c => c.Name.StartsWith(n) && IsSpecialCustomer(c));
```

Cela peut être acceptable si la base de données contient un petit nombre de lignes, mais peut entraîner des problèmes de performances significatifs, voire une défaillance de l'application si la base de données contient un grand nombre de lignes.

Dans EF Core 3.0, nous avons restreint l'évaluation du client à la seule projection au niveau supérieur (essentiellement le dernier appel à `Select()`). Lorsqu'EF Core 3.0 détecte des expressions qui ne peuvent pas être traduites ailleurs dans la requête, une exception runtime est levée.

Pour évaluer une condition de prédictat sur le client comme dans l'exemple précédent, les développeurs ont désormais besoin de basculer explicitement l'évaluation de la requête LINQ to Objects :

```
var specialCustomers = context.Customers
    .Where(c => c.Name.StartsWith(n))
    .AsEnumerable() // switches to LINQ to Objects
    .Where(c => IsSpecialCustomer(c));
```

Consultez la [documentation des changements cassants](#) pour en savoir plus sur la façon dont cela peut affecter des applications existantes.

### Une seule instruction SQL par requête LINQ

Un autre aspect de la conception qui a beaucoup changé dans la version 3.0 est que nous créons maintenant toujours une seule instruction SQL par requête LINQ. Dans les versions précédentes, nous avions l'habitude de créer plusieurs instructions SQL dans certains cas, à savoir des appels `Include()` traduits sur les propriétés de navigation de collection et des requêtes traduites qui suivaient certains modèles avec des sous-requêtes. Même si c'était pratique dans certains cas, et que, pour `Include()` cela aidait même à éviter d'envoyer des données redondantes sur le réseau, l'implémentation était complexe et des comportements extrêmement inefficaces en résultait (requêtes N+1). Dans certaines situations, les données retournées sur plusieurs requêtes étaient potentiellement incohérentes.

Similairement à l'évaluation du client, si EF Core 3.0 ne peut pas traduire une requête LINQ en instruction SQL unique, une exception runtime est levée. Cependant, nous avons rendu EF Core capable de traduire beaucoup des modèles courants utilisés pour générer plusieurs requêtes relatives à une requête unique avec des jointures.

## Prise en charge de Cosmos DB

Le fournisseur Cosmos DB pour EF Core permet aux développeurs qui maîtrisent le modèle de programmation d'EF de cibler facilement Azure Cosmos DB comme base de données d'application. L'objectif est de rendre certains des avantages de Cosmos DB (par exemple la distribution mondiale, la disponibilité « Always On », la scalabilité élastique et la faible latence) encore plus accessibles aux développeurs .NET. Le fournisseur propose la plupart des fonctionnalités d'EF Core, comme le suivi automatique des modifications, les conversions de valeurs et LINQ, sur l'API SQL dans Cosmos DB.

Consultez la [documentation fournisseur Cosmos DB](#) pour plus de détails.

## Prise en charge de C# 8.0

EF Core 3.0 bénéficie de quelques-unes des [nouvelles fonctionnalités de C# 8.0](#) :

### Flux asynchrones

Les résultats des requêtes asynchrones sont maintenant exposés à l'aide de la nouvelle interface `IAsyncEnumerable<T>` standard et peuvent être consommés avec `await foreach`.

```
var orders =
    from o in context.Orders
    where o.Status == OrderStatus.Pending
    select o;

await foreach(var o in orders.AsAsyncEnumerable())
{
    Process(o);
}
```

Consultez la rubrique [Flux asynchrones dans la documentation C#](#) pour plus de détails.

### Types références Nullables

Lorsque cette nouvelle fonctionnalité est activée dans votre code, EF Core examine la possibilité de valeur Null des

propriétés de type référence et l'applique aux colonnes et aux relations correspondantes dans la base de données : les propriétés de types de références sans possibilité de valeur Null sont traitées comme si elles avaient l'attribut d'annotation de données `[Required]`.

Par exemple, dans la classe suivante, les propriétés marquées comme étant de type `string?` sont configurées comme facultatives, tandis que `string` est configuré comme obligatoire :

```
public class Customer
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string? MiddleName { get; set; }
}
```

Consultez [Utilisation de types références avec possibilité de valeur Null](#) dans la documentation EF Core pour plus de détails.

## Interception d'opérations de base de données

La nouvelle API d'interception dans EF Core 3.0 permet de fournir une logique personnalisée à appeler automatiquement à chaque fois que des opérations de base de données de bas niveau sont effectuées dans le cadre du fonctionnement normal d'EF Core. Par exemple, lors de l'ouverture de connexions, de la validation de transactions ou de l'exécution de commandes.

Similairement aux fonctionnalités d'interception qui existaient dans EF 6, les intercepteurs vous permettent d'intercepter des opérations avant ou après leur exécution. Lorsque vous les interceptez avant leur exécution, vous êtes autorisé à contourner l'exécution et à fournir d'autres résultats provenant de la logique d'interception.

Par exemple, pour manipuler un texte de commande, vous pouvez créer un `IDbCommandInterceptor` :

```
public class HintCommandInterceptor : DbCommandInterceptor
{
    public override InterceptionResult ReaderExecuting(
        DbCommand command,
        CommandEventData eventData,
        InterceptionResult result)
    {
        // Manipulate the command text, etc. here...
        command.CommandText += " OPTION (OPTIMIZE FOR UNKNOWN)";
        return result;
    }
}
```

Et l'enregistrer avec votre `DbContext` :

```
services.AddDbContext(b => b
    .UseSqlServer(connectionString)
    .AddInterceptors(new HintCommandInterceptor()));
```

## Ingénierie à rebours des vues de base de données

Les types de requêtes qui représentent des données pouvant être lues à partir de la base de données, mais ne pouvant pas être mises à jour, ont été renommés [types d'entités sans clé](#). Comme ils sont très bien adaptés au mappage d'affichages de bases de données dans la plupart des scénarios, EF Core crée désormais automatiquement des types d'entités sans clé lors de l'ingénierie à rebours d'affichages de base de données.

Par exemple, avec l'[outil dotnet ef command-line](#), vous pouvez saisir :

```
dotnet ef dbcontext scaffold "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer
```

L'outil génère désormais automatiquement des types de vues et de tables sans clés :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)  
{  
    modelBuilder.Entity<Names>(entity =>  
    {  
        entity.HasKey();  
        entity.ToView("Names");  
    });  
  
    modelBuilder.Entity<Things>(entity =>  
    {  
        entity.HasKey();  
    });  
}
```

## Les entités dépendantes qui partagent la table avec le principal sont maintenant facultatives

À partir d'EF Core 3.0, si `OrderDetails` est détenu par `Order` ou explicitement mappé à la même table, il est possible d'ajouter un `Order` sans `OrderDetails` et toutes les propriétés `OrderDetails` à l'exception de la clé primaire sont mappées à des colonnes de type nullable.

Pendant l'interrogation, EF Core définit `OrderDetails` sur `null` si une de ses propriétés obligatoires n'a pas de valeur, ou s'il n'a pas de propriété obligatoire en plus de la clé primaire et que toutes les propriétés sont `null`.

```
public class Order  
{  
    public int Id { get; set; }  
    public int CustomerId { get; set; }  
    public OrderDetails Details { get; set; }  
}  
  
[Owned]  
public class OrderDetails  
{  
    public int Id { get; set; }  
    public string ShippingAddress { get; set; }  
}
```

## EF 6.3 sur .NET Core

Ce n'est pas véritablement une fonctionnalité d'EF Core 3.0, mais nous pensons qu'elle est importante pour beaucoup de vos clients actuels.

Nous sommes conscients du fait que de nombreuses applications utilisent d'anciennes versions d'EF et qu'une migration vers EF Core dans l'unique objectif de tirer parti de .NET Core représente un effort considérable. C'est pourquoi nous avons décidé de déplacer la version la plus récente d'EF 6 de façon à ce qu'elle s'exécute sur .NET Core 3.0.

Pour plus d'informations, consultez les [nouveautés de la base de données dans EF 6](#).

## Fonctionnalités différées

Certaines fonctionnalités prévues à l'origine pour EF Core 3.0 ont été différées à des versions futures :

- Capacité d'ignorer des parties d'un modèle lors de migrations, référence de suivi [#2725](#).
- Entités des conteneurs de propriétés, suivies comme deux problèmes distincts : [#9914](#) pour les entités de type partagé et [#13610](#) pour le support du mappage de propriété.

# Dernières modifications incluses dans EF Core 3,0

10/01/2020 • 86 minutes to read • [Edit Online](#)

Les modifications d'API et de comportement suivantes peuvent bloquer les applications existantes lors de leur mise à niveau vers 3.0.0. Les changements qui, selon nous, auront une incidence uniquement sur les fournisseurs de base de données sont documentés dans [Changements ayant un impact sur les fournisseurs](#).

## Récapitulatif

| MODIFICATION CRITIQUE   | IMPACT |
|---|--------|
| Les requêtes LINQ ne sont plus évaluées sur le client                                 | Élevée |
| EF Core 3.0 cible .NET Standard 2.1 plutôt que .NET Standard 2.0                      | Élevée |
| L'outil en ligne de commande EF Core, dotnet ef, ne fait plus partie du SDK .NET Core | Élevée |
| DetectChanges honore les valeurs de clés générées par le magasin                      | Élevée |
| FromSql, ExecuteSql et ExecuteSqlAsync ont été renommés                               | Élevée |
| Les types de requêtes sont regroupés avec les types d'entités                         | Élevée |
| Entity Framework Core ne fait plus partie du framework partagé ASP.NET Core           | Moyen  |
| Les suppressions en cascade se produisent désormais immédiatement par défaut          | Moyen  |
| Le chargement hâtif des entités associées se produit désormais dans une seule requête | Moyen  |
| La sémantique de DeleteBehavior.Restrict est désormais plus propre                    | Moyen  |
| L'API de configuration pour les relations de type détenu a changé                     | Moyen  |
| Chaque propriété utilise la génération indépendante de clé de type entier en mémoire  | Moyen  |
| Les requêtes sans suivi ne procèdent plus à la résolution de l'identité               | Moyen  |
| Changements apportés à l'API de métadonnées   | Moyen  |
| Modifications de l'API de métadonnées spécifiques au fournisseur                      | Moyen  |

| MODIFICATION CRITIQUE   | IMPACT |
|---|--------|
| UseRowNumberForPaging a été supprimé  | Moyen  |
| La méthode FromSql quand elle est utilisée avec une procédure stockée ne peut pas être composée                       | Moyen  |
| Les méthodes FromSql peuvent uniquement être spécifiées sur les racines de requête                                    | Faible |
| ~~L'exécution de requêtes est enregistrée au niveau du débogage ~~Rétabli   | Faible |
| Les valeurs de clés temporaires ne sont plus définies sur les instances d'entités                                     | Faible |
| Les entités dépendantes qui partagent la table avec le principal sont maintenant facultatives                         | Faible |
| Toutes les entités qui partagent une table avec une colonne de jeton de concurrence doivent la mapper à une propriété | Faible |
| Les entités détenues ne peuvent pas être interrogées sans le propriétaire à l'aide d'une requête de suivi             | Faible |
| Les propriétés héritées de types non mappés sont maintenant mappées à une seule colonne pour tous les types dérivés   | Faible |
| La convention de propriété de clé étrangère ne correspond plus au nom de la propriété principale                      | Faible |
| La connexion de base de données est maintenant fermée si elle n'est plus utilisée avant la fin de TransactionScope    | Faible |
| Les champs de stockage sont utilisés par défaut   | Faible |
| Erreur si plusieurs champs de stockage compatibles sont détectés  | Faible |
| Les noms de propriété de type « champ uniquement » doivent correspondre au nom du champ                               | Faible |
| AddDbContext/AddDbContextPool n'appelle plus AddLogging et AddMemoryCache   | Faible |
| AddEntityFramework * ajoute IMemoryCache avec une limite de taille  | Faible |
| DbContext.Entry effectue maintenant un DetectChanges local  | Faible |
| Les clés de tableaux d'octets et de chaînes ne sont pas générées par client par défaut                                | Faible |
| ILoggerFactory est désormais un service délimité  | Faible |

| MODIFICATION CRITIQUE  | IMPACT |
|--|--------|
| Les proxys à chargement différé ne partent plus du principe que les propriétés de navigation sont entièrement chargées | Faible |
| La création excessive de fournisseurs de services internes est maintenant une erreur par défaut                        | Faible |
| Nouveau comportement de HasOne/HasMany appelé avec une chaîne unique   | Faible |
| Le type de retour Task pour plusieurs méthodes asynchrones a été remplacé par ValueTask                                | Faible |
| L'annotation Relational:TypeMapping est désormais simplement TypeMapping   | Faible |
| ToTable sur un type dérivé lève une exception  | Faible |
| EF Core n'envoie plus de pragma pour l'application de clé étrangère SQLite   | Faible |
| Microsoft.EntityFrameworkCore.Sqlite dépend désormais de SQLitePCLRaw.bundle_e_sqlite3                                 | Faible |
| Les valeurs GUID sont maintenant stockées au format TEXT sur SQLite  | Faible |
| Les valeurs char sont maintenant stockées au format TEXT sur SQLite  | Faible |
| Les ID de migration sont maintenant générés à l'aide du calendrier de la culture invariante                            | Faible |
| Les infos/métadonnées d'extension ont été supprimées de IDbContextOptionsExtension                                     | Faible |
| LogQueryPossibleExceptionWithAggregateOperator a été renommé   | Faible |
| Clarifier les noms de contrainte de clé étrangère dans l'API   | Faible |
| IRelationalDatabaseCreator.HasTables/HasTablesAsync ont été rendues publiques  | Faible |
| Microsoft.EntityFrameworkCore.Design est désormais un package DevelopmentDependency                                    | Faible |
| SQLitePCL.raw mis à jour vers la version 2.0.0   | Faible |
| NetTopologySuite mis à jour vers la version 2.0.0  | Faible |
| Microsoft. Data. SqlClient est utilisé à la place de System. Data. SqlClient   | Faible |

| MODIFICATION CRITIQUE  | IMPACT |
|--|--------|
| Plusieurs relations autoréférencées ambiguës doivent être configurées  | Faible |
| DbFunction. Schema étant null ou une chaîne vide, il est configuré pour être dans le schéma par défaut du modèle | Faible |

## Les requêtes LINQ ne sont plus évaluées sur le client

[Suivi de problème no 14935](#) Voir également problème no 12795

### Ancien comportement

Avant la version 3.0, quand EF Core ne pouvait pas convertir une expression faisant partie d'une requête en SQL ou en paramètre, elle évaluait automatiquement l'expression sur le client. Par défaut, l'évaluation sur le client d'expressions potentiellement coûteuses déclencheait uniquement un avertissement.

### Nouveau comportement

À compter de la version 3.0, EF Core autorise uniquement l'évaluation sur le client des expressions qui figurent dans la projection de niveau supérieur (le dernier appel `Select()` dans la requête). Quand des expressions d'une autre partie de la requête ne peuvent pas être converties en SQL ou en paramètre, une exception est levée.

### Pourquoi ?

L'évaluation automatique des requêtes sur le client permet à de nombreuses requêtes d'être exécutées même si certaines de leurs parties importantes ne peuvent pas être traduites. Ce comportement peut entraîner un comportement inattendu et potentiellement dangereux qui peut devenir évident uniquement en production. Par exemple, une condition dans un appel `Where()` qui ne peut pas être traduite peut provoquer le transfert de toutes les lignes de la table hors du serveur de base de données, et l'application du filtre sur le client. Cette situation peut facilement passer inaperçue si la table contient uniquement quelques lignes lors du développement, mais poser davantage de problèmes quand l'application passe en production, où la table peut contenir plusieurs millions de lignes. Les avertissements relatifs à l'évaluation sur le client étaient également trop faciles à ignorer pendant le développement.

De plus, l'évaluation automatique sur le client peut entraîner des problèmes selon lesquels l'amélioration de la traduction des requêtes pour des expressions spécifiques provoque un changement cassant involontaire entre les versions.

### Atténuations

Si une requête ne peut pas être entièrement traduite, réécrivez-la sous une forme qui peut être traduite ou utilisez `AsEnumerable()`, `ToList()` ou autre méthode similaire pour réimporter explicitement les données sur le client, où elles peuvent ensuite être traitées à l'aide de LINQ-to-Objects.

## EF Core 3.0 cible .NET Standard 2.1 plutôt que .NET Standard 2.0

[Suivi de problème no 15498](#)

### Ancien comportement

Avant la version 3.0, EF Core ciblait .NET Standard 2.0 et s'exécutait sur toutes les plateformes qui prennent en charge cette norme, y compris .NET Framework.

### Nouveau comportement

À partir de la version 3.0, EF Core cible .NET Standard 2.1 et s'exécute sur toutes les plateformes qui prennent en charge cette norme. Cela n'inclut pas .NET Framework.

## Pourquoi ?

Cela fait partie d'une décision stratégique dans les technologies .NET de concentrer l'énergie sur .NET Core et d'autres plateformes .NET modernes, telles que Xamarin.

## Atténuations

Envisagez de passer à une plateforme .NET moderne. Si ce n'est pas possible, continuez à utiliser EF Core 2.1 ou EF Core 2.2, qui prennent tous deux en charge .NET Framework.

## Entity Framework Core ne fait plus partie du framework partagé ASP.NET Core

[Annonces de suivi de problème n°325](#)

### Ancien comportement

Avant ASP.NET Core 3.0, quand vous ajoutiez une référence de package à `Microsoft.AspNetCore.App` ou `Microsoft.AspNetCore.All`, elle incluait EF Core et certains des fournisseurs de données EF Core tels que le fournisseur SQL Server.

### Nouveau comportement

À compter de la version 3.0, le framework partagé ASP.NET Core n'inclut ni EF Core, ni aucun fournisseur de données EF Core.

## Pourquoi ?

Avant ce changement, l'obtention d'EF Core nécessitait différentes étapes selon que l'application ciblait ASP.NET Core et SQL Server ou non. De plus, la mise à niveau d'ASP.NET Core forçait la mise à niveau d'EF Core et du fournisseur SQL Server, ce qui n'est pas toujours souhaitable.

Avec ce changement, l'expérience d'obtention d'EF Core est la même pour tous les fournisseurs, implémentations .NET prises en charge et types d'applications. Les développeurs peuvent désormais contrôler exactement quand EF Core et les fournisseurs de données EF Core sont mis à niveau.

## Atténuations

Pour utiliser EF Core dans une application ASP.NET Core 3.0 ou toute autre application prise en charge, ajoutez explicitement une référence de package au fournisseur de base de données EF Core que votre application utilisera.

## L'outil en ligne de commande EF Core, `dotnet ef`, ne fait plus partie du SDK .NET Core

[Suivi du problème n° 14016](#)

### Ancien comportement

Avant la version 3.0, l'outil `dotnet ef` était inclus dans le SDK .NET Core et prêt à l'emploi depuis la ligne de commande d'un projet sans nécessiter d'étapes supplémentaires.

### Nouveau comportement

À compter de la version 3.0, le SDK .NET n'inclut pas l'outil `dotnet ef`, donc vous pouvez explicitement l'installer pour pouvoir l'utiliser comme outil local ou global.

## Pourquoi ?

Ce changement nous permet de distribuer et mettre à jour `dotnet ef` sous forme d'outil CLI .NET normal sur NuGet, ce qui est cohérent avec le fait qu'EF Core 3.0 est également toujours distribué sous forme de package NuGet.

## Atténuations

Pour pouvoir gérer des migrations ou générer un `DbContext`, installez `dotnet-ef` comme outil général :

```
$ dotnet tool install --global dotnet-ef
```

Vous pouvez également obtenir un outil local quand vous restaurez les dépendances d'un projet qui le déclare en tant que dépendance d'outil à l'aide d'un [fichier manifeste d'outil](#).

### FromSql, ExecuteSql et ExecuteSqlAsync ont été renommés

[Suivi du problème no 10996](#)

#### Ancien comportement

Avant EF Core 3.0, ces noms de méthode étaient surchargés pour être utilisés avec une chaîne normale ou une chaîne devant être interpolée dans SQL et dans des paramètres.

#### Nouveau comportement

À compter d'EF Core 3.0, utilisez `FromSqlRaw`, `ExecuteSqlRaw` et `ExecuteSqlRawAsync` pour créer une requête paramétrable, où les paramètres sont passés séparément à partir de la chaîne de requête. Par exemple :

```
context.Products.FromSqlRaw(  
    "SELECT * FROM Products WHERE Name = {0}",  
    product.Name);
```

Utilisez `FromSqlInterpolated`, `ExecuteSqlInterpolated` et `ExecuteSqlInterpolatedAsync` pour créer une requête paramétrable, où les paramètres sont passés dans le cadre d'une chaîne de requête interpolée. Par exemple :

```
context.Products.FromSqlInterpolated(  
    $"SELECT * FROM Products WHERE Name = {product.Name}");
```

Notez que les deux requêtes ci-dessus produisent le même SQL paramétrable avec les mêmes paramètres SQL.

#### Pourquoi ?

Les surcharges de méthode comme celle-ci rendent très facile un appel accidentel à la méthode de chaîne brute quand l'intention est d'appeler la méthode de chaîne interpolée, et inversement. De ce fait, les requêtes ne sont plus paramétrables alors qu'elles devraient l'être.

#### Atténuations

Utilisez plutôt les nouveaux noms de méthode.

**La méthode FromSql quand elle est utilisée avec une procédure stockée ne peut pas être composée**  
[#15392](#) du problème de suivi

#### Ancien comportement

Avant le EF Core 3.0, la méthode `FromSql` a tenté de détecter si le SQL passé peut être composé. Il a fait l'évaluation du client lorsque le SQL n'était pas composable comme une procédure stockée. La requête suivante a fonctionné en exécutant la procédure stockée sur le serveur et en procédant à l'opération `FirstOrDefault` côté client.

```
context.Products.FromSqlRaw("[dbo].[Ten Most Expensive Products]").FirstOrDefault();
```

#### Nouveau comportement

À compter de EF Core 3,0, EF Core n'essaiera pas d'analyser le SQL. Par conséquent, si vous effectuez une composition après FromSqlRaw/FromSqlInterpolated, EF Core compose le SQL en provoquant une sous-requête. Par conséquent, si vous utilisez une procédure stockée avec la composition, vous obtiendrez une exception pour la syntaxe SQL non valide.

## Pourquoi ?

EF Core 3,0 ne prend pas en charge l'évaluation automatique du client, car il s'agit d'une erreur, comme expliqué [ici](#).

## Atténuation

Si vous utilisez une procédure stockée dans FromSqlRaw/FromSqlInterpolated, vous savez qu'elle ne peut pas être composée. Vous pouvez donc ajouter **AsEnumerable/AsAsyncEnumerable** juste après l'appel de la méthode FromSql pour éviter toute composition côté serveur.

```
context.Products.FromSqlRaw("[dbo].[Ten Most Expensive Products]").AsEnumerable().FirstOrDefault();
```

## Les méthodes FromSql peuvent uniquement être spécifiées sur les racines de requête

[Suivi de problème no 15704](#)

### Ancien comportement

Avant EF Core 3,0, la méthode `FromSql` pouvant être spécifiée n'importe où dans la requête.

### Nouveau comportement

À partir d'EF Core 3,0, les nouvelles méthodes `FromSqlRaw` et `FromSqlInterpolated` (qui remplacent `FromSql`) peuvent être spécifiées uniquement sur les racines de requête, par exemple, directement sur le `DbSet<>`. Une erreur de compilation survient si vous tentez de les spécifier à un autre emplacement.

## Pourquoi ?

La spécification de `FromSql` n'importe où autre que sur un `DbSet` n'avait aucune signification ni valeur ajoutée et pouvait entraîner une ambiguïté dans certains scénarios.

## Atténuations

Les appels `FromSql` doivent être déplacés pour être directement sur le `DbSet` auquel ils s'appliquent.

## Les requêtes sans suivi ne procèdent plus à la résolution de l'identité

[Suivi de problème no 13518](#)

### Ancien comportement

Avant EF Core 3,0, la même instance d'entité était utilisée pour chaque occurrence d'une entité avec un type et un ID donnés. Cela correspond au comportement des requêtes de suivi. Par exemple, cette requête :

```
var results = context.Products.Include(e => e.Category).AsNoTracking().ToList();
```

retournait la même instance `Category` pour chaque `Product` associé à la catégorie donnée.

### Nouveau comportement

À compter de EF Core 3,0, différentes instances d'entité sont créées lorsqu'une entité avec un type et un ID donnés est rencontrée à différents emplacements dans le graphe retourné. Par exemple, la requête ci-dessus retourne à présent une nouvelle instance `Category` pour chaque `Product` même si deux produits sont associés à la même catégorie.

## Pourquoi ?

La résolution de l'identité (autrement dit, le fait de déterminer qu'une entité a les mêmes type et ID qu'une entité précédemment rencontrée) améliore les performances et la surcharge de la mémoire. Cela agit généralement à l'opposé de la raison pour laquelle aucune requête de suivi n'est utilisée en premier lieu. En outre, même si la résolution de l'identité peut parfois être utile, elle n'est pas nécessaire si les entités doivent être serialisées et envoyées à un client, ce qui est courant pour les requêtes sans suivi.

## Atténuations

Utilisez une requête de suivi si la résolution de l'identité est requise.

### L'exécution de requêtes est enregistrée au niveau du débogage rétabli

#### Suivi de problème n°14523

Nous avons rétabli ce changement car la nouvelle configuration dans EF Core 3.0 permet la spécification du niveau d'enregistrement d'un événement par l'application. Par exemple, pour basculer l'enregistrement de SQL vers `Debug`, configurez explicitement le niveau dans `OnConfiguring` ou `AddDbContext` :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder
        .UseSqlServer(connectionString)
        .ConfigureWarnings(c => c.Log((RelationalEventId.CommandExecuting, LogLevel.Debug)));
```

### Les valeurs de clés temporaires ne sont plus définies sur les instances d'entités

#### Suivi de problème n°12378

## Ancien comportement

Avant EF Core 3.0, des valeurs temporaires étaient affectées à toutes les propriétés de clé qui auraient plus tard une valeur réelle générée par la base de données. Généralement, ces valeurs temporaires étaient de grands nombres négatifs.

## Nouveau comportement

À compter de la version 3.0, EF Core stocke la valeur de clé temporaire dans le cadre des informations de suivi de l'entité, et laisse la propriété de clé proprement dite inchangée.

## Pourquoi ?

Ce changement a été apporté afin d'empêcher que les valeurs de clés temporaires ne deviennent à tort permanentes quand une entité qui a été suivie par une instance `DbContext` est déplacée vers une autre instance `DbContext`.

## Atténuations

Les applications qui affectent des valeurs de clés primaires sur des clés étrangères afin de former des associations entre des entités peuvent dépendre de l'ancien comportement si les clés primaires sont générées par le magasin et appartiennent à des entités à l'état `Added`. Vous pouvez éviter cela en :

- N'utilisant pas de clés générées par le magasin.
- Définissant des propriétés de navigation afin d'établir des relations au lieu de définir des valeurs de clés étrangères.
- Obtenant les valeurs de clés temporaires réelles à partir des informations de suivi de l'entité. Par exemple, `context.Entry(blog).Property(e => e.Id).CurrentValue` retournera la valeur temporaire même si la propriété `blog.Id` elle-même n'a pas été définie.

## DetectChanges honore les valeurs de clés générées par le magasin

## Suivi de problème n°14616

### Ancien comportement

Avant EF Core 3.0, une entité non suivie détectée par `DetectChanges` était suivie à l'état `Added` et insérée en tant que nouvelle ligne quand `SaveChanges` était appelée.

### Nouveau comportement

À compter d'EF Core 3.0, si une entité utilise des valeurs de clés générées et qu'une valeur de clé est définie, alors l'entité est suivie à l'état `Modified`. Cela signifie qu'une ligne pour l'entité est supposée exister et qu'elle sera mise à jour lors de l'appel à `SaveChanges`. Si la valeur de clé n'est pas définie, ou si le type d'entité n'utilise pas de clés générées, alors la nouvelle entité sera quand même suivie comme `Added`, comme dans les versions précédentes.

### Pourquoi ?

Ce changement a été apporté afin de simplifier l'utilisation des graphes d'entités déconnectées lors de l'utilisation de clés générées par le magasin.

### Atténuations

Ce changement peut casser une application si un type d'entité est configuré pour utiliser des clés générées, mais que les valeurs de clés sont définies explicitement pour les nouvelles instances. La solution consiste à configurer explicitement les propriétés de clés de façon à ne pas utiliser des valeurs générées. Par exemple, avec l'API Fluent :

```
modelBuilder
    .Entity<Blog>()
    .Property(e => e.Id)
    .ValueGeneratedNever();
```

Ou avec des annotations de données :

```
[DatabaseGenerated(DatabaseGeneratedOption.None)]
public string Id { get; set; }
```

### Les suppressions en cascade se produisent désormais immédiatement par défaut

## Suivi de problème n°10114

### Ancien comportement

Avant la version 3.0, les actions en cascade appliquées par EF Core (suppression d'entités dépendantes quand un principal requis est supprimé ou quand la relation à un principal requis est interrompue) ne se produisaient qu'une fois que `SaveChanges` était appelée.

### Nouveau comportement

À compter de la version 3.0, EF Core applique les actions en cascade dès que la condition de déclenchement est détectée. Par exemple, si vous appelez `context.Remove()` pour supprimer une entité principale, toutes les dépendances requises suivies associées seront également définies immédiatement sur `Deleted`.

### Pourquoi ?

Cette modification a été apportée pour améliorer l'expérience des scénarios de liaison de données et d'audit dans lesquels il est important de comprendre les entités qui seront supprimées *avant* l'appel de `SaveChanges`.

### Atténuations

Vous pouvez restaurer le comportement précédent par le biais des paramètres sur `context.ChangeTracker`. Par

exemple :

```
context.ChangeTracker.CascadeDeleteTiming = CascadeTiming.OnSaveChanges;
context.ChangeTracker.DeleteOrphansTiming = CascadeTiming.OnSaveChanges;
```

## Le chargement hâtif des entités associées se produit désormais dans une seule requête

[#18022 du problème de suivi](#)

### Ancien comportement

Avant le 3,0, le chargement des navigations de collections à l'aide d'opérateurs `Include` entraînait la génération de plusieurs requêtes sur une base de données relationnelle, une pour chaque type d'entité associée.

### Nouveau comportement

À partir de 3,0, EF Core génère une requête unique avec des JOINTURES sur des bases de données relationnelles.

### Pourquoi ?

L'émission de plusieurs requêtes pour implémenter une seule requête LINQ a entraîné de nombreux problèmes, notamment des performances négatives lorsque plusieurs allers-retours de base de données étaient nécessaires et des problèmes de cohérence des données à mesure que chaque requête pouvait observer un état différent de la base de données.

### Atténuations

Bien qu'techniquement, il ne s'agit pas d'une modification avec rupture, elle peut avoir un impact considérable sur les performances de l'application lorsqu'une seule requête contient un grand nombre d'`Include` opérateur dans les navigations de collection. Pour plus d'informations et pour réécrire des requêtes de manière plus efficace, [consultez ce commentaire](#).

\*\*

## La sémantique de `DeleteBehavior.Restrict` est désormais plus propre

[Suivi de problème no 12661](#)

### Ancien comportement

Avant la version 3,0, `DeleteBehavior.Restrict` créait les clés étrangères dans la base de données avec la sémantique `Restrict`, et il modifiait les corrections internes d'une manière non évidente.

### Nouveau comportement

À compter de la version 3,0, `DeleteBehavior.Restrict` garantit la création des clés étrangères avec la sémantique `Restrict` (autrement dit, sans levée de cascades lors d'une violation de contrainte), sans impact sur les corrections internes EF.

### Pourquoi ?

Ce changement a été apporté pour améliorer l'expérience et permettre une utilisation intuitive de `DeleteBehavior`, sans effets secondaires inattendus.

### Atténuations

Vous pouvez restaurer le comportement précédent par le biais de `DeleteBehavior.ClientNoAction`.

## Les types de requêtes sont regroupés avec les types d'entités

[Suivi de problème n°14194](#)

### Ancien comportement

Avant EF Core 3.0, les [types de requêtes](#) étaient un moyen d'interroger des données qui ne définissait pas de clé primaire de façon structurée. Autrement dit, on utilisait un type de requête pour mapper des types d'entités sans clés (le plus souvent à partir d'une vue, mais aussi éventuellement à partir d'une table), alors qu'on utilisait un type d'entité normal quand une clé était disponible (le plus souvent à partir d'une table, mais aussi éventuellement à partir d'une vue).

## Nouveau comportement

Un type de requête devient maintenant simplement un type d'entité sans clé primaire. Les types d'entités sans clé ont les mêmes fonctionnalités que les types de requêtes dans les versions précédentes.

## Pourquoi ?

Ce changement a été apporté afin de réduire la confusion concernant l'objectif des types de requêtes. Plus précisément, il s'agit de types d'entités sans clé et sont intrinsèquement en lecture seule pour cette raison, mais vous ne devez pas les utiliser au seul motif qu'un type d'entité doit être en lecture seule. De même, ils sont souvent mappés à des vues, mais c'est uniquement car les vues définissent rarement des clés.

## Atténuations

Les parties suivantes de l'API sont désormais obsolètes :

- `ModelBuilder.Query<>()` - Au lieu de cela, vous devez appeler `ModelBuilder.Entity<>().HasNoKey()` pour marquer un type d'entité comme n'ayant pas de clé. Cela ne serait toujours pas configuré par convention, afin d'éviter une configuration incorrecte quand une clé primaire est attendue mais ne correspond pas à la convention.
- `DbQuery<>` - Utilisez plutôt `DbSet<>`.
- `DbContext.Query<>()` - Utilisez plutôt `DbContext.Set<>()`.

## L'API de configuration pour les relations de type détenu a changé

[Suivi de problème n°12444](#) [Suivi de problème n°9148](#) [Suivi de problème n°14153](#)

## Ancien comportement

Avant EF Core 3.0, la configuration de la relation détenue était effectuée directement après l'appel à `OwnsOne` ou `OwnsMany`.

## Nouveau comportement

À compter d'EF Core 3.0, il existe une API Fluent afin de configurer une propriété de navigation pour le propriétaire à l'aide de `WithOwner()`. Par exemple :

```
modelBuilder.Entity<Order>.OwnsOne(e => e.Details).WithOwner(e => e.Order);
```

La configuration liée à la relation entre le propriétaire et le détenu doit maintenant être chaînée après `WithOwner()` tout comme les autres relations. En revanche, la configuration du type détenu lui-même serait toujours chaînée après `OwnsOne()/OwnsMany()`. Par exemple :

```

modelBuilder.Entity<Order>.OwnsOne(e => e.Details, eb =>
{
    eb.WithOwner()
        .HasForeignKey(e => e.AlternateId)
        .HasConstraintName("FK_OrderDetails");

    eb.ToTable("OrderDetails");
    eb.HasKey(e => e.AlternateId);
    eb.HasIndex(e => e.Id);

    eb.HasOne(e => e.Customer).WithOne();

    eb.HasData(
        new OrderDetails
    {
        AlternateId = 1,
        Id = -1
    });
});

```

De plus, l'appel à `Entity()`, `HasOne()` ou `Set()` avec un type détenu comme cible lève désormais une exception.

## Pourquoi ?

Ce changement a été apporté afin de créer une distinction plus claire entre la configuration du type détenu lui-même et la *relation au type détenu*. Cela lève ainsi toute ambiguïté et toute confusion autour des méthodes telles que `HasForeignKey`.

## Atténuations

Changez la configuration des relations de type détenu de façon à utiliser la nouvelle surface d'API, comme indiqué dans l'exemple ci-dessus.

## Les entités dépendantes qui partagent la table avec le principal sont maintenant facultatives

[Suivi du problème no 9005](#)

### Ancien comportement

Considérez le modèle suivant :

```

public class Order
{
    public int Id { get; set; }
    public int CustomerId { get; set; }
    public OrderDetails Details { get; set; }
}

public class OrderDetails
{
    public int Id { get; set; }
    public string ShippingAddress { get; set; }
}

```

Avant EF Core 3.0, si `OrderDetails` est détenu par `Order` ou explicitement mappé à la même table, une instance `OrderDetails` est toujours nécessaire pour ajouter un nouveau `Order`.

### Nouveau comportement

À partir de la version 3.0, EF Core permet d'ajouter un `Order` sans `OrderDetails` et mappe toutes les propriétés de `OrderDetails` à l'exception de la clé primaire aux colonnes de type nullable. Pendant l'interrogation, EF Core définit `OrderDetails` sur `null` si une de ses propriétés obligatoires n'a pas de valeur, ou s'il n'a pas de propriété

obligatoire en plus de la clé primaire et que toutes les propriétés sont `null`.

## Atténuations

Si votre modèle a une table qui partage des dépendances avec toutes les colonnes facultatives, mais que la navigation qui pointe vers lui n'est pas censée être `null`, l'application doit être modifiée pour gérer les situations où la navigation est `null`. Si ce n'est pas possible, une propriété obligatoire doit être ajoutée au type d'entité ou au moins une propriété doit avoir une valeur non-`null`.

**Toutes les entités qui partagent une table avec une colonne de jeton de concurrence doivent la mapper à une propriété**

[Suivi du problème no 14154](#)

## Ancien comportement

Considérez le modèle suivant :

```
public class Order
{
    public int Id { get; set; }
    public int CustomerId { get; set; }
    public byte[] Version { get; set; }
    public OrderDetails Details { get; set; }
}

public class OrderDetails
{
    public int Id { get; set; }
    public string ShippingAddress { get; set; }
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Order>()
        .Property(o => o.Version).IsRowVersion().HasColumnName("Version");
}
```

Avant EF Core 3.0, si `OrderDetails` est détenu par `Order` ou explicitement mappé à la même table, la mise à jour de `OrderDetails` seulement ne met pas à jour la valeur de `Version` sur le client et la prochaine mise à jour échoue.

## Nouveau comportement

À compter de la version 3.0, EF Core propage la nouvelle valeur `version` sur `Order` s'il est propriétaire de `OrderDetails`. Sinon, une exception est levée pendant la validation de modèle.

## Pourquoi ?

Ce changement a été effectué pour éviter la préemption de la valeur du jeton de concurrence quand une seule des entités mappées à la même table est mise à jour.

## Atténuations

Toutes les entités partageant la table doivent inclure une propriété mappée à la colonne de jeton de concurrence. Vous pouvez en créer une dans un état fantôme :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<OrderDetails>()
        .Property<byte[]>("Version").IsRowVersion().HasColumnName("Version");
}
```

## Les entités détenues ne peuvent pas être interrogées sans le propriétaire à l'aide d'une requête de suivi

[#18876 du problème de suivi](#)

### Ancien comportement

Avant le EF Core 3,0, les entités appartenant peuvent être interrogées comme n'importe quelle autre navigation.

```
context.People.Select(p => p.Address);
```

### Nouveau comportement

À partir de 3,0, EF Core lève une exception si une requête de suivi projette une entité détenue sans le propriétaire.

### Pourquoi ?

Les entités détenues ne peuvent pas être manipulées sans le propriétaire, donc dans la grande majorité des cas, l'interrogation de cette façon est une erreur.

### Atténuations

Si l'entité possédée doit être suivie pour être modifiée de quelque façon que ce soit ultérieurement, le propriétaire doit être inclus dans la requête.

Sinon, ajoutez un appel `AsNoTracking()` :

```
context.People.Select(p => p.Address).AsNoTracking();
```

## Les propriétés héritées de types non mappés sont maintenant mappées à une seule colonne pour tous les types dérivés

[Suivi du problème no 13998](#)

### Ancien comportement

Considérez le modèle suivant :

```

public abstract class EntityBase
{
    public int Id { get; set; }
}

public abstract class OrderBase : EntityBase
{
    public int ShippingAddress { get; set; }
}

public class BulkOrder : OrderBase
{
}

public class Order : OrderBase
{
}

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Ignore<OrderBase>();
    modelBuilder.Entity<EntityBase>();
    modelBuilder.Entity<BulkOrder>();
    modelBuilder.Entity<Order>();
}

```

Avant EF Core 3.0, la propriété `ShippingAddress` est mappée à des colonnes distinctes pour `BulkOrder` et `Order` par défaut.

## Nouveau comportement

À compter de la version 3.0, EF Core crée uniquement une colonne pour `ShippingAddress`.

### Pourquoi ?

L'ancien comportement n'était pas attendu.

### Atténuations

La propriété peut toujours être mappée explicitement à une colonne distincte sur les types dérivés :

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Ignore<OrderBase>();
    modelBuilder.Entity<EntityBase>();
    modelBuilder.Entity<BulkOrder>()
        .Property(o => o.ShippingAddress).HasColumnName("BulkShippingAddress");
    modelBuilder.Entity<Order>()
        .Property(o => o.ShippingAddress).HasColumnName("ShippingAddress");
}

```

**La convention de propriété de clé étrangère ne correspond plus au nom de la propriété principale**

[Suivi de problème n°13274](#)

### Ancien comportement

Considérez le modèle suivant :

```

public class Customer
{
    public int CustomerId { get; set; }
    public ICollection<Order> Orders { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public int CustomerId { get; set; }
}

```

Avant EF Core 3.0, la propriété `CustomerId` était utilisée pour la clé étrangère par convention. Toutefois, si `Order` est un type détenu, cela fait également de `CustomerId` la clé primaire, ce qui ne correspond généralement pas aux attentes.

### Nouveau comportement

À compter de la version 3.0, EF Core ne tente pas d'utiliser des propriétés pour les clés étrangères par convention si elles ont le même nom que la propriété principale. Les modèles de nom du type de principal concaténé au nom de la propriété de principal, et de nom de navigation concaténé au nom de propriété de principal sont toujours mis en correspondance. Par exemple :

```

public class Customer
{
    public int Id { get; set; }
    public ICollection<Order> Orders { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public int CustomerId { get; set; }
}

```

```

public class Customer
{
    public int Id { get; set; }
    public ICollection<Order> Orders { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public int BuyerId { get; set; }
    public Customer Buyer { get; set; }
}

```

### Pourquoi ?

Ce changement a été apporté afin d'éviter de définir de manière erronée une propriété de clé primaire sur le type détenu.

### Atténuations

Si la propriété était censée être la clé étrangère, et par conséquent une partie de la clé primaire, configurez-la explicitement comme telle.

**La connexion de base de données est maintenant fermée si elle n'est plus utilisée avant la fin de TransactionScope**

## Suivi du problème no 14218

### Ancien comportement

Avant EF Core 3.0, si le contexte ouvre la connexion à l'intérieur d'un `TransactionScope`, la connexion reste ouverte quand le `TransactionScope` actuel est actif.

```
using (new TransactionScope())
{
    using (AdventureWorks context = new AdventureWorks())
    {
        context.ProductCategories.Add(new ProductCategory());
        context.SaveChanges();

        // Old behavior: Connection is still open at this point

        var categories = context.ProductCategories().ToList();
    }
}
```

### Nouveau comportement

À compter de la version 3.0, EF Core ferme la connexion dès qu'il ne l'utilise plus.

### Pourquoi ?

Ce changement permet d'utiliser plusieurs contextes dans le même `TransactionScope`. Le nouveau comportement correspond également à EF6.

### Atténuations

Si la connexion doit rester ouverte, un appel explicite à `openConnection()` garantit qu'EF Core ne la ferme pas prématurément :

```
using (new TransactionScope())
{
    using (AdventureWorks context = new AdventureWorks())
    {
        context.Database.OpenConnection();
        context.ProductCategories.Add(new ProductCategory());
        context.SaveChanges();

        var categories = context.ProductCategories().ToList();
        context.Database.CloseConnection();
    }
}
```

## Chaque propriété utilise la génération indépendante de clé de type entier en mémoire

### Suivi de problème n°6872

### Ancien comportement

Avant EF Core 3.0, un seul générateur de valeurs partagées était utilisé pour toutes les propriétés de clé de type entier en mémoire.

### Nouveau comportement

À compter d'EF Core 3.0, chaque propriété de clé de type entier obtient son propre générateur de valeur lors de l'utilisation de la base de données en mémoire. De plus, si la base de données est supprimée, la génération de clé est réinitialisée pour toutes les tables.

## Pourquoi ?

Ce changement a été apporté afin d'aligner plus étroitement la génération de clé en mémoire à la génération de clé de base de données réelle, et afin d'améliorer la capacité à isoler les tests les uns des autres lors de l'utilisation de la base de données en mémoire.

## Atténuations

Ceci peut casser une application qui repose sur la définition de valeurs de clé en mémoire spécifiques. Au lieu de cela, ne vous appuyez pas sur des valeurs de clés spécifiques, ou procédez à une mise à jour pour vous aligner au nouveau comportement.

### Les champs de stockage sont utilisés par défaut

[Suivi de problème n°12430](#)

#### Ancien comportement

Avant la version 3.0, même si le champ de stockage d'une propriété était connu, par défaut EF Core lisait et écrivait toujours la valeur de propriété à l'aide des méthodes get et set de la propriété. L'exception à cette règle concernait l'exécution des requêtes, où le champ de stockage était défini directement s'il était connu.

#### Nouveau comportement

Depuis EF Core 3.0, si le champ de stockage d'une propriété est connu, alors EF Core lit et écrit toujours cette propriété à l'aide du champ de stockage. Cela peut casser une application si celle-ci repose sur un comportement supplémentaire codé dans les méthodes get ou set.

## Pourquoi ?

Ce changement a été apporté afin d'empêcher EF Core de déclencher par erreur une logique métier par défaut quand vous effectuez des opérations de base de données impliquant les entités.

## Atténuations

Vous pouvez restaurer le comportement antérieur à la version 3.0 en configurant le mode d'accès à la propriété sur `ModelBuilder`. Par exemple :

```
modelBuilder.UsePropertyAccessMode(PropertyAccessMode.PreferFieldDuringConstruction);
```

### Erreur si plusieurs champs de stockage compatibles sont détectés

[Suivi de problème n°12523](#)

#### Ancien comportement

Avant EF Core 3.0, si plusieurs champs respectaient les règles de recherche du champ de stockage d'une propriété, l'un d'entre eux était choisi selon un ordre de priorité. Cela pouvait entraîner l'utilisation d'un champ incorrect en cas d'ambiguïté.

#### Nouveau comportement

À compter d'EF Core 3.0, si plusieurs champs sont mis en correspondance avec la même propriété, une exception est levée.

## Pourquoi ?

Ce changement a été apporté afin d'éviter d'utiliser en mode silencieux un champ plutôt qu'un autre quand un seul peut être correct.

## Atténuations

Pour les propriétés comportant des champs de stockage ambigus, le champ à utiliser doit être spécifié explicitement. Par exemple, à l'aide de l'API Fluent :

```
modelBuilder
    .Entity<Blog>()
    .Property(e => e.Id)
    .HasField("_id");
```

### Les noms de propriété de type « champ uniquement » doivent correspondre au nom du champ

#### Ancien comportement

Avant le EF Core 3,0, une propriété pourrait être spécifiée par une valeur de chaîne et si aucune propriété portant ce nom n'a été trouvée sur le type .NET, EF Core essaiera de la faire correspondre à un champ à l'aide de règles de Convention.

```
private class Blog
{
    private int _id;
    public string Name { get; set; }
}
```

```
modelBuilder
    .Entity<Blog>()
    .Property("Id");
```

#### Nouveau comportement

À compter d'EF Core 3,0, une propriété de type « champ uniquement » doit correspondre exactement au nom du champ.

```
modelBuilder
    .Entity<Blog>()
    .Property("_id");
```

#### Pourquoi ?

Ce changement a été apporté pour éviter d'utiliser un même champ pour deux propriétés portant un nom similaire. De plus, les règles de correspondance des propriétés de type « champ uniquement » sont désormais les mêmes que pour les propriétés mappées aux propriétés CLR.

#### Atténuations

Les propriétés de type « champ uniquement » doivent porter le même nom que le champ auquel elles sont mappées. Dans une version ultérieure de EF Core après 3,0, nous prévoyons de réactiver explicitement la configuration d'un nom de champ différent du nom de la propriété (voir le problème [#15307](#)) :

```
modelBuilder
    .Entity<Blog>()
    .Property("Id")
    .HasField("_id");
```

### AddDbContext/AddDbContextPool n'appelle plus AddLogging et AddMemoryCache

[Suivi de problème no 14756](#)

#### Ancien comportement

Avant le EF Core 3,0, l'appel de `AddDbContext` ou `AddDbContextPool` inscrirait également les services de journalisation et de mise en cache de la mémoire avec des appels à `AddLogging` et `AddMemoryCache`.

## Nouveau comportement

À compter d'EF Core 3,0, `AddDbContext` et `AddDbContextPool` n'inscriront plus ces services auprès de l'injection de dépendances.

### Pourquoi ?

Il n'est pas nécessaire pour EF Core 3,0 que ces services se trouvent dans le conteneur d'injection de dépendances de l'application. Toutefois, `ILoggerFactory` est utilisé par EF Core même s'il est inscrit dans ce conteneur.

## Atténuations

Si votre application a besoin de ces services, inscrivez-les explicitement auprès du conteneur d'injection de dépendances avec `AddLogging` ou `AddMemoryCache`.

### AddEntityFramework \* ajoute IMemoryCache avec une limite de taille

[#12905 du problème de suivi](#)

## Ancien comportement

Avant le EF Core 3,0, l'appel de `AddEntityFramework*` méthodes inscrirait également les services de mise en cache de mémoire avec DI sans limite de taille.

## Nouveau comportement

À compter de EF Core 3,0, `AddEntityFramework*` inscrit un service `IMemoryCache` avec une limite de taille. Si d'autres services ajoutés par la suite dépendent de `IMemoryCache`, ils peuvent rapidement atteindre la limite par défaut provoquant des exceptions ou une dégradation des performances.

### Pourquoi ?

L'utilisation de `IMemoryCache` sans limite peut entraîner une utilisation incontrôlée de la mémoire s'il existe un bogue dans la logique de mise en cache des requêtes ou si les requêtes sont générées dynamiquement. Le fait d'avoir une limite par défaut atténue une attaque potentielle par déni de compte.

## Atténuations

Dans la plupart des cas, l'appel de `AddEntityFramework*` n'est pas nécessaire si `AddDbContext` ou `AddDbContextPool` est également appelé. Par conséquent, la meilleure atténuation consiste à supprimer l'appel de `AddEntityFramework*`.

Si votre application a besoin de ces services, inscrivez une implémentation `IMemoryCache` explicitement avec le conteneur DI au préalable à l'aide de `AddMemoryCache`.

### DbContext.Entry effectue maintenant un DetectChanges local

[Suivi de problème n°13552](#)

## Ancien comportement

Avant EF Core 3,0, le fait d'appeler `DbContext.Entry` provoquait la détection des changements pour toutes les entités suivies. Cela garantissait que l'état exposé dans `EntityEntry` était à jour.

## Nouveau comportement

À compter d'EF Core 3,0, l'appel à `DbContext.Entry` tente uniquement de détecter les changements apportés dans l'entité donnée et dans toute entité principale suivie qui lui est associée. Cela signifie que les changements apportés ailleurs peuvent ne pas avoir été détectés par l'appel à cette méthode, ce qui pourrait avoir des

conséquences sur l'état de l'application.

Notez que si `ChangeTracker.AutoDetectChangesEnabled` a la valeur `false`, même cette détection de changement locale sera désactivée.

Les autres méthodes qui provoquent une détection des changements (par exemple `ChangeTracker.Entries` et `SaveChanges`) entraînent toujours un `DetectChanges` complet de toutes les entités suivies.

## Pourquoi ?

Ce changement a été apporté afin d'améliorer les performances par défaut de `context.Entry`.

## Atténuations

Appelez `ChgangeTracker.DetectChanges()` explicitement avant d'appeler `Entry` pour garantir le comportement antérieur à la version 3.0.

## Les clés de tableaux d'octets et de chaînes ne sont pas générées par client par défaut

[Suivi de problème n°14617](#)

### Ancien comportement

Avant EF Core 3.0, les propriétés de clés `string` et `byte[]` pouvaient être utilisées sans définir explicitement de valeur non null. Dans ce cas, la valeur de la clé était générée sur le client en tant que GUID, sérialisé en octets pour `byte[]`.

### Nouveau comportement

À compter d'EF Core 3.0, une exception est levée pour signaler qu'aucune valeur de clé n'a été définie.

## Pourquoi ?

Ce changement a été apporté car les valeurs `string` / `byte[]` générées par le client ne sont généralement pas utiles, et le comportement par défaut rendait les valeurs de clés générées de manière courante difficiles à expliquer.

## Atténuations

Vos pouvez obtenir le comportement antérieur à la version 3.0 en spécifiant explicitement que les propriétés de clés doivent utiliser des valeurs générées si aucune autre valeur non nulle n'est définie. Par exemple, avec l'API Fluent :

```
modelBuilder
    .Entity<Blog>()
    .Property(e => e.Id)
    .ValueGeneratedOnAdd();
```

Ou avec des annotations de données :

```
[DatabaseGenerated(DatabaseGeneratedOption.Identity)]
public string Id { get; set; }
```

## ILoggerFactory est désormais un service délimité

[Suivi de problème n°14698](#)

### Ancien comportement

Avant EF Core 3.0, `ILoggerFactory` était inscrit en tant que service singleton.

### Nouveau comportement

À compter d'EF Core 3.0, `ILoggerFactory` est inscrit en tant que service délimité.

## Pourquoi ?

Ce changement a été apporté afin d'autoriser l'association d'un journaliseur avec une instance `DbContext`, ce qui active d'autres fonctionnalités et élimine certains cas de comportement pathologique tels que l'explosion des fournisseurs de services internes.

## Atténuations

Ce changement ne devrait pas avoir d'impact sur le code de l'application, sauf en cas d'inscription et d'utilisation de services personnalisés sur le fournisseur de service interne EF Core, ce qui n'est pas courant. Dans ces cas-là, la plupart des composants continueront à fonctionner, mais tout service singleton qui dépendait de `ILoggerFactory` devra être modifié pour obtenir le `ILoggerFactory` d'une manière différente.

Si vous faites face à ce genre de situation, veuillez soumettre un problème par le biais du [suivi des problèmes GitHub EF Core](#) pour nous dire comment vous utilisez `ILoggerFactory`, afin que nous puissions mieux comprendre comment ne rien casser à l'avenir.

## Les proxys à chargement différé ne partent plus du principe que les propriétés de navigation sont entièrement chargées

[Suivi de problème n°12780](#)

### Ancien comportement

Avant EF Core 3.0, une fois qu'un `DbContext` était supprimé, il n'existe aucun moyen de savoir si une propriété de navigation donnée sur une entité obtenue à partir de ce contexte était entièrement chargée. Les serveurs proxy partaient du principe qu'une navigation de référence était chargée si elle avait une valeur non null, et qu'une navigation de collection était chargée si elle n'était pas vide. Dans ces cas-là, toute tentative de chargement différé constituait une no-op.

### Nouveau comportement

À compter d'EF Core 3.0, les proxys effectuent le suivi du chargement d'une propriété de navigation. Cela signifie qu'une tentative d'accès à une propriété de navigation qui est chargée une fois que le contexte a été supprimé sera toujours une no-op, même quand la navigation chargée est vide ou null. À l'inverse, une tentative d'accès à une propriété de navigation qui n'est pas chargée lèvera une exception si le contexte est supprimé, même si la propriété de navigation est une collection non vide. Si cette situation se présente, cela signifie que le code d'application tente d'utiliser le chargement différé à un moment non valide, et que l'application doit être modifiée afin de ne pas tenter cette opération.

## Pourquoi ?

Ce changement a été apporté afin de rendre le comportement cohérent et correct lors de la tentative de chargement différé sur une instance `DbContext` supprimée.

## Atténuations

Mettez à jour le code d'application pour qu'il ne tente pas d'effectuer un chargement différé avec un contexte supprimé, ou configuez cette opération pour qu'il s'agisse d'une no-op comme décrit dans le message d'exception.

## La création excessive de fournisseurs de services internes est maintenant une erreur par défaut

[Suivi de problème n°10236](#)

### Ancien comportement

Avant EF Core 3.0, un avertissement était enregistré dans le journal quand une application créait une quantité pathologique de fournisseurs de services internes.

## Nouveau comportement

À compter d'EF Core 3.0, cet avertissement est désormais considéré comme une erreur, et une exception est levée.

### Pourquoi ?

Ce changement a été apporté afin d'améliorer le code d'application avec une exposition plus explicite de ce cas pathologique.

### Atténuations

En présence de cette erreur, le mieux consiste à tenter de comprendre la cause racine, et de cesser de créer autant de fournisseurs de services internes. Toutefois, vous pouvez reconvertir cette erreur en avertissement (ou l'ignorer) par le biais de la configuration sur le `DbContextOptionsBuilder`. Par exemple :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .ConfigureWarnings(w => w.Log(CoreEventId.ManyServiceProvidersCreatedWarning));
}
```

## Nouveau comportement de HasOne/HasMany appelé avec une chaîne unique

[Suivi de problème no 9171](#)

### Ancien comportement

Avant EF Core 3.0, la façon dont était interprété le code qui appelait `HasOne` ou `HasMany` avec une seule chaîne prêtait à confusion. Par exemple :

```
modelBuilder.Entity<Samurai>().HasOne("Entrance").WithOne();
```

Le code semble relier `Samurai` à un autre type d'entité avec la propriété de navigation `Entrance`, qui peut être privée.

En réalité, ce code tente de créer une relation avec un certain type d'entité nommé `Entrance` sans propriété de navigation.

### Nouveau comportement

À compter d'EF Core 3.0, le code ci-dessus effectue maintenant ce qu'il semblait devoir faire avant.

### Pourquoi ?

L'ancien comportement était très déroutant, en particulier pour qui lisait le code de configuration à la recherche d'erreurs.

### Atténuations

Seules les applications qui configurent explicitement des relations en utilisant des chaînes comme noms de type sans spécifier explicitement la propriété de navigation sont concernées, ce qui n'est pas courant. Pour revenir au comportement précédent, transmettez explicitement `null` comme nom de propriété de navigation. Par exemple :

```
modelBuilder.Entity<Samurai>().HasOne("Some.Entity.Type.Name", null).WithOne();
```

## Le type de retour Task pour plusieurs méthodes asynchrones a été remplacé par ValueTask

[Suivi du problème no 15184](#)

## Ancien comportement

Les méthodes asynchrones suivantes retournaient précédemment un `Task<T>` :

- `DbContext.FindAsync()`
- `DbSet.FindAsync()`
- `DbContext.AddAsync()`
- `DbSet.AddAsync()`
- `ValueGenerator.NextValueAsync()` (et classes dérivées)

## Nouveau comportement

Les méthodes mentionnées précédemment retournent maintenant un `ValueTask<T>` sur le même `T` qu'avant.

### Pourquoi ?

Ce changement réduit le nombre d'allocations de tas induits par l'appel de ces méthodes, ce qui améliore les performances générales.

### Atténuations

Les applications qui sont simplement en attente des API ci-dessus doivent uniquement être recompilées, vous n'avez pas besoin de changer la source. Une utilisation plus complexe (par exemple, le passage du `Task` retourné à `Task.WhenAny()`) nécessite généralement que le `ValueTask<T>` retourné soit converti en `Task<T>` en appelant `AsTask()` sur lui. Notez que cette opération annule la réduction d'allocation apportée par ce changement.

## L'annotation Relational:TypeMapping est désormais simplement TypeMapping

[Suivi de problème n°9913](#)

## Ancien comportement

Le nom d'annotation pour les annotations de mappage de types était « annotations ».

## Nouveau comportement

Le nom d'annotation pour les annotations de mappage de types est désormais « TypeMapping ».

### Pourquoi ?

Les mappages de types ne sont désormais plus utilisés uniquement pour les fournisseurs de bases de données relationnelles.

### Atténuations

Ce changement casse uniquement les applications qui accèdent au mappage de types directement en tant qu'annotation, ce qui n'est pas courant. La meilleure solution consiste à utiliser la surface d'API pour accéder aux mappages de types, plutôt que d'utiliser directement l'annotation.

## ToTable sur un type dérivé lève une exception

[Suivi de problème n°11811](#)

## Ancien comportement

Avant EF Core 3.0, l'appel à `ToTable()` sur un type dérivé était ignoré, car seule la stratégie de mappage d'héritage était TPH où cela n'est pas valide.

## Nouveau comportement

À compter d'EF Core 3.0, et en préparation à l'ajout de la prise en charge de TPT et TPC dans une version ultérieure, l'appel à `ToTable()` sur un type dérivé lève maintenant une exception afin d'éviter tout changement

inattendu du mappage à l'avenir.

## Pourquoi ?

Actuellement le mappage d'un type dérivé à une autre table n'est pas une opération valide. Ce changement permettra d'éviter toute cassure ultérieure quand cette opération deviendra valide.

## Atténuations

Supprimez toutes les tentatives de mappage de types dérivés à d'autres tables.

## ForSqlServerHasIndex est remplacé par HasIndex

[Suivi de problème n°12366](#)

### Ancien comportement

Avant EF Core 3.0, `ForSqlServerHasIndex().ForSqlServerInclude()` offrait un moyen de configurer des colonnes utilisées avec `INCLUDE`.

### Nouveau comportement

À compter d'EF Core 3.0, l'utilisation de `Include` sur un index est prise en charge au niveau relationnel. Utilisez `HasIndex().ForSqlServerInclude()`.

## Pourquoi ?

Ce changement a été apporté afin de consolider l'API pour les index avec `Include` dans un seul emplacement pour tous les fournisseurs de bases de données.

## Atténuations

Utilisez la nouvelle API, comme indiqué ci-dessus.

## Changements apportés à l'API de métadonnées

[Suivi du problème no 214](#)

### Nouveau comportement

Les propriétés suivantes ont été converties en méthodes d'extension :

- `IEntityType.QueryFilter` -> `GetQueryFilter()`
- `IEntityType.DefiningQuery` -> `GetDefiningQuery()`
- `IProperty.IsShadowProperty` -> `IsShadowProperty()`
- `IProperty.BeforeSaveBehavior` -> `GetBeforeSaveBehavior()`
- `IProperty.AfterSaveBehavior` -> `GetAfterSaveBehavior()`

## Pourquoi ?

Ce changement simplifie l'implémentation des interfaces mentionnées précédemment.

## Atténuations

Utilisez les nouvelles méthodes d'extension.

## Modifications de l'API de métadonnées spécifiques au fournisseur

[Suivi du problème no 214](#)

### Nouveau comportement

Les méthodes d'extension spécifiques au fournisseur seront aplatis :

- `IProperty.Relational().ColumnName` -> `IProperty.GetColumnName()`
- `IEntityType.SqlServer().IsMemoryOptimized` -> `IEntityType.IsMemoryOptimized()`
- `PropertyBuilder.UseSqlServerIdentityColumn()` -> `PropertyBuilder.UseIdentityColumn()`

## Pourquoi ?

Ce changement simplifie l'implémentation des méthodes d'extension mentionnées précédemment.

## Atténuations

Utilisez les nouvelles méthodes d'extension.

## EF Core n'envoie plus de pragma pour l'application de clé étrangère SQLite

[Suivi de problème n°12151](#)

### Ancien comportement

Avant EF Core 3.0, EF Core envoyait `PRAGMA foreign_keys = 1` quand une connexion à SQLite était ouverte.

### Nouveau comportement

À compter d'EF Core 3.0, EF Core n'envoie plus de `PRAGMA foreign_keys = 1` quand une connexion à SQLite est ouverte.

## Pourquoi ?

Ce changement a été apporté car EF Core utilise `SQLitePCLRaw.bundle_e_sqlite3` par défaut, ce qui signifie que l'application de clé étrangère est activée par défaut et n'a pas besoin d'être activée explicitement chaque fois qu'une connexion est ouverte.

## Atténuations

Les clés étrangères sont activées par défaut dans `SQLitePCLRaw.bundle_e_sqlite3`, qui est utilisé par défaut pour EF Core. Pour les autres cas, vous pouvez activer les clés étrangères en spécifiant `Foreign Keys=True` dans votre chaîne de connexion.

## Microsoft.EntityFrameworkCore.Sqlite dépend désormais de SQLitePCLRaw.bundle\_e\_sqlite3

### Ancien comportement

Avant EF Core 3.0, EF Core utilisait `SQLitePCLRaw.bundle_green`.

### Nouveau comportement

À compter d'EF Core 3.0, EF Core utilise `SQLitePCLRaw.bundle_e_sqlite3`.

## Pourquoi ?

Ce changement a été apporté afin que la version de SQLite utilisée sur iOS soit cohérente avec d'autres plateformes.

## Atténuations

Pour utiliser la version de SQLite native sur iOS, configurez `Microsoft.Data.Sqlite` de façon à utiliser un autre bundle `SQLitePCLRaw`.

## Les valeurs GUID sont maintenant stockées au format TEXT sur SQLite

[Suivi de problème #15078](#)

### Ancien comportement

Avant, les valeurs GUID étaient stockées comme valeurs BLOB sur SQLite.

## Nouveau comportement

Maintenant, les valeurs Guid sont stockées au format TEXT.

### Pourquoi ?

Le format binaire des valeurs GUID n'est pas normalisé. Le stockage des valeurs au format TEXT rend la base de données plus compatible avec d'autres technologies.

### Atténuations

Vous pouvez migrer des bases de données existantes vers le nouveau format en exécutant SQL comme suit.

```
UPDATE MyTable
SET GuidColumn = hex(substr(GuidColumn, 4, 1)) ||
    hex(substr(GuidColumn, 3, 1)) ||
    hex(substr(GuidColumn, 2, 1)) ||
    hex(substr(GuidColumn, 1, 1)) || '-' ||
    hex(substr(GuidColumn, 6, 1)) ||
    hex(substr(GuidColumn, 5, 1)) || '-' ||
    hex(substr(GuidColumn, 8, 1)) ||
    hex(substr(GuidColumn, 7, 1)) || '-' ||
    hex(substr(GuidColumn, 9, 2)) || '-' ||
    hex(substr(GuidColumn, 11, 6))
WHERE typeof(GuidColumn) == 'blob';
```

Dans EF Core, vous pouvez également continuer à utiliser le comportement précédent en configurant un convertisseur de valeur sur ces propriétés.

```
modelBuilder
    .Entity<MyEntity>()
    .Property(e => e.GuidProperty)
    .HasConversion(
        g => g.ToByteArray(),
        b => new Guid(b));
```

Microsoft.Data.Sqlite peut toujours lire les valeurs GUID dans les colonnes BLOB et TEXT. Toutefois, en raison du changement du format par défaut pour les paramètres et les constantes, vous devrez probablement apporter des modifications dans la plupart des scénarios utilisant des valeurs GUID.

## Les valeurs char sont maintenant stockées au format TEXT sur SQLite

[Suivi de problème no 15020](#)

### Ancien comportement

Avant, les valeurs char étaient stockées comme valeurs INTEGER sur SQLite. Par exemple, une valeur char de A était stockée comme valeur entière 65.

## Nouveau comportement

Maintenant, les valeurs char sont stockées au format TEXT.

### Pourquoi ?

Le stockage des valeurs au format TEXT est plus naturel et rend la base de données plus compatible avec d'autres technologies.

### Atténuations

Vous pouvez migrer des bases de données existantes vers le nouveau format en exécutant SQL comme suit.

```
UPDATE MyTable
SET CharColumn = char(CharColumn)
WHERE typeof(CharColumn) = 'integer';
```

Dans EF Core, vous pouvez également continuer à utiliser le comportement précédent en configurant un convertisseur de valeur sur ces propriétés.

```
modelBuilder
    .Entity<MyEntity>()
    .Property(e => e.CharProperty)
    .HasConversion(
        c => (long)c,
        i => (char)i);
```

Microsoft.Data.Sqlite est aussi toujours capable de lire les valeurs de caractère à partir de colonnes INTEGER et TEXT, donc certains scénarios peuvent ne nécessiter aucune action.

## Les ID de migration sont maintenant générés à l'aide du calendrier de la culture invariante

[Suivi de problème no 12978](#)

### Ancien comportement

Les ID de migration ont été générés par inadvertance à l'aide du calendrier de la culture actuelle.

### Nouveau comportement

Maintenant, les ID de migration sont toujours générés à l'aide du calendrier de la culture invariante (grégorien).

### Pourquoi ?

L'ordre des migrations est important lors de la mise à jour de la base de données ou de la résolution des conflits de fusion. L'utilisation du calendrier invariant permet d'éviter les problèmes de classement qui peuvent se produire si les membres de l'équipe ont des calendriers système différents.

### Atténuations

Ce changement affecte tous ceux qui utilisent un calendrier non grégorien où l'année est en avance sur le calendrier grégorien (comme le calendrier bouddhiste thaïlandais). Les ID de migration existants devront être mis à jour afin que les nouvelles migrations soient classées après les migrations existantes.

Vous trouverez les ID de migration dans l'attribut Migration des fichiers de concepteur des migrations.

```
[DbContext(typeof(MyDbContext))]
-[Migration("25620318122820_MyMigration")]
+[Migration("20190318122820_MyMigration")]
partial class MyMigration
{
```

Le tableau de l'historique Migrations doit également être mis à jour.

```
UPDATE __EFMigrationsHistory
SET MigrationId = CONCAT(LEFT(MigrationId, 4) - 543, SUBSTRING(MigrationId, 4, 150))
```

## UseRowNumberForPaging a été supprimé

[Suivi de problème no 16400](#)

### Ancien comportement

Avant EF Core 3.0, `UseRowNumberForPaging` pouvait être utilisé pour générer SQL pour la pagination qui est compatible avec SQL Server 2008.

## Nouveau comportement

À compter de EF Core 3.0, EF génère uniquement SQL pour la pagination qui est uniquement compatible avec les versions de SQL Server ultérieures.

### Pourquoi ?

Nous effectuons cette modification, car [SQL Server 2008 n'est plus un produit pris en charge](#) et la mise à jour de cette fonctionnalité pour fonctionner avec les modifications de requête effectuées dans EF Core 3.0 est un travail significatif.

### Atténuations

Nous vous recommandons d'effectuer la mise à jour vers une version plus récente de SQL Server ou d'utiliser un niveau de compatibilité plus élevé, afin que le SQL généré soit pris en charge. Cela dit, si vous ne pouvez pas faire cela, veuillez [commenter le problème de suivi](#) en incluant des détails. Nous pourrions revisiter cette décision en fonction des commentaires.

## Les info/métadonnées d'extension ont été supprimées de `IDbContextOptionsExtension`

[Suivi du problème no 16119](#)

### Ancien comportement

`IDbContextOptionsExtension` contenait des méthodes permettant de fournir des métadonnées relatives à l'extension.

### Nouveau comportement

Ces méthodes ont été déplacées vers une nouvelle classe de base abstraite `DbContextOptionsExtensionInfo`, qui est retournée à partir d'une nouvelle propriété `IDbContextOptionsExtension.Info`.

### Pourquoi ?

Sur les versions de 2.0 à 3.0, nous devions ajouter ou modifier ces méthodes plusieurs fois. Les sortir dans une nouvelle classe de base abstraite simplifie l'apport de ces types de changements sans résiliation des extensions existantes.

### Atténuations

Mettre à jour des extensions pour suivre le nouveau modèle. Des exemples se trouvent dans la plupart des implémentations de `IDbContextOptionsExtension` pour différents types d'extensions dans le code source EF Core.

## `LogQueryPossibleExceptionWithAggregateOperator` a été renommé

[Suivi de problème #10985](#)

### Changement

`RelationalEventId.LogQueryPossibleExceptionWithAggregateOperator` a été renommé en  
`RelationalEventId.LogQueryPossibleExceptionWithAggregateOperatorWarning`.

### Pourquoi ?

Aline le nom de cet événement d'avertissement avec tous les autres événements d'avertissement.

### Atténuations

Utilisez le nouveau nom. (Notez que le numéro d'ID événement n'a pas changé.)

## Clarifier les noms de contrainte de clé étrangère dans l'API

[Suivi du problème #10730](#)

### Ancien comportement

Avant EF Core 3.0, les noms de contrainte de clé étrangère étaient désignés simplement par le terme « nom ». Par exemple :

```
var constraintName = myForeignKey.Name;
```

### Nouveau comportement

À compter d'EF Core 3.0, les noms de contrainte de clé étrangère sont désignés par « noms de contrainte ». Par exemple :

```
var constraintName = myForeignKey.ConstraintName;
```

### Pourquoi ?

Ce changement rend le nommage plus cohérent dans ce domaine. Il clarifie également le fait qu'il s'agit du nom de la contrainte de clé étrangère et pas du nom de la colonne ou de la propriété sur laquelle la clé étrangère est définie.

### Atténuations

Utilisez le nouveau nom.

## IRelationalDatabaseCreator.HasTables/HasTablesAsync ont été rendues publiques

[Suivi du problème no 15997](#)

### Ancien comportement

Avant EF Core 3.0, ces méthodes étaient protégées.

### Nouveau comportement

À compter d'EF Core 3.0, ces méthodes sont publiques.

### Pourquoi ?

Ces méthodes sont utilisées par EF pour déterminer si une base de données est créée mais vide. Cela peut également être utile depuis l'extérieur d'EF lorsque vous déterminez s'il faut appliquer des migrations ou pas.

### Atténuations

Modifiez l'accessibilité de n'importe quel remplacements.

## Microsoft.EntityFrameworkCore.Design est désormais un package DevelopmentDependency

[Suivi du problème no 11506](#)

### Ancien comportement

Avant EF Core 3.0, Microsoft.EntityFrameworkCore.Design était un package NuGet normal dont l'assembly pouvait être référencée par des projets qui en dépendaient.

### Nouveau comportement

À compter d'EF Core 3.0, il s'agit d'un package DevelopmentDependency. Cela signifie que la dépendance ne sera pas transitive dans d'autres projets, et que vous ne pouvez plus, par défaut, référencer son assembly.

## Pourquoi ?

Ce package est uniquement destiné à être utilisé au moment du design. Les applications déployées ne doivent pas y faire référence. Le fait de faire de ce package un DevelopmentDependency renforce cette recommandation.

## Atténuations

Si vous avez besoin de référencer ce package pour remplacer le comportement de EF Core au moment de la conception, vous pouvez mettre à jour les métadonnées de l'élément PackageReference dans votre projet.

```
<PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="3.0.0">
  <PrivateAssets>all</PrivateAssets>
  <!-- Remove IncludeAssets to allow compiling against the assembly -->
  <!--<IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>-->
</PackageReference>
```

Si le package est référencé de manière transitive via Microsoft.EntityFrameworkCore.Tools, vous devez ajouter un PackageReference explicite au package pour modifier ses métadonnées. Une telle référence explicite doit être ajoutée à n'importe quel projet où les types du package sont nécessaires.

## SQLitePCL.raw mis à jour vers la version 2.0.0

[Suivi du problème no 14824](#)

### Ancien comportement

Microsoft.EntityFrameworkCore.Sqlite dépendait précédemment de la version 1.1.12 de SQLitePCL.raw.

### Nouveau comportement

Nous avons mis à jour notre package pour dépendre de la version 2.0.0.

## Pourquoi ?

La version 2.0.0 de SQLitePCL.raw cible .NET Standard 2.0. Elle ciblait précédemment .NET Standard 1.1 qui nécessitait une fermeture volumineuse de packages transitifs pour fonctionner.

## Atténuations

SQLitePCL.raw version 2.0.0 inclut des changements cassants. Pour plus d'informations, consultez les [notes de publication](#).

## NetTopologySuite mis à jour vers la version 2.0.0

[Suivi de problème no 14825](#)

### Ancien comportement

Les packages spatiaux dépendaient précédemment de la version 1.15.1 de NetTopologySuite.

### Nouveau comportement

Nous avons mis à jour notre package pour dépendre de la version 2.0.0.

## Pourquoi ?

La version 2.0.0 de NetTopologySuite vise à résoudre plusieurs problèmes de facilité d'utilisation rencontrés par les utilisateurs EF Core.

## Atténuations

NetTopologySuite version 2.0.0 inclut des changements cassants. Pour plus d'informations, consultez les [notes de publication](#).

## **Microsoft. Data. SqlClient est utilisé à la place de System. Data. SqlClient**

[#15636 du problème de suivi](#)

### **Ancien comportement**

Microsoft.EntityFrameworkCore.SqlServer était auparavant tributaire de System. Data. SqlClient.

### **Nouveau comportement**

Nous avons mis à jour notre package pour dépendre de Microsoft. Data. SqlClient.

### **Pourquoi ?**

Microsoft. Data. SqlClient est le pilote d'accès aux données phare pour la SQL Server à l'avenir, et System. Data. SqlClient ne représente plus le point de vue du développement. Certaines fonctionnalités importantes, telles que Always Encrypted, sont uniquement disponibles sur Microsoft. Data. SqlClient.

### **Atténuations**

Si votre code prend une dépendance directe sur System. Data. SqlClient, vous devez le modifier pour qu'il fasse référence à Microsoft. Data. SqlClient à la place. étant donné que les deux packages maintiennent un très haut niveau de compatibilité d'API, il ne doit y avoir qu'une simple modification de package et d'espace de noms.

## **Plusieurs relations autoréférencées ambiguës doivent être configurées**

[Suivi de problème no 13573](#)

### **Ancien comportement**

Un type d'entité avec plusieurs propriétés de navigation unidirectionnelle autoréférencées et les clés étrangères correspondantes a été incorrectement configuré en tant que relation unique. Par exemple :

```
public class User
{
    public Guid Id { get; set; }
    public User CreatedBy { get; set; }
    public User UpdatedBy { get; set; }
    public Guid CreatedById { get; set; }
    public Guid? UpdatedById { get; set; }
}
```

### **Nouveau comportement**

Ce scénario est maintenant détecté dans la génération de modèle et une exception est levée, indiquant que le modèle est ambigu.

### **Pourquoi ?**

Le modèle résultant est ambigu et probablement erroné dans ce cas.

### **Atténuations**

Utilisez la configuration complète de la relation. Par exemple :

```
modelBuilder
    .Entity<User>()
    .HasOne(e => e.CreatedBy)
    .WithMany();

modelBuilder
    .Entity<User>()
    .HasOne(e => e.UpdatedBy)
    .WithMany();
```

**DbFunction. Schema étant null ou une chaîne vide, il est configuré pour être dans le schéma par défaut du modèle**

[#12757 du problème de suivi](#)

### Ancien comportement

Un DbFunction configuré avec un schéma sous forme de chaîne vide a été traité comme une fonction intégrée sans schéma. Par exemple, le code suivant mappe `DatePart` fonction CLR à `DATEPART` fonction intégrée sur SqlServer.

```
[DbFunction("DATEPART", Schema = "")]  
public static int? DatePart(string datePartArg, DateTime? date) => throw new Exception();
```

### Nouveau comportement

Tous les mappages de DbFunction sont considérés comme mappés à des fonctions définies par l'utilisateur. Par conséquent, la valeur de chaîne vide placerait la fonction dans le schéma par défaut pour le modèle. Ce peut être le schéma configuré explicitement via l'API Fluent `modelBuilder.HasDefaultSchema()` ou `dbo` dans le cas contraire.

### Pourquoi ?

Le schéma précédemment vide était un moyen de traiter que la fonction est intégrée, mais cette logique s'applique uniquement à SqlServer, où les fonctions intégrées n'appartiennent à aucun schéma.

### Atténuations

Configurez la traduction de DbFunction manuellement pour la mapper à une fonction intégrée.

```
modelBuilder
    .HasDbFunction(typeof(MyContext).GetMethod(nameof(MyContext.DatePart)))
    .HasTranslation(args => SqlFunctionExpression.Create("DatePart", args, typeof(int?), null));
```

# Nouvelles fonctionnalités d'EF Core 2.2

07/11/2019 • 4 minutes to read • [Edit Online](#)

## Prise en charge des données spatiales

Les données spatiales peuvent être utilisées pour représenter l'emplacement physique et la forme d'objets. Plusieurs bases de données peuvent nativement stocker, indexer et interroger des données spatiales. Les scénarios courants incluent l'interrogation d'objets dans un rayon donné et le test pour vérifier si un polygone contient un emplacement donné. EF Core 2.2 prend désormais en charge l'utilisation de données spatiales provenant de diverses bases de données à l'aide de types issus de la bibliothèque [NetTopologySuite](#) (NTS).

La prise en charge des données spatiales est implémenté sous forme d'une série de packages d'extension spécifiques à un fournisseur. Chacun de ces packages contribue aux mappages des types NTS et des méthodes ainsi que des types spatiaux et fonctions correspondants dans la base de données. Ces extensions de fournisseur sont désormais disponibles pour [SQL Server](#), [SQLite](#), et [PostgreSQL](#) (à partir du [projet Npgsql](#)). Les types spatiaux peuvent être utilisés directement avec le [fournisseur en mémoire EF Core](#), sans extensions supplémentaires.

Une fois l'extension du fournisseur installée, vous pouvez ajouter à vos entités des propriétés de types pris en charge. Exemple :

```
using NetTopologySuite.Geometries;

namespace MyApp
{
    public class Friend
    {
        [Key]
        public string Name { get; set; }

        [Required]
        public Point Location { get; set; }
    }
}
```

Vous pouvez alors conserver les entités avec des données spatiales :

```
using (var context = new MyDbContext())
{
    context.Add(
        new Friend
        {
            Name = "Bill",
            Location = new Point(-122.34877, 47.6233355) {SRID = 4326 }
        });
    context.SaveChanges();
}
```

Et vous pouvez exécuter des requêtes de base de données basées sur des données spatiales et des opérations :

```
var nearestFriends =
    (from f in context.Friends
    orderby f.Location.Distance(myLocation) descending
    select f).Take(5).ToList();
```

Pour plus d'informations sur cette fonctionnalité, consultez la [documentation sur les types spatiaux](#).

## Collections d'entités détenues

EF Core 2.0 a ajouté la possibilité de modéliser la propriété dans les associations un-à-un. EF Core 2.2 étend la capacité d'exprimer la propriété aux associations un-à-plusieurs. La propriété permet de limiter la façon dont les entités sont utilisées.

Par exemple, les entités détenues :

- Peuvent apparaître uniquement sur les propriétés de navigation d'autres types d'entités.
- Sont automatiquement chargées et ne peut être uniquement suivies par un objet DbContext en même temps que leur propriétaire.

Dans les bases de données relationnelles, les collections détenues sont mappées à des tables distinctes du propriétaire, comme des associations un-à-plusieurs standard. Mais dans les bases de données orientées document, nous prévoyons d'imbriquer les entités détenues (dans des collections ou références détenues) dans le même document que le propriétaire.

Vous pouvez utiliser cette fonctionnalité en appelant la nouvelle API `OwnsMany()` :

```
modelBuilder.Entity<Customer>().OwnsMany(c => c.Addresses);
```

Pour plus d'informations, consultez la [documentation mise à jour sur les entités détenues](#).

## Balises de requête

Cette fonctionnalité simplifie la corrélation des requêtes LINQ dans le code avec des requêtes SQL générées et capturées dans des journaux.

Pour tirer parti des balises de requête, vous annotez une requête LINQ à l'aide de la nouvelle méthode `TagWith()`. En utilisant la requête spatiale d'un exemple précédent :

```
var nearestFriends =
    (from f in context.Friends.TagWith(@"This is my spatial query!")
    orderby f.Location.Distance(myLocation) descending
    select f).Take(5).ToList();
```

Cette requête LINQ générera la sortie SQL suivante :

```
-- This is my spatial query!

SELECT TOP(@__p_1) [f].[Name], [f].[Location]
FROM [Friends] AS [f]
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

Pour plus d'informations, consultez la [documentation sur les balises de requête](#).

# Nouvelles fonctionnalités d'EF Core 2.1

07/11/2019 • 12 minutes to read • [Edit Online](#)

En plus de nombreux correctifs de bogues et de petites améliorations des fonctionnalités et des performances, EF Core 2.1 inclut de nouvelles fonctionnalités intéressantes :

## Chargement différé

EF Core contient désormais les blocs de construction nécessaires pour permettre à quiconque de créer des classes d'entité capables de charger leurs propriétés de navigation à la demande. Nous avons également introduit un nouveau package, Microsoft.EntityFrameworkCore.Proxies, qui tire parti de ces composants pour produire des classes proxy de chargement différé basées sur des classes d'entité ayant subi des modifications minimales (par exemple des classes avec des propriétés de navigation virtuelles).

Pour plus d'informations sur cette rubrique, lisez la [section sur le chargement différé](#).

## Paramètres dans les constructeurs d'entité

L'un des blocs de construction nécessaires pour le chargement différé prend désormais en charge la création d'entités qui acceptent des paramètres dans leurs constructeurs. Vous pouvez utiliser des paramètres pour injecter des valeurs de propriété, des délégués de chargement différé et des services.

Pour plus d'informations sur cette rubrique, lisez la [section sur le constructeur d'entité avec des paramètres](#).

## Conversions de valeurs

Jusqu'à présent, EF Core pouvait uniquement mapper les propriétés de types pris en charge nativement par le fournisseur de base de données sous-jacent. Les valeurs étaient copiées entre les colonnes et les propriétés sans aucune transformation. À compter d'EF Core 2.1, les conversions de valeurs peuvent être appliquées pour transformer les valeurs obtenues à partir de colonnes avant leur application aux propriétés, et vice versa. Nous avons un certain nombre de conversions qui peuvent être appliquées par convention, le cas échéant, ainsi qu'une API de configuration explicite qui permet d'inscrire des conversions personnalisées entre des colonnes et des propriétés. Parmi les applications de cette fonctionnalité, citons les suivantes :

- Stockage des enums sous forme de chaînes
- Mappage d'entiers non signés avec SQL Server
- Chiffrement et déchiffrement automatiques des valeurs de propriété

Pour plus d'informations sur cette rubrique, lisez la [section sur les conversions de valeurs](#).

## Traduction LINQ GroupBy

Avant la version 2.1, l'opérateur LINQ GroupBy dans EF Core était toujours évalué en mémoire. Il est désormais possible de le traduire en clause SQL GROUP BY dans les scénarios les plus courants.

Cet exemple illustre une requête avec GroupBy utilisée pour calculer différentes fonctions d'agrégation :

```
var query = context.Orders
    .GroupBy(o => new { o.CustomerId, o.EmployeeId })
    .Select(g => new
    {
        g.Key.CustomerId,
        g.Key.EmployeeId,
        Sum = g.Sum(o => o.Amount),
        Min = g.Min(o => o.Amount),
        Max = g.Max(o => o.Amount),
        Avg = g.Average(o => o.Amount)
    });
});
```

La traduction SQL correspondante ressemble à ceci :

```
SELECT [o].[CustomerId], [o].[EmployeeId],
    SUM([o].[Amount]), MIN([o].[Amount]), MAX([o].[Amount]), AVG([o].[Amount])
FROM [Orders] AS [o]
GROUP BY [o].[CustomerId], [o].[EmployeeId];
```

## Amorçage des données

Avec la nouvelle version, il sera possible de fournir des données initiales pour remplir une base de données. Contrairement à ce qui se passe dans EF6, l'amorçage des données est associé à un type d'entité dans le cadre de la configuration du modèle. Les migrations EF Core peuvent automatiquement calculer les opérations d'insertion, de mise à jour ou de suppression à appliquer au moment de la mise à niveau de la base de données vers une nouvelle version du modèle.

Par exemple, utilisez ceci pour configurer les données d'amorçage d'un Post dans `OnModelCreating` :

```
modelBuilder.Entity<Post>().HasData(new Post{ Id = 1, Text = "Hello World!" });
```

Pour plus d'informations sur cette rubrique, lisez la [section sur l'amorçage des données](#).

## Types de requêtes

Un modèle EF Core peut désormais inclure des types de requêtes. Contrairement aux types d'entités, les types de requêtes ne sont pas associés à une définition de clé et ne peuvent pas être insérés, supprimés ou mis à jour (autrement dit, ils sont en lecture seule). Toutefois, ils peuvent être retournés directement par des requêtes. Voici quelques scénarios d'usages pour les types de requêtes :

- Mappage à des vues sans clés primaires
- Mappage à des tables sans clés primaires
- Mappage à des requêtes définies dans le modèle
- Utilisation comme type de retour pour les requêtes `FromSql()`

Pour plus d'informations sur cette rubrique, lisez la [section sur les types de requêtes](#).

## Include pour les types dérivés

Vous pouvez désormais spécifier des propriétés de navigation définies uniquement sur des types dérivés lors de l'écriture d'expressions pour la méthode `Include`. Pour la version fortement typée de `Include`, nous prenons en charge l'utilisation d'un cast explicite ou de l'opérateur `as`. Nous prenons également en charge le référencement des noms de propriété de navigation définis sur des types dérivés dans la version chaîne de `Include` :

```
var option1 = context.People.Include(p => ((Student)p).School);
var option2 = context.People.Include(p => (p as Student).School);
var option3 = context.People.Include("School");
```

Pour plus d'informations sur cette rubrique, lisez la [section sur Include avec des types dérivés](#).

## Prise en charge de System.Transactions

Vous pouvez désormais utiliser les fonctionnalités System.Transactions, notamment TransactionScope. Celles-ci fonctionnent sur le .NET Framework et .NET Core avec des fournisseurs de base de données compatibles.

Pour plus d'informations sur cette rubrique, lisez la [section sur System.Transactions](#).

## Meilleur classement des colonnes dans la migration initiale

Après avoir passé en revue les commentaires des clients, nous avons mis à jour les migrations de manière à générer initialement des colonnes pour les tables dans le même ordre que les propriétés déclarées dans les classes. Notez qu'EF Core ne peut pas changer l'ordre si de nouveaux membres sont ajoutés après la création de la table initiale.

## Optimisation des sous-requêtes corrélées

Nous avons amélioré la traduction des requêtes pour éviter l'exécution de requêtes SQL « N + 1 » dans de nombreux scénarios courants dans lesquels l'utilisation d'une propriété de navigation dans la projection débouche sur la jointure des données de la requête racine avec les données d'une sous-requête corrélée. L'optimisation nécessite la mise en mémoire tampon des résultats à partir de la sous-requête. Nous vous demandons par ailleurs de modifier la requête pour procéder à l'adhésion du nouveau comportement.

Par exemple, la requête suivante est normalement traduite en une seule requête pour Customers, plus N requêtes séparées pour Orders (où « N » est le nombre de clients retournés) :

```
var query = context.Customers.Select(
    c => c.Orders.Where(o => o.Amount > 100).Select(o => o.Amount));
```

En incluant `ToList()` au bon endroit, vous indiquez que la mise en mémoire tampon est appropriée pour Orders, ce qui déclenche l'optimisation :

```
var query = context.Customers.Select(
    c => c.Orders.Where(o => o.Amount > 100).Select(o => o.Amount).ToList());
```

Notez que cette requête est traduite en deux requêtes SQL seulement : une pour Customers et la suivante pour Orders.

## Attribut [Owned]

Il est désormais possible de configurer des [types d'entités détenus](#) en annotant simplement le type avec `[Owned]` et en veillant à ce que l'entité propriétaire soit ajoutée au modèle :

```
[Owned]
public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}

public class Order
{
    public int Id { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}
```

## Outil en ligne de commande dotnet-ef inclus dans le SDK .NET Core

Les commandes `ef-dotnet` font désormais partie du SDK .NET Core. Ainsi, il n'est plus nécessaire d'utiliser `DotNetCliToolReference` dans le projet pour pouvoir utiliser des migrations ou structurer un `DbContext` à partir d'une base de données existante.

Pour plus d'informations sur la façon d'activer les outils en ligne de commande pour différentes versions du SDK .NET Core et EF Core, consultez la section sur [l'installation des outils](#).

## Package Microsoft.EntityFrameworkCore.Abstractions

Le nouveau package contient des attributs et des interfaces que vous pouvez utiliser dans vos projets pour alléger les fonctionnalités EF Core sans dépendance envers Core EF dans sa globalité. Par exemple, l'attribut `[Owned]` et l'interface `ILazyLoader` se trouvent ici.

## Événements de changement d'état

Les nouveaux événements `Tracked` et `StateChanged` sur `ChangeTracker` peuvent être utilisés pour écrire une logique qui réagit aux entités qui entrent dans `DbContext` ou modifiant leur état.

## Analyseur de paramètre SQL brut

Un nouvel analyseur de code est inclus avec EF Core. Il détecte les utilisations potentiellement dangereuses de nos API SQL brutes, comme `FromSql` ou `ExecuteSqlCommand`. Par exemple, dans la requête suivante, un avertissement s'affiche, car `minAge` n'est pas paramétrable :

```
var sql = $"SELECT * FROM People WHERE Age > {minAge}";
var query = context.People.FromSql(sql);
```

## Compatibilité des fournisseurs de base de données

Nous vous recommandons d'utiliser EF Core 2.1 avec des fournisseurs qui ont été mis à jour ou au moins testés pour fonctionner avec EF Core 2.1.

### TIP

Si vous vous heurtez à des incompatibilités inattendues ou à d'autres problèmes liés aux nouvelles fonctionnalités, ou si vous avez des commentaires à propos de ces nouveautés, utilisez [notre système de suivi des problèmes](#) pour nous en faire part.

# Nouvelles fonctionnalités d'EF Core 2.0

05/12/2019 • 18 minutes to read • [Edit Online](#)

## .NET Standard 2.0

EF Core cible désormais .NET Standard 2.0, ce qui signifie qu'il peut fonctionner avec .NET Core 2.0, .NET Framework 4.6.1 et d'autres bibliothèques qui implémentent .NET Standard 2.0. Pour plus d'informations sur ce qui est pris en charge, consultez [Implémentations .NET prises en charge](#).

## Modélisation

### Fractionnement de table

Vous pouvez désormais mapper deux ou plusieurs types d'entité à la même table, où la ou les colonnes de clé primaire sont partagées et chaque ligne correspond à deux ou plusieurs entités.

Pour utiliser le fractionnement de table, vous devez configurer une relation d'identification (où les propriétés de clé étrangère forment la clé primaire) entre tous les types d'entité partageant la table :

```
modelBuilder.Entity<Product>()
    .HasOne(e => e.Details).WithOne(e => e.Product)
    .HasForeignKey<ProductDetails>(e => e.Id);
modelBuilder.Entity<Product>().ToTable("Products");
modelBuilder.Entity<ProductDetails>().ToTable("Products");
```

Pour plus d'informations sur cette fonctionnalité, lisez la [section sur le fractionnement de table](#).

### Types détenus

Un type d'entité détenu peut partager le même type .NET avec un autre type d'entité détenu, mais étant donné qu'il ne peut pas être identifié uniquement par le type .NET, il doit y avoir une navigation à partir d'un autre type d'entité. L'entité qui contient la navigation définie est le propriétaire. Quand le propriétaire fait l'objet d'une interrogation, les types détenus sont inclus par défaut.

Par convention, une clé primaire cachée est créée pour le type détenu et est mappée à la même table que le propriétaire à l'aide du fractionnement de table. Cela permet d'utiliser des types détenus à l'image des types complexes dans EF6 :

```

modelBuilder.Entity<Order>().OwnsOne(p => p.OrderDetails, cb =>
{
    cb.OwnsOne(c => c.BillingAddress);
    cb.OwnsOne(c => c.ShippingAddress);
});

public class Order
{
    public int Id { get; set; }
    public OrderDetails OrderDetails { get; set; }
}

public class OrderDetails
{
    public StreetAddress BillingAddress { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}

public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}

```

Pour plus d'informations sur cette fonctionnalité, lisez la [section sur les types d'entités détenus](#).

### Filtres de requête au niveau du modèle

EF Core 2.0 inclut une nouvelle fonctionnalité que nous appelons « filtres de requête au niveau du modèle ». Cette fonctionnalité permet aux prédictats de requête LINQ (expression booléenne généralement passée à l'opérateur de requête LINQ Where) d'être définis directement sur des types d'entité dans le modèle de métadonnées (généralement dans OnModelCreating). Ces filtres sont automatiquement appliqués à toutes les requêtes LINQ impliquant ces types d'entités, y compris ceux référencés indirectement, par exemple à l'aide d'Include ou de références de propriété de navigation directe. Voici deux applications courantes de cette fonctionnalité :

- Suppression réversible : un type d'entité définit une propriété IsDeleted.
- Architecture multilocataire : un type d'entité définit une propriété TenantId.

Voici un exemple simple illustrant la fonctionnalité pour les deux scénarios répertoriés ci-dessus :

```

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    public int TenantId { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>().HasQueryFilter(
            p => !p.IsDeleted
            && p.TenantId == this.TenantId);
    }
}

```

Nous définissons un filtre au niveau du modèle qui implémente une architecture multilocataire et une suppression réversible pour des instances du type d'entité `Post`. Notez l'utilisation d'un `DbContext` propriété au niveau de l'instance : `TenantId`. Les filtres au niveau du modèle utilisent la valeur de l'instance de contexte correcte (c'est-à-dire celle qui exécute la requête).

Les filtres peuvent être désactivés pour des requêtes LINQ individuelles à l'aide de l'opérateur

`IgnoreQueryFilters()`.

#### Limitations

- Les références de navigation ne sont pas autorisées. Cette fonctionnalité peut être ajoutée en fonction de vos commentaires.
- Les filtres ne peuvent être définis que sur le type d'entité racine d'une hiérarchie.

#### Mappage de fonctions scalaires de base de données

EF Core 2.0 inclut une contribution importante de [Paul Middleton](#) qui permet de mapper les fonctions scalaires de base de données à des stubs de méthode en vue de les utiliser dans des requêtes LINQ et de les convertir en SQL.

Voici une brève description de l'utilisation de la fonctionnalité :

Déclarez une méthode statique sur votre `DbContext` et annotez-la avec `DbFunctionAttribute` :

```
public class BloggingContext : DbContext
{
    [DbFunction]
    public static int PostReadCount(int blogId)
    {
        throw new NotImplementedException();
    }
}
```

Les méthodes telles que celle-ci sont inscrites automatiquement. Une fois inscrits, les appels à la méthode dans une requête LINQ peuvent être traduits en appels de fonction dans SQL :

```
var query =
    from p in context.Posts
    where BloggingContext.PostReadCount(p.Id) > 5
    select p;
```

Quelques points à noter :

- Par Convention, le nom de la méthode est utilisé comme nom d'une fonction (dans ce cas, une fonction définie par l'utilisateur) lors de la génération du SQL, mais vous pouvez remplacer le nom et le schéma lors de l'inscription de la méthode.
- Actuellement, seules les fonctions scalaires sont prises en charge.
- Vous devez créer la fonction mappée dans la base de données. EF Core migrations ne prend pas en charge la création.

#### Configuration du type autonome pour Code First

Dans EF6, il était possible d'encapsuler la configuration Code First d'un type d'entité spécifique en effectuant une dérivation à partir `d'EntityTypeConfiguration`. Dans EF Core 2.0, nous reprenons ce modèle :

```
class CustomerConfiguration : IEntityTypeConfiguration<Customer>
{
    public void Configure(EntityTypeBuilder<Customer> builder)
    {
        builder.HasKey(c => c.AlternateKey);
        builder.Property(c => c.Name).HasMaxLength(200);
    }
}

...
// OnModelCreating
builder.ApplyConfiguration(new CustomerConfiguration());
```

# Performances élevées

## Regroupement DbContext

En règle générale, le modèle de base pour l'utilisation d'EF Core dans une application ASP.NET Core implique l'inscription d'un type DbContext personnalisé dans le système d'injection de dépendances, puis l'obtention d'instances de ce type par le biais de paramètres de constructeur dans les contrôleurs. Cela signifie qu'une instance de DbContext est créée pour chaque demande.

La version 2.0 offre une nouvelle façon d'inscrire des types DbContext personnalisés dans l'injection de dépendances, qui introduit en toute transparence un groupe d'instances de DbContext réutilisables. Pour utiliser le regroupement DbContext, utilisez `AddDbContextPool` au lieu de `AddDbContext` durant l'inscription des services :

```
services.AddDbContextPool<BlogginContext>()
    options => options.UseSqlServer(connectionString));
```

Si cette méthode est utilisée, au moment où une instance de DbContext est demandée par un contrôleur, nous vérifions d'abord s'il existe une instance disponible dans le pool. Une fois finalisé le traitement de la demande, tout état sur l'instance est réinitialisé et l'instance elle-même est retournée au pool.

Ce regroupement est conceptuellement semblable au regroupement de connexions dans les fournisseurs ADO.NET et présente l'avantage de réduire les coûts d'initialisation de l'instance de DbContext.

## Limitations

La nouvelle méthode présente quelques limitations quant à ce qui peut être effectué dans la méthode `OnConfiguring()` de DbContext.

### WARNING

Évitez d'utiliser le regroupement DbContext si vous gérez votre propre état (par exemple, champs privés) dans votre classe DbContext dérivée qui ne doit pas être partagée entre les requêtes. EF Core réinitialise uniquement l'état dont il a connaissance avant d'ajouter une instance de DbContext au pool.

## Requêtes compilées explicitement

Il s'agit de la deuxième fonctionnalité de gain de performance à utiliser au choix conçue pour offrir des avantages dans les scénarios à grande échelle.

Les API de requête manuelles ou compilées explicitement étaient disponibles dans les versions précédentes d'EF, ainsi que dans LINQ to SQL, pour permettre aux applications de mettre en cache la conversion des requêtes afin qu'elles puissent être calculées une seule fois et exécutées plusieurs fois.

Bien qu'en général EF Core puisse automatiquement compiler et mettre en cache les requêtes en fonction d'une représentation hachée des expressions de requête, ce mécanisme peut être utilisé pour obtenir un petit gain de performances en contournant le calcul du hachage et la recherche dans le cache, ce qui permet à l'application d'utiliser une requête déjà compilée par le biais de l'appel d'un délégué.

```

// Create an explicitly compiled query
private static Func<CustomerContext, int, Customer> _customerById =
    EF.CompileQuery((CustomerContext db, int id) =>
        db.Customers
            .Include(c => c.Address)
            .Single(c => c.Id == id));

// Use the compiled query by invoking it
using (var db = new CustomerContext())
{
    var customer = _customerById(db, 147);
}

```

## Suivi des modifications

### **Attach peut suivre un graphique des entités nouvelles et existantes**

EF Core prend en charge la génération automatique des valeurs de clés par le biais d'une variété de mécanismes. Quand vous utilisez cette fonctionnalité, une valeur est générée si la propriété de clé est la valeur par défaut du CLR, généralement zéro ou null. Cela signifie qu'un graphique d'entités peut être transmis à `DbContext.Attach` ou `DbSet.Attach` et qu'EF Core marque les entités qui ont déjà une clé définie comme `Unchanged` tandis que les entités qui n'ont pas de clé définie sont marquées comme `Added`. Il est ainsi plus facile de joindre un graphique regroupant des entités nouvelles et existantes quand des clés générées sont utilisées. `DbContext.Update` et `DbSet.Update` fonctionnent de la même manière, à la différence que les entités ayant une clé définie sont marquées comme `Modified` au lieu de `Unchanged`.

## Query

### **Traduction LINQ améliorée**

Permet l'exécution de davantage de requêtes, avec davantage de logique évaluée dans la base de données (plutôt qu'en mémoire) et moins de données inutilement récupérées de la base de données.

### **Améliorations apportées aux jonctions de groupes**

Le SQL généré pour les jonctions de groupes est amélioré. Les jonctions de groupes sont souvent le résultat de sous-requêtes sur des propriétés de navigation facultatives.

### **Interpolation de chaîne dans FromSql et ExecuteSqlCommand**

C# 6 a introduit l'interpolation de chaîne, fonctionnalité qui permet d'incorporer les expressions C# directement dans des littéraux de chaîne, offrant ainsi un moyen agréable de générer des chaînes au moment de l'exécution. Dans EF Core 2.0, nous avons ajouté une prise en charge spéciale pour les chaînes interpolées à nos deux API principales qui acceptent des chaînes SQL brutes : `FromSql` et `ExecuteSqlCommand`. Cette nouvelle prise en charge permet d'utiliser l'interpolation de chaîne en mode « sécurisé ». c'est-à-dire d'une façon qui protège contre les erreurs d'injection SQL courantes pouvant se produire pendant la construction dynamique du code SQL au moment de l'exécution.

Voici un exemple :

```

var city = "London";
var contactTitle = "Sales Representative";

using (var context = CreateContext())
{
    context.Set<Customer>()
        .FromSql($@"
            SELECT *
            FROM ""Customers"""
            WHERE ""City"" = {city} AND
                  ""ContactTitle"" = {contactTitle}")
        .ToArray();
}

```

Dans cet exemple, il existe deux variables incorporées dans la chaîne de format SQL. EF Core génère le code SQL suivant :

```

@p0='London' (Size = 4000)
@p1='Sales Representative' (Size = 4000)

SELECT *
FROM ""Customers"""
WHERE ""City"" = @p0
    AND ""ContactTitle"" = @p1

```

## EF.Functions.Like()

Nous avons ajouté la propriété `EF.Functions` qui peut être utilisée par EF Core ou des fournisseurs pour définir des méthodes mappées à des fonctions ou des opérateurs de base de données afin qu'elles puissent être appelées dans les requêtes LINQ. Le premier exemple de ce type de méthode est `Like()` :

```

var aCustomers =
    from c in context.Customers
    where EF.Functions.Like(c.Name, "a%")
    select c;

```

Notez que `Like()` est fourni avec une implémentation en mémoire, ce qui peut être pratique quand vous utilisez une base de données en mémoire ou que l'évaluation du prédicat doit se produire côté client.

## Gestion de base de données

### Crochet de pluralisation pour la génération de modèles de DbContext

EF Core 2.0 introduit un nouveau service `IPluralizer` qui est utilisé pour singulariser les noms des types d'entité et pluraliser les noms `DbSet`. L'implémentation par défaut étant l'absence d'opération, il s'agit simplement d'un dispositif permettant à l'utilisateur d'incorporer facilement son propre pluraliseur.

Voici à quoi cela ressemble dans le cas d'un développeur souhaitant mettre en place son propre pluraliseur :

```

public class MyDesignTimeServices : IDesignTimeServices
{
    public void ConfigureDesignTimeServices(IServiceCollection services)
    {
        services.AddSingleton<IPluralizer, MyPluralizer>();
    }
}

public class MyPluralizer : IPluralizer
{
    public string Pluralize(string name)
    {
        return Inflector.Inflector.Pluralize(name) ?? name;
    }

    public string Singularize(string name)
    {
        return Inflector.Inflector.Singularize(name) ?? name;
    }
}

```

## Autres

### Déplacement du fournisseur ADO.NET SQLite vers SQLitePCL.raw

Nous disposons ainsi d'une solution plus fiable dans Microsoft.Data.Sqlite pour distribuer des fichiers binaires SQLite natifs sur différentes plateformes.

### Un seul fournisseur par modèle

Renforce considérablement les possibilités d'interaction des fournisseurs avec le modèle et simplifie le fonctionnement des conventions, annotations et API Fluent avec les différents fournisseurs.

EF Core 2.0 génère désormais un [IModel](#) différent par fournisseur utilisé. Cela est généralement transparent pour l'application. Il en résulte une simplification des API de métadonnées de niveau inférieur, au point que tout accès aux *concepts de métadonnées relationnelles communs* est toujours établi par le biais d'un appel à `.Relational` au lieu de `.SqlServer`, `.Sqlite`, etc.

### Journalisation et diagnostics consolidés

Les mécanismes de journalisation (basés sur [ILogger](#)) et de diagnostics (basés sur [DiagnosticSource](#)) partagent désormais plus de code.

Les ID d'événement pour les messages envoyés à un [ILogger](#) ont été changés dans la version 2.0. Les ID d'événement sont maintenant uniques dans le code EF Core. En outre, ces messages suivent désormais le modèle standard de la journalisation structurée utilisé, par exemple, par le modèle MVC.

Les catégories d'enregistreurs d'événements ont également changé. Il existe désormais un jeu connu de catégories accessibles par le biais de [DbLoggerCategory](#).

Les événements [DiagnosticSource](#) utilisent désormais les mêmes noms d'ID d'événement que les messages [ILogger](#) correspondants.

# Nouvelles fonctionnalités d'EF Core 1.1

07/11/2019 • 2 minutes to read • [Edit Online](#)

## Modélisation

### Mappage de champs

Permet de configurer un champ de stockage pour une propriété. Cela peut être utile pour les propriétés en lecture seule ou les données utilisant les méthodes Get/Set au lieu d'une propriété.

### Mappage de tables à mémoire optimisée dans SQL Server

Vous pouvez spécifier que la table à laquelle est mappée une entité a une mémoire optimisée. Quand EF Core est utilisé pour créer et gérer une base de données basée sur votre modèle (avec des migrations ou `Database.EnsureCreated()`), une table à mémoire optimisée est créée pour ces entités.

## Change tracking

### API de suivi des modifications supplémentaires dans EF6

Par exemple : `Reload` , `GetModifiedProperties` , `GetDatabaseValues` , etc.

## Query

### Chargement explicite

Permet de déclencher le remplissage d'une propriété de navigation sur une entité qui a été précédemment chargée à partir de la base de données.

### DbSet.Find

Fournit un moyen simple de récupérer une entité en fonction de sa valeur de clé primaire.

## Autre

### Résilience de la connexion

Effectue automatiquement de nouvelles tentatives de commandes de base de données ayant échoué. Cela est particulièrement utile durant la connexion à SQL Azure, où les défaiillances passagères sont courantes.

### Remplacement de services simplifié

Facilite le remplacement de services internes utilisés par EF.

# Fonctionnalités incluses dans EF Core 1.0

07/11/2019 • 8 minutes to read • [Edit Online](#)

## Plateformes

### .NET Framework 4.5.1

Inclut la console, WPF, WinForms, ASP.NET 4, etc.

### .NET Standard 1.3

Inclut ASP.NET Core ciblant à la fois .NET Framework et .NET Core sur Windows, OSX et Linux.

## Modélisation

### Modélisation de base

Basée sur les entités OCT avec des propriétés get/set de types scalaires communs (`int`, `string`, etc.).

### Relations et propriétés de navigation

Les relations un-à-plusieurs et un-à-zéro-ou un-à-un peuvent être spécifiées dans le modèle en fonction d'une clé étrangère. Les propriétés de navigation de types de collection ou de référence simples peuvent être associées à ces relations.

### Conventions intégrées

Ces conventions génèrent un modèle initial basé sur la forme des classes d'entité.

### API Fluent

Permet de remplacer la méthode `OnModelCreating` sur votre contexte pour configurer davantage le modèle détecté par convention.

### Annotations de données

Il s'agit d'attributs qui peuvent être ajoutés à vos propriétés/classes d'entité et qui influenceront le modèle EF. Par exemple, l'ajout de la mention `[Required]` informera EF qu'une propriété est obligatoire.

### Mappage de tables relationnelles

Permet de mapper des entités à des tables ou des colonnes.

### Génération de valeur de clé

Inclut la génération côté client et la génération de base de données.

### Valeurs générées de base de données

Permet à la base de données de générer des valeurs par insertion (valeurs par défaut) ou par mise à jour (colonnes calculées).

### Séquences dans SQL Server

Permet de définir des objets de la séquence dans le modèle.

### Contraintes uniques

Permet de définir d'autres clés ainsi que les relations ciblant ces clés.

### Index

La définition d'index dans le modèle introduit automatiquement les index dans la base de données. Les index uniques sont également pris en charge.

## **Propriétés d'état de clichés instantanés**

Permet de définir dans le modèle des propriétés qui ne sont pas déclarées ni stockées dans la classe .NET, mais qui peuvent être suivies et mises à jour par EF Core. Elles sont couramment utilisées pour les propriétés de clé étrangère quand l'exposition de ces dernières dans l'objet n'est pas souhaitée.

## **Modèle d'héritage de table par hiérarchie**

Permet d'enregistrer les entités d'une hiérarchie d'héritage dans une table unique à l'aide d'une colonne de discriminateur pour identifier le type d'entité pour un enregistrement donné dans la base de données.

## **Validation de modèle**

Déetecte les modèles non valides dans le modèle et fournit des messages d'erreur utiles.

# Change tracking

## **Suivi des modifications par instantané**

Permet de détecter automatiquement les modifications apportées aux entités en comparant l'état actuel avec une copie (instantané) de l'état d'origine.

## **Suivi des modifications par notification**

Permet aux entités d'avertir le traceur de modifications dès que des valeurs de propriété sont modifiées.

## **État du suivi de l'accès**

Via `DbContext.Entry` et `DbContext.ChangeTracker`.

## **Attachement d'entités/graphes détachés**

La nouvelle API `DbContext.AttachGraph` permet de rattacher des entités à un contexte pour pouvoir enregistrer des entités modifiées ou de nouvelles entités.

# Enregistrement de données

## **Fonctionnalité d'enregistrement de base**

Permet de conserver les modifications apportées aux instances dans la base de données.

## **Accès concurrentiel optimiste**

Évite le remplacement de modifications apportées par un autre utilisateur, sachant que les données ont été extraites de la base de données.

## **SaveChanges asynchrone**

Peut libérer le thread actuel pour traiter d'autres requêtes pendant que la base de données traite les commandes émises à partir de `SaveChanges`.

## **Transactions de base de données**

Signifie que `SaveChanges` est toujours atomique (en d'autres termes, soit sa réussite est complète, soit aucune modification n'est apportée à la base de données). Il existe également des API liées aux transactions pour autoriser le partage de transactions entre des instances de contexte, etc.

## **Relationnel : traitement par lot d'instructions**

Offre de meilleures performances en regroupant les différentes commandes INSERT/UPDATE/DELETE dans une seule boucle pour la base de données.

# Query

## **Prise en charge de base de LINQ**

Offre la possibilité d'utiliser LINQ pour récupérer des données à partir de la base de données.

## **Évaluation du client/serveur mixte**

Permet aux requêtes de contenir la logique qui ne peut pas être évaluée dans la base de données et qui doit par conséquent être évaluée une fois les données récupérées dans la mémoire.

## **NoTracking**

Permet d'accélérer l'exécution des requêtes quand le contexte n'a pas besoin de surveiller les changements apportés aux instances d'entité (cela s'avère utile si les résultats en lecture seule).

## **Chargement hâtif**

Fournit les méthodes `Include` et `ThenInclude` pour identifier les données associées qui doivent également être extraites durant l'interrogation.

## **Requête asynchrone**

Peut libérer le thread actuel (et ses ressources associées) pour traiter d'autres requêtes pendant que la base de données traite la requête.

## **Requêtes SQL brutes**

Fournit la méthode `DbSet.FromSql` pour utiliser des requêtes SQL brutes pour extraire des données. Ces requêtes peuvent également être composées à l'aide de LINQ.

# Gestion du schéma de base de données

## **API de création/suppression de base de données**

Elles sont principalement conçues tester l'emplacement où vous souhaitez créer/supprimer rapidement la base de données sans utiliser de migrations.

## **Migrations de base de données relationnelle**

Permet à un schéma de base de données relationnelle d'évoluer à travers le temps au fur et à mesure que votre modèle change.

## **Ingénierie à rebours à partir de la base de données**

Permet de générer automatiquement un modèle EF basé sur un schéma de base de données relationnelle existante.

# Fournisseurs de bases de données

## **SQL Server**

Se connecte à Microsoft SQL Server 2008 et versions ultérieures.

## **SQLite**

Se connecte à une base de données SQLite 3.

## **En mémoire**

Fonctionnalité conçue pour tester facilement sans vous connecter à une base de données réelle.

## **Fournisseurs tiers**

Plusieurs fournisseurs sont disponibles pour d'autres moteurs de base de données. Consultez [Fournisseurs de bases de données](#) pour en obtenir la liste complète.

# Mise à niveau de EF Core 1,0 RC1 vers 1,0 RC2

11/10/2019 • 9 minutes to read

Cet article fournit des conseils pour déplacer une application générée avec les packages RC1 vers RC2.

## Noms et versions des packages

Entre RC1 et RC2, nous avons changé de « Entity Framework 7 » en « Entity Framework Core ». Pour plus d'informations sur les raisons de cette modification, consultez le [billet de blog de Scott Hanselman](#). En raison de cette modification, le nom de nos packages est passé de `EntityFramework.*` à `Microsoft.EntityFrameworkCore.*` et nos versions de `7.0.0-rc1-final` à `1.0.0-rc2-final` (ou `1.0.0-preview1-final` pour les outils).

**Vous devrez supprimer complètement les packages RC1, puis installer les packages RC2.** Voici le mappage de certains packages courants.

| PACKAGE RC1   | ÉQUIVALENT RC2   |
|---|--|
| EntityFramework. MicrosoftSqlServer 7.0.0-RC1-final         | Microsoft.EntityFrameworkCore.SqlServer 1.0.0-rc2-final        |
| EntityFramework. SQLite 7.0.0-RC1-final                     | Microsoft.EntityFrameworkCore.Sqlite 1.0.0-rc2-final           |
| EntityFramework7. npgsql 3.1.0-RC1-3                        | Npgsql.EntityFrameworkCore.Postgres                            |
| EntityFramework.SqlServerCompact35 7.0.0-rc1-final          | EntityFrameworkCore.SqlServerCompact35 1.0.0-rc2-final         |
| EntityFramework.SqlServerCompact40 7.0.0-rc1-final          | EntityFrameworkCore.SqlServerCompact40 1.0.0-rc2-final         |
| EntityFramework. InMemory 7.0.0-RC1-final                   | Microsoft. EntityFrameworkCore. InMemory 1.0.0-RC2-final       |
| EntityFramework. IBMDataserver 7.0.0-beta1                  | Pas encore disponible pour RC2                                 |
| EntityFramework. Commands 7.0.0-RC1-final                   | Microsoft. EntityFrameworkCore. Tools 1.0.0-Preview1-final     |
| EntityFramework. MicrosoftSqlServer. Design 7.0.0-RC1-final | Microsoft.EntityFrameworkCore.SqlServer.Design 1.0.0-rc2-final |

## Espaces de noms

Avec les noms de packages, les espaces de noms ont été modifiés de `Microsoft.Data.Entity.*` à `Microsoft.EntityFrameworkCore.*`. Vous pouvez gérer cette modification avec une recherche/remplacement de `using Microsoft.Data.Entity` avec `using Microsoft.EntityFrameworkCore`.

## Modification de la Convention d'affectation des noms de table

Une modification fonctionnelle significative que nous avons apportée à RC2 consistait à utiliser le nom de la propriété `DbSet< TEntity >` pour une entité donnée comme nom de table auquel elle est mappée, plutôt que simplement le nom de la classe. Pour plus d'informations sur ce changement, consultez [le numéro d'annonce associé](#).

Pour les applications RC1 existantes, nous vous recommandons d'ajouter le code suivant au début de votre

méthode `OnModelCreating` pour conserver la stratégie d'appellation RC1 :

```
foreach (var entity in modelBuilder.Model.GetEntityTypes())
{
    entity.Relational().TableName = entity.DisplayName();
}
```

Si vous souhaitez adopter la nouvelle stratégie de nommage, nous vous recommandons de terminer avec succès le reste des étapes de mise à niveau, puis de supprimer le code et de créer une migration pour appliquer les renommages de tables.

## Modifications de AddDbContext/Startup.cs (projets ASP.NET Core uniquement)

Dans RC1, vous deviez ajouter des services Entity Framework au fournisseur de services d'application-dans

`Startup.ConfigureServices(...)` :

```
services.AddEntityFramework()
    .AddSqlServer()
    .AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration["ConnectionStrings:DefaultConnection"]));

```

Dans RC2, vous pouvez supprimer les appels à `AddEntityFramework()`, `AddSqlServer()`, etc. :

```
services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(Configuration["ConnectionStrings:DefaultConnection"]));

```

Vous devez également ajouter un constructeur, à votre contexte dérivé, qui prend des options de contexte et les passe au constructeur de base. Cela est nécessaire, car nous avons supprimé une partie de la magie de l'glissé en arrière-plan :

```
public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
    : base(options)
{
}
```

## Passage d'un IServiceProvider

Si vous avez du code RC1 qui passe un `IServiceProvider` au contexte, cette valeur a été déplacée vers `DbContextOptions`, au lieu d'être un paramètre de constructeur séparé. Utilisez `DbContextOptionsBuilder.UseInternalServiceProvider(...)` pour définir le fournisseur de services.

### Test

Le scénario le plus courant pour effectuer cette tâche consistait à contrôler l'étendue d'une base de données InMemory lors du test. Pour obtenir un exemple de cette procédure avec RC2, consultez l'article sur les [tests](#) mis à jour.

## Résolution des services internes à partir du fournisseur de services d'application (projets ASP.NET Core uniquement)

Si vous disposez d'une application ASP.NET Core et que vous souhaitez que EF résolve les services internes à partir du fournisseur de services d'application, il existe une surcharge de `AddDbContext` qui vous permet de configurer ce qui suit :

```
services.AddEntityFrameworkSqlServer()
    .AddDbContext<ApplicationContext>((serviceProvider, options) =>
    options.UseSqlServer(Configuration["ConnectionStrings:DefaultConnection"])
        .UseInternalServiceProvider(serviceProvider));
```

#### WARNING

Nous vous recommandons d'autoriser EF à gérer ses propres services en interne, sauf si vous avez une raison d'associer les services EF internes à votre fournisseur de services d'application. La raison principale pour laquelle vous pouvez effectuer cette opération consiste à utiliser votre fournisseur de services d'application pour remplacer les services utilisés en interne par EF.

## Commandes DNX = > CLI .NET (projets ASP.NET Core uniquement)

Si vous avez précédemment utilisé les commandes `dnx ef` pour les projets ASP.NET 5, ceux-ci ont été déplacés vers les commandes de `dotnet ef`. La même syntaxe de commande s'applique toujours. Vous pouvez utiliser `dotnet ef --help` pour les informations de syntaxe.

La façon dont les commandes sont inscrites a changé en RC2, en raison du remplacement de DNX par l'interface CLI .NET. Les commandes sont maintenant inscrites dans une section `tools` dans `project.json` :

```
"tools": {
  "Microsoft.EntityFrameworkCore.Tools": {
    "version": "1.0.0-preview1-final",
    "imports": [
      "portable-net45+win8+dnxcore50",
      "portable-net45+win8"
    ]
  }
}
```

#### TIP

Si vous utilisez Visual Studio, vous pouvez désormais utiliser la console du gestionnaire de package pour exécuter des commandes EF pour les projets ASP.NET Core (cela n'était pas pris en charge dans RC1). Pour ce faire, vous devez toujours enregistrer les commandes dans la section `tools` de `project.json`.

## Les commandes du gestionnaire de package requièrent PowerShell 5

Si vous utilisez les commandes Entity Framework dans la console du gestionnaire de package dans Visual Studio, vous devez vous assurer que PowerShell 5 est installé. Il s'agit d'une exigence temporaire qui sera supprimée dans la prochaine version (pour plus d'informations, consultez [#5327 du problème](#)).

## Utilisation de « Imports » dans Project. JSON

Certaines des dépendances de EF Core ne prennent pas encore en charge .NET Standard. EF Core dans les projets .NET Standard et .NET Core peuvent nécessiter l'ajout de « Imports » à Project. JSON comme solution de contournement temporaire.

Lorsque vous ajoutez EF, la restauration NuGet affiche ce message d'erreur :

```
Package Ix-Async 1.2.5 is not compatible with netcoreapp1.0 (.NETCoreApp,Version=v1.0). Package Ix-Async 1.2.5
supports:
- net40 (.NETFramework,Version=v4.0)
- net45 (.NETFramework,Version=v4.5)
- portable-net45+win8+wp8 (.NETPortable,Version=v0.0,Profile=Profile78)
Package Remotion.Linq 2.0.2 is not compatible with netcoreapp1.0 (.NETCoreApp,Version=v1.0). Package
Remotion.Linq 2.0.2 supports:
- net35 (.NETFramework,Version=v3.5)
- net40 (.NETFramework,Version=v4.0)
- net45 (.NETFramework,Version=v4.5)
- portable-net45+win8+wp8+wpa81 (.NETPortable,Version=v0.0,Profile=Profile259)
```

La solution consiste à importer manuellement le profil portable « portable-net451 + WIN8 ». Cela force NuGet à traiter ces fichiers binaires qui correspondent à ce qui est fourni comme une infrastructure compatible avec .NET Standard, même s'ils ne le sont pas. Bien que « portable-net451 + WIN8 » ne soit pas compatible 100% avec .NET Standard, il est suffisamment compatible pour la transition de PCL vers .NET Standard. Les importations peuvent être supprimées lorsque les dépendances EF finissent par être mises à niveau vers .NET Standard.

Plusieurs infrastructures peuvent être ajoutées à « Imports » dans la syntaxe de tableau. D'autres importations peuvent être nécessaires si vous ajoutez des bibliothèques supplémentaires à votre projet.

```
{
  "frameworks": {
    "netcoreapp1.0": {
      "imports": [ "dnxcore50", "portable-net451+win8" ]
    }
  }
}
```

Consultez [#5176](#) de problèmes.

# Mise à niveau de EF Core 1.0 RC2 vers la version RTM

07/11/2019 • 5 minutes to read

Cet article fournit des conseils pour déplacer une application générée avec les packages RC2 vers 1.0.0 RTM.

## Versions de package

Les noms des packages de niveau supérieur que vous installez généralement dans une application n'ont pas changé entre RC2 et RTM.

### **Vous devez mettre à niveau les packages installés vers les versions RTM :**

- Les packages d'exécution (par exemple, `Microsoft.EntityFrameworkCore.SqlServer`) ont été modifiés de `1.0.0-rc2-final` à `1.0.0`.
- Le package de `Microsoft.EntityFrameworkCore.Tools` a été modifié de `1.0.0-preview1-final` à `1.0.0-preview2-final`. Notez que les outils sont toujours en version préliminaire.

## Des migrations existantes peuvent nécessiter l'ajout de maxLength

Dans RC2, la définition de colonne dans une migration a été recherchée comme

`table.Column<string>(nullable: true)` et la longueur de la colonne a été recherchée dans certaines métadonnées que nous stockons dans le code de la migration. Dans RTM, la longueur est désormais incluse dans le code de génération de modèles automatique `table.Column<string>(maxLength: 450, nullable: true)`.

L'argument `maxLength` n'est pas spécifié pour les migrations existantes qui ont été créées avant l'utilisation de RTM. Cela signifie que la longueur maximale prise en charge par la base de données sera utilisée (`nvarchar(max)` sur SQL Server). Cela peut s'avérer parfait pour certaines colonnes, mais les colonnes qui font partie d'une clé, d'une clé étrangère ou d'un index doivent être mises à jour pour inclure une longueur maximale. Par Convention, 450 est la longueur maximale utilisée pour les clés, les clés étrangères et les colonnes indexées. Si vous avez configuré explicitement une longueur dans le modèle, vous devez utiliser cette longueur à la place.

### **ASP.NET Identity**

Cette modification a un impact sur les projets qui utilisent ASP.NET Identity et ont été créés à partir d'un modèle de projet antérieur à la version RTM. Le modèle de projet comprend une migration utilisée pour créer la base de données. Cette migration doit être modifiée pour spécifier une longueur maximale de `256` pour les colonnes suivantes.

- **AspNetRoles**
  - Name
  - NormalizedName
- **AspNetUsers**
  - Messagerie
  - NormalizedEmail
  - NormalizedUserName
  - UserName

Si vous n'effectuez pas cette modification, l'exception suivante se produit lorsque la migration initiale est appliquée à une base de données.

```
System.Data.SqlClient.SqlException (0x80131904): Column 'NormalizedNome' in table 'AspNetRoles' is of a type  
that is invalid for use as a key column in an index.
```

## .NET Core : supprimer « Imports » dans Project. JSON

Si vous ciblez .NET Core avec RC2, vous deviez ajouter des `imports` à Project. JSON comme solution de contournement temporaire pour certaines des dépendances de EF Core ne prenant pas en charge .NET Standard. Vous pouvez maintenant les supprimer.

```
{  
  "frameworks": {  
    "netcoreapp1.0": {  
      "imports": ["dnxcore50", "portable-net451+win8"]  
    }  
  }  
}
```

### NOTE

Depuis la version 1,0 RTM, le [Kit SDK .net Core](#) ne prend plus en charge `project.json` ni le développement d'applications .net core à l'aide de Visual Studio 2015. Nous vous recommandons de [migrer de project.json vers csproj](#). Si vous utilisez Visual Studio, nous vous recommandons de mettre à niveau vers [Visual studio 2017](#).

## UWP : ajouter des redirections de liaison

Si vous tentez d'exécuter des commandes EF sur des projets plateforme Windows universelle (UWP), l'erreur suivante se produit :

```
System.IO.FileLoadException: Could not load file or assembly 'System.IO.FileSystem.Primitives, Version=4.0.0.0,  
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a' or one of its dependencies. The located assembly's manifest  
definition does not match the assembly reference.
```

Vous devez ajouter manuellement les redirections de liaison au projet UWP. Créez un fichier nommé `App.config` dans le dossier racine du projet, puis ajoutez des redirections aux versions d'assembly appropriées.

```
<configuration>  
  <runtime>  
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">  
      <dependentAssembly>  
        <assemblyIdentity name="System.IO.FileSystem.Primitives"  
          publicKeyToken="b03f5f7f11d50a3a"  
          culture="neutral" />  
        <bindingRedirect oldVersion="4.0.0.0"  
          newVersion="4.0.1.0"/>  
      </dependentAssembly>  
      <dependentAssembly>  
        <assemblyIdentity name="System.Threading.Overlapped"  
          publicKeyToken="b03f5f7f11d50a3a"  
          culture="neutral" />  
        <bindingRedirect oldVersion="4.0.0.0"  
          newVersion="4.0.1.0"/>  
      </dependentAssembly>  
    </assemblyBinding>  
  </runtime>  
</configuration>
```

# Mise à niveau des applications des versions précédentes vers EF Core 2,0

05/12/2019 • 15 minutes to read

Nous avons pris la possibilité d'affiner de manière significative nos API et comportements existants dans 2,0. Il existe quelques améliorations qui peuvent nécessiter la modification du code d'application existant, même si nous pensons que pour la majorité des applications, l'impact sera faible, dans la plupart des cas nécessitant juste une recompilation et des modifications guidées minimales pour remplacer les API obsolètes.

La mise à jour d'une application existante vers EF Core 2,0 peut nécessiter les éléments suivants :

1. Mise à niveau de l'implémentation .NET cible de l'application vers une implémentation qui prend en charge .NET Standard 2,0. Pour plus d'informations, consultez [implémentations .net prises en charge](#).
2. Identifiez un fournisseur pour la base de données cible qui est compatible avec EF Core 2,0. Consultez [EF Core 2,0 requiert un fournisseur de base de données 2,0](#) ci-dessous.
3. Mise à niveau de tous les packages de EF Core (Runtime et outils) vers 2,0. Pour plus d'informations, consultez [installation de EF Core](#).
4. Apportez les modifications nécessaires au code pour compenser les modifications avec rupture décrites dans le reste de ce document.

## ASP.NET Core comprend maintenant EF Core

Les applications ciblant ASP.NET Core 2.0 peuvent utiliser EF Core 2.0 sans dépendances supplémentaires en plus des fournisseurs de base de données tiers. Toutefois, les applications ciblant des versions antérieures de ASP.NET Core doivent effectuer une mise à niveau vers ASP.NET Core 2,0 afin d'utiliser EF Core 2,0. Pour plus d'informations sur la mise à niveau des applications ASP.NET Core vers 2,0, consultez [la documentation ASP.net Core sur le sujet](#).

## Nouvelle méthode d'obtention des services d'application dans ASP.NET Core

Le modèle recommandé pour les applications Web ASP.NET Core a été mis à jour pour 2,0 d'une manière qui a enfreint la logique au moment de la conception EF Core utilisée dans 1.x. Auparavant, lors de la conception, EF Core essaierait d'appeler `Startup.ConfigureServices` directement afin d'accéder au fournisseur de services de l'application. Dans ASP.NET Core 2,0, la configuration est initialisée en dehors de la classe `Startup`. Les applications qui utilisent EF Core généralement accéder à leur chaîne de connexion à partir de la configuration, de sorte que `Startup` n'est plus suffisant. Si vous mettez à niveau une application ASP.NET Core 1.x, vous risquez de recevoir l'erreur suivante lors de l'utilisation des outils de EF Core.

Aucun constructeur sans paramètre n'a été trouvé sur 'ApplicationContext'. Ajoutez un constructeur sans paramètre à 'ApplicationContext' ou ajoutez une implémentation de 'IDesignTimeDbContextFactory<ApplicationContext>' dans le même assembly que 'ApplicationContext'

Un nouveau Hook au moment du design a été ajouté au modèle par défaut de ASP.NET Core 2.0. La méthode `Program.BuildWebHost` statique permet à EF Core d'accéder au fournisseur de services de l'application au moment de la conception. Si vous mettez à niveau une application ASP.NET Core 1.x, vous devrez mettre à jour la classe `Program` pour qu'elle ressemble à ce qui suit.

```

using Microsoft.AspNetCore;
using Microsoft.AspNetCore.Hosting;

namespace AspNetCoreDotNetCore2._0App
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

        public static IWebHost BuildWebHost(string[] args) =>
            WebHost.CreateDefaultBuilder(args)
                .UseStartup<Startup>()
                .Build();
    }
}

```

L'adoption de ce nouveau modèle lors de la mise à jour des applications vers 2,0 est vivement recommandée et est nécessaire pour que des fonctionnalités de produit telles que Entity Framework Core migrations fonctionnent. L'autre alternative courante consiste à [implémenter `IDesignTimeDbContextFactory<TContext>`](#).

## IDbContextFactory renommée

Afin de prendre en charge divers modèles d'application et de permettre aux utilisateurs de mieux contrôler la façon dont leurs `DbContext` sont utilisés au moment de la conception, nous avons, par le passé, fourni l'interface `IDbContextFactory<TContext>`. Au moment du design, les outils de EF Core découvrent les implémentations de cette interface dans votre projet et l'utilisent pour créer des objets `DbContext`.

Cette interface a un nom très général qui induire certains utilisateurs à essayer de les réutiliser pour d'autres scénarios de création de `DbContext`. Ils ont été interceptés contre la protection lorsque les outils EF ont ensuite essayé d'utiliser leur implémentation au moment de la conception et ont provoqué l'échec des commandes comme `Update-Database` OU `dotnet ef database update`.

Pour pouvoir communiquer la sémantique forte au moment de la conception de cette interface, nous l'avons renommée en `IDesignTimeDbContextFactory<TContext>`.

Pour la version 2,0, le `IDbContextFactory<TContext>` existe toujours, mais est marqué comme obsolète.

## DbContextOptions supprimé

En raison des modifications apportées à l'ASP.NET Core 2,0 décrites ci-dessus, nous avons découvert que `DbContextOptions` n'était plus nécessaire sur la nouvelle interface `IDesignTimeDbContextFactory<TContext>`. Voici les alternatives que vous devez utiliser à la place.

| DBCONTEXTFACTORYOPTIONS | ALTERNATIVE   |
|-------------------------|---|
| ApplicationBasePath     | <code>ApplicationContext.BaseDirectory</code>                             |
| ContentRootPath         | <code>Directory.GetCurrentDirectory()</code>                              |
| EnvironmentName         | <code>Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT")</code> |

## Modification du répertoire de travail au moment du design

Les modifications apportées au ASP.NET Core 2,0 ont également nécessité le répertoire de travail utilisé par `dotnet ef` pour s'aligner avec le répertoire de travail utilisé par Visual Studio lors de l'exécution de votre application. L'un des effets secondaires observables est que les noms de fichiers SQLite sont désormais relatifs au répertoire du projet, et non au répertoire de sortie comme s'ils étaient utilisés.

## EF Core 2,0 requiert un fournisseur de base de données 2,0

Pour EF Core 2,0, nous avons apporté de nombreuses simplifications et améliorations au fonctionnement des fournisseurs de bases de données. Cela signifie que les fournisseurs 1.0.x et 1.1.x ne fonctionneront pas avec EF Core 2,0.

Les fournisseurs SQL Server et SQLite sont fournis par l'équipe EF et les versions 2,0 seront disponibles dans le cadre de la version 2,0. Les fournisseurs tiers Open source pour [SQL Compact](#), [PostgreSQL](#) et [MySQL](#) sont mis à jour pour 2,0. Pour tous les autres fournisseurs, contactez le rédacteur du fournisseur.

## Les événements de journalisation et de diagnostic ont changé

Remarque : ces modifications ne doivent pas avoir d'impact sur la plupart du code d'application.

Les ID d'événement pour les messages envoyés à un `ILogger` ont été modifiés dans 2,0. Les ID d'événement sont maintenant uniques dans le code EF Core. En outre, ces messages suivent désormais le modèle standard de la journalisation structurée utilisé, par exemple, par le modèle MVC.

Les catégories d'enregistreurs d'événements ont également changé. Il existe désormais un jeu connu de catégories accessibles par le biais de `DbLoggerCategory`.

Les événements `DiagnosticSource` utilisent désormais les mêmes noms d'ID d'événements que les messages `ILogger` correspondants. Les charges utiles d'événement sont tous des types nominaux dérivés de `EventData`.

Les ID d'événement, les types de charge utile et les catégories sont documentés dans les classes `CoreEventId` et `RelationalEventId`.

Les ID ont également été déplacés de `Microsoft.EntityFrameworkCore.Infrastructure` vers le nouvel espace de noms `Microsoft.EntityFrameworkCore.Diagnostics`.

## EF Core les modifications de l'API de métadonnées relationnelles

EF Core 2.0 génère désormais un `IModel` différent par fournisseur utilisé. Cela est généralement transparent pour l'application. Cela a facilité la simplification des API de métadonnées de niveau inférieur, de telle sorte que tout accès à des *concepts de métadonnées relationnelles communs* soit toujours effectué via un appel à `.Relational()` au lieu de `.SqlServer`, `.Sqlite`, etc. Par exemple, le code 1.1.x ressemble à ceci :

```
var tableName = context.Model.FindEntityType(typeof(User)).SqlServer().TableName;
```

Doit maintenant être écrit comme suit :

```
var tableName = context.Model.FindEntityType(typeof(User)).Relational().TableName;
```

Au lieu d'utiliser des méthodes comme `ForSqlServerToTable`, les méthodes d'extension sont désormais disponibles pour écrire du code conditionnel en fonction du fournisseur actuel en cours d'utilisation. Par exemple :

```
modelBuilder.Entity<User>().ToTable(
    Database.IsSqlServer() ? "SqlServerName" : "OtherName");
```

Notez que cette modification s'applique uniquement aux API/métadonnées définies pour *tous les* fournisseurs relationnels. L'API et les métadonnées restent les mêmes lorsqu'elles sont spécifiques à un seul fournisseur. Par exemple, les index cluster sont spécifiques à SQL Server, de sorte que `ForSqlServerIsClustered` et `.SqlServer().IsClustered()` doivent toujours être utilisés.

## Ne prenez pas le contrôle du fournisseur de services EF

EF Core utilise un `IServiceProvider` interne (un conteneur d'injection de dépendances) pour son implémentation interne. Les applications doivent autoriser EF Core à créer et à gérer ce fournisseur, sauf dans des cas spéciaux. Envisagez fortement de supprimer les appels à `UseInternalServiceProvider`. Si une application n'a pas besoin d'appeler `UseInternalServiceProvider`, envisagez de [soumettre un problème](#) afin que nous puissions étudier d'autres façons de gérer votre scénario.

L'appel de `AddEntityFramework`, `AddEntityFrameworkSqlServer`, etc. n'est pas requis par le code d'application, sauf si `UseInternalServiceProvider` est également appelé. Supprimez tous les appels existants à `AddEntityFramework` ou `AddEntityFrameworkSqlServer`, etc. `AddDbContext` doit toujours être utilisé de la même façon qu'auparavant.

## Les bases de données en mémoire doivent être nommées

La base de données en mémoire sans nom globale a été supprimée, mais toutes les bases de données en mémoire doivent être nommées. Par exemple :

```
optionsBuilder.UseInMemoryDatabase("MyDatabase");
```

Crée/utilise une base de données nommée « MyDatabase ». Si `UseInMemoryDatabase` est à nouveau appelée avec le même nom, la même base de données en mémoire est utilisée, ce qui lui permet d'être partagé par plusieurs instances de contexte.

## Modifications de l'API en lecture seule

`IsReadOnlyBeforeSave`, `IsReadOnlyAfterSave` et `IsStoreGeneratedAlways` ont été obsolètes et remplacés par `BeforeSaveBehavior` et `AfterSaveBehavior`. Ces comportements s'appliquent à n'importe quelle propriété (non seulement les propriétés générées par le magasin) et déterminent la façon dont la valeur de la propriété doit être utilisée lors de l'insertion dans une ligne de base de données (`BeforeSaveBehavior`) ou lors de la mise à jour d'une ligne de base de données existante (`AfterSaveBehavior`).

Les propriétés marquées comme `ValueGenerated.OnAddOrUpdate` (par exemple, pour les colonnes calculées) ignorent par défaut toute valeur actuellement définie sur la propriété. Cela signifie qu'une valeur générée par le magasin sera toujours obtenue, qu'une valeur ait été définie ou modifiée sur l'entité suivie. Vous pouvez modifier cette valeur en définissant une `Before\AfterSaveBehavior` différente.

## Nouveau comportement de suppression de ClientSetNull

Dans les versions précédentes, `DeleteBehavior.Restrict` avait un comportement pour les entités suivies par le contexte qui correspondait plus à la sémantique `SetNull`. Dans EF Core 2.0, un nouveau comportement de `ClientSetNull` a été introduit comme valeur par défaut pour les relations facultatives. Ce comportement a des `SetNull` sémantiques pour les entités suivies et le comportement des `Restrict` pour les bases de données créées à l'aide de EF Core. Dans notre expérience, il s'agit des comportements les plus attendus/utiles pour les entités suivies et la base de données. `DeleteBehavior.Restrict` est désormais respecté pour les entités suivies lorsqu'elles sont définies pour des relations facultatives.

## Packages au moment de la conception du fournisseur supprimés

Le package de `Microsoft.EntityFrameworkCore.Relational.Design` a été supprimé. Son contenu a été consolidé en `Microsoft.EntityFrameworkCore.Relational` et `Microsoft.EntityFrameworkCore.Design`.

Cela se propage dans les packages au moment de la conception du fournisseur. Ces packages (`Microsoft.EntityFrameworkCore.Sqlite.Design`, `Microsoft.EntityFrameworkCore.SqlServer.Design`, etc.) ont été supprimés et leur contenu est consolidé dans les packages de fournisseurs principaux.

Pour activer `Scaffold-DbContext` ou `dotnet ef dbcontext scaffold` dans EF Core 2.0, il vous suffit de référencer le package de fournisseur unique :

```
<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer"
    Version="2.0.0" />
<PackageReference Include="Microsoft.EntityFrameworkCore.Tools"
    Version="2.0.0"
    PrivateAssets="All" />
<DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
    Version="2.0.0" />
```

# Bien démarrer avec EF Core

10/01/2020 • 7 minutes to read • [Edit Online](#)

Dans ce tutoriel, vous allez créer une application console .NET Core qui effectue un accès aux données d'une base de données SQLite à l'aide d'Entity Framework Core.

Vous pouvez suivre le tutoriel à l'aide de Visual Studio sur Windows, ou à l'aide de l'interface CLI .NET Core sur Windows, macOS ou Linux.

[Affichez l'exemple proposé dans cet article sur GitHub.](#)

## Prérequis

Installez les logiciels suivants :

- [CLI .NET Core](#)
- [Visual Studio](#)
- [SDK .NET Core 3.0.](#)

## Créer un projet

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet new console -o EFGetStarted  
cd EFGetStarted
```

## Installer Entity Framework Core

Pour installer EF Core, installez le package pour le ou les fournisseurs de bases de données EF Core à cibler. Ce tutoriel utilise SQLite, car il s'exécute sur toutes les plateformes prises en charge par .NET Core. Pour obtenir la liste des fournisseurs disponibles, consultez [Fournisseurs de bases de données](#).

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
```

## Créer le modèle

Définissez une classe de contexte et des classes d'entité qui composent le modèle.

- [CLI .NET Core](#)
- [Visual Studio](#)
- Dans le répertoire du projet, créez **Model.cs** avec le code suivant.

```

using System.Collections.Generic;
using Microsoft.EntityFrameworkCore;

namespace EFGetStarted
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder options)
            => options.UseSqlite("Data Source=blogging.db");
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }

        public List<Post> Posts { get; } = new List<Post>();
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public Blog Blog { get; set; }
    }
}

```

EF Core peut également [rétroconcevoir](#) un modèle à partir d'une base de données existante.

**Conseil :** Dans une application réelle, vous placez chaque classe dans un fichier distinct et la [chaîne de connexion](#) dans un fichier de configuration ou une variable d'environnement. Pour simplifier le tutoriel, tout est placé dans un seul et même fichier.

## Créer la base de données

Les étapes suivantes utilisent des [migrations](#) pour créer une base de données.

- [CLI .NET Core](#)
- [Visual Studio](#)
- Exécutez les commandes suivantes :

```

dotnet tool install --global dotnet-ef
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet ef migrations add InitialCreate
dotnet ef database update

```

Cette action installe [dotnet ef](#) et le package de conception requis pour exécuter la commande sur un projet. La commande `migrations` structure une migration afin de créer le jeu initial de tables du modèle. La commande `database update` crée la base de données et lui applique la nouvelle migration.

## Créer, lire, mettre à jour et supprimer

- Ouvrez le fichier `Program.cs` et remplacez le contenu par le code suivant :

```

using System;
using System.Linq;

namespace EFGetStarted
{
    class Program
    {
        static void Main()
        {
            using (var db = new BloggingContext())
            {
                // Create
                Console.WriteLine("Inserting a new blog");
                db.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
                db.SaveChanges();

                // Read
                Console.WriteLine("Querying for a blog");
                var blog = db.Blogs
                    .OrderBy(b => b.BlogId)
                    .First();

                // Update
                Console.WriteLine("Updating the blog and adding a post");
                blog.Url = "https://devblogs.microsoft.com/dotnet";
                blog.Posts.Add(
                    new Post
                    {
                        Title = "Hello World",
                        Content = "I wrote an app using EF Core!"
                    });
                db.SaveChanges();

                // Delete
                Console.WriteLine("Delete the blog");
                db.Remove(blog);
                db.SaveChanges();
            }
        }
    }
}

```

## Exécuter l'application

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet run
```

## Étapes suivantes

- Suivre le [tutoriel ASP.NET Core](#) pour utiliser EF Core dans une application web
- Découvrir les [expressions de requête LINQ](#)
- [Configurer votre modèle](#) pour spécifier des éléments tels que la longueur **requise** et la longueur **maximale**
- Utiliser des [migrations](#) pour mettre à jour le schéma de base de données après avoir changé votre modèle

# Installation d'Entity Framework Core

05/12/2019 • 10 minutes to read • [Edit Online](#)

## Prérequis

- EF Core est une bibliothèque [.NET Standard 2.1](#). Ainsi, l'exécution d'EF Core nécessite une implémentation .NET prenant en charge .NET Standard 2.1. EF Core peut aussi être référencé par d'autres bibliothèques .NET Standard 2.1.
- Par exemple, vous pouvez utiliser EF Core pour développer des applications qui ciblent .NET Core. La création d'applications .NET Core nécessite le [SDK .NET Core](#). Vous pouvez aussi utiliser un environnement de développement comme [Visual Studio](#), [Visual Studio pour Mac](#) ou [Visual Studio Code](#). Pour plus d'informations, consultez [Bien démarrer avec .NET Core](#).
- Vous pouvez utiliser EF Core pour développer des applications sur Windows à l'aide de Visual Studio. La dernière version de [Visual Studio](#) est recommandée.
- EF Core peut s'exécuter sur d'autres implémentations .NET comme [Xamarin](#) et .NET Native. Toutefois, dans la pratique, ces implémentations ont des limitations de runtime qui peuvent affecter le fonctionnement d'EF Core sur votre application. Pour plus d'informations, consultez [Implémentations .NET prises en charge par EF Core](#).
- Enfin, différents fournisseurs de base de données peuvent nécessiter des versions de moteur de base de données, des implémentations .NET ou des systèmes d'exploitation spécifiques. Vérifiez qu'un [fournisseur de base de données EF Core](#) qui prend en charge l'environnement approprié pour votre application est disponible.

## Obtenir le runtime Entity Framework Core

Pour ajouter EF Core à une application, installez le package NuGet du fournisseur de base de données à utiliser.

Si vous créez une application ASP.NET Core, vous n'avez pas besoin d'installer les fournisseurs en mémoire et SQL Server. Ces fournisseurs sont inclus dans les versions actuelles d'ASP.NET Core, en même temps que le runtime EF Core.

Pour installer ou mettre à jour les packages NuGet, vous pouvez utiliser l'interface de ligne de commande (CLI) .NET Core, la boîte de dialogue Gestionnaire de package Visual Studio ou la console du Gestionnaire de package Visual Studio.

### CLI .NET Core

- Utilisez la commande CLI .NET Core suivante à partir de la ligne de commande du système d'exploitation pour installer ou mettre à jour le fournisseur EF Core SQL Server :

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

- Vous pouvez indiquer une version spécifique dans la commande `dotnet add package`, à l'aide du modificateur `-v`. Par exemple, pour installer des packages EF Core 2.2.0, ajoutez `-v 2.2.0` à la commande.

Pour plus d'informations, consultez [Outils de l'interface de ligne de commande \(CLI\) .NET](#).

### Boîte de dialogue Gestionnaire de package NuGet Visual Studio

- Dans le menu Visual Studio, sélectionnez **Projet > Gérer les packages NuGet**

- Cliquez sur l'onglet **Parcourir ou Mises à jour**.
- Pour installer ou mettre à jour le fournisseur SQL Server, sélectionnez le package `Microsoft.EntityFrameworkCore.SqlServer` et confirmez.

Pour plus d'informations, consultez [Boîte de dialogue Gestionnaire de package NuGet](#).

### Console du Gestionnaire de package NuGet Visual Studio

- Dans le menu Visual Studio, sélectionnez **Outils > Gestionnaire de package NuGet > Console du Gestionnaire de package**
- Pour installer le fournisseur SQL Server, exéutez la commande suivante dans la Console du Gestionnaire de package :

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

- Pour mettre à jour le fournisseur, utilisez la commande `Update-Package`.
- Pour indiquer une version spécifique, utilisez le modificateur `-Version`. Par exemple, pour installer les packages EF Core 2.2.0, ajoutez `-Version 2.2.0` aux commandes

Pour plus d'informations, consultez [Console Gestionnaire de package](#).

## Obtenir les outils Entity Framework Core

Vous pouvez installer des outils pour effectuer des tâches liées à EF Core dans votre projet, comme créer et appliquer des migrations de base de données, ou créer un modèle EF Core basé sur une base de données existante.

Deux ensembles d'outils sont disponibles :

- Les [outils de l'interface de ligne de commande \(CLI\) .NET Core](#) peuvent être utilisés sur Windows, Linux ou macOS. Ces commandes commencent par `dotnet ef`.
- Les [outils de la console du Gestionnaire de package](#) s'exécutent dans Visual Studio sur Windows. Ces commandes commencent par un verbe, par exemple `Add-Migration`, `Update-Database`.

Même si vous pouvez utiliser les commandes `dotnet ef` à partir de la console du Gestionnaire de package, nous vous recommandons d'utiliser les outils de la console du Gestionnaire de package avec Visual Studio :

- Ils fonctionnent automatiquement avec le projet sélectionné dans la console du Gestionnaire de package dans Visual Studio, sans que vous ayez besoin de changer manuellement les répertoires.
- Elles ouvrent automatiquement les fichiers générés par les commandes de Visual Studio après avoir été exécutées.

### Obtenir les outils CLI .NET Core

Les outils CLI .NET Core nécessitent le SDK .NET Core, mentionné précédemment dans les [Prérequis](#).

Les commandes `dotnet ef` sont incluses dans les versions actuelles du SDK .NET Core, mais pour les activer sur un projet spécifique, vous devez installer le package `Microsoft.EntityFrameworkCore.Design` :

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

Pour les applications ASP.NET Core, ce package est inclus automatiquement.

## IMPORTANT

Utilisez toujours la version des packages d'outils qui correspond à la version principale des packages du runtime.

### Obtenir les outils de la console du Gestionnaire de package

Pour obtenir les outils de la console du Gestionnaire de package pour EF Core, installez le package `Microsoft.EntityFrameworkCore.Tools`. Par exemple, à partir de Visual Studio :

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

Pour les applications ASP.NET Core, ce package est inclus automatiquement.

## Mise à niveau vers la dernière version EF Core

- Chaque fois que nous publions une nouvelle version d'EF Core, nous publions aussi une nouvelle version des fournisseurs qui font partie du projet EF Core, comme `Microsoft.EntityFrameworkCore.SqlServer`, `Microsoft.EntityFrameworkCore.Sqlite` et `Microsoft.EntityFrameworkCore.InMemory`. Vous pouvez simplement passer à la nouvelle version du fournisseur pour obtenir toutes les améliorations.
- EF Core ainsi que les fournisseurs en mémoire et SQL Server sont inclus dans les versions actuelles d'ASP.NET Core. Pour mettre à niveau une application ASP.NET Core existante vers une version plus récente d'EF Core, mettez toujours à niveau la version d'ASP.NET Core.
- Si vous devez mettre à jour une application qui utilise un fournisseur de base de données tiers, recherchez toujours une mise à jour du fournisseur qui est compatible avec la version d'EF Core à utiliser. Par exemple, les fournisseurs de base de données pour les versions antérieures ne sont pas compatibles avec la version 2.0 du runtime EF Core.
- Les fournisseurs tiers d'EF Core ne publient généralement pas de versions correctives en même temps que le runtime EF Core. Pour mettre à niveau une application qui utilise un fournisseur tiers vers une version corrective d'EF Core, vous devez peut-être ajouter une référence directe à des composants d'exécution EF Core individuels, comme `Microsoft.EntityFrameworkCore` et `Microsoft.EntityFrameworkCore.Relational`.
- Si vous mettez à niveau une application existante vers la dernière version d'EF Core, vous pouvez être amené à supprimer manuellement des références à des packages EF Core plus anciens :
  - Les packages design-time des fournisseurs de base de données comme `Microsoft.EntityFrameworkCore.SqlServer.Design` ne sont plus obligatoires ou pris en charge dans EF Core 2.0, mais ne sont pas automatiquement supprimés durant la mise à niveau des autres packages.
  - Les outils CLI .NET étant inclus dans le SDK .NET depuis la version 2.1, la référence à ce package peut être supprimée du fichier projet :

```
<DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.0" />
```

# Chaînes de connexion

20/09/2019 • 4 minutes to read

La plupart des fournisseurs de base de données requièrent une forme de chaîne de connexion pour se connecter à la base de données. Parfois, cette chaîne de connexion contient des informations sensibles qui doivent être protégées. Vous devrez peut-être également modifier la chaîne de connexion lorsque vous déplacez votre application entre des environnements, tels que le développement, le test et la production.

## WinForms & les applications WPF

Les applications WinForms, WPF et ASP.NET 4 ont un modèle de chaîne de connexion essayé et testé. La chaîne de connexion doit être ajoutée au fichier app.config de votre application (Web.config si vous utilisez ASP.NET). Si votre chaîne de connexion contient des informations sensibles, telles que le nom d'utilisateur et le mot de passe, vous pouvez protéger le contenu du fichier de configuration à l'aide de l' [outil Gestionnaire de secret](#).

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

    <connectionStrings>
        <add name="BloggingDatabase"
            connectionString="Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;" />
    </connectionStrings>
</configuration>
```

### TIP

Le `providerName` paramètre n'est pas requis sur les chaînes de connexion EF Core stockées dans App.config, car le fournisseur de base de données est configuré par le biais du code.

Vous pouvez ensuite lire la chaîne de connexion à `ConfigurationManager` l'aide de l'API `OnConfiguring` dans la méthode de votre contexte. Vous devrez peut-être ajouter une référence à `System.Configuration` l'assembly de Framework pour pouvoir utiliser cette API.

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {

        optionsBuilder.UseSqlServer(ConfigurationManager.ConnectionStrings["BloggingDatabase"].ConnectionString);
    }
}
```

## Plateforme Windows universelle (UWP)

Les chaînes de connexion dans une application UWP sont généralement une connexion SQLite qui spécifie simplement un nom de fichier local. En général, elles ne contiennent pas d'informations sensibles et n'ont pas besoin d'être modifiées lorsqu'une application est déployée. Par conséquent, ces chaînes de connexion sont généralement confinées dans le code, comme indiqué ci-dessous. Si vous souhaitez les déplacer hors du code, UWP

prend en charge le concept de paramètres. Pour plus d'informations, consultez la [section paramètres de l'application dans la documentation UWP](#).

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=blogging.db");
    }
}
```

## ASP.NET Core

Dans ASP.net core le système de configuration est très flexible, et la chaîne de connexion peut être stockée dans `appsettings.json`, une variable d'environnement, le magasin des secrets de l'utilisateur ou une autre source de configuration. Pour plus d'informations, consultez la [section Configuration de la documentation de ASP.net Core](#). L'exemple suivant illustre la chaîne de connexion stockée dans `appsettings.json`.

```
{
  "ConnectionStrings": {
    "BloggingDatabase": "Server=(localdb)\\mssqllocaldb;Database=EFGetStarted.ConsoleApp.NewDb;Trusted_Connection=True;"
  },
}
```

Le contexte est généralement configuré dans `Startup.cs` avec la chaîne de connexion lue à partir de la configuration. Remarque la `GetConnectionString()` méthode recherche une valeur de configuration dont la clé `ConnectionStrings:<connection string name>` est. Vous devez importer l'espace de noms [Microsoft.Extensions.Configuration](#) pour utiliser cette méthode d'extension.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<BloggingContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("BloggingDatabase")));
}
```

# Journalisation

09/01/2020 • 4 minutes to read

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Applications ASP.NET Core

EF Core s'intègre automatiquement avec les mécanismes de journalisation de ASP.NET Core chaque fois que `AddDbContext` ou `AddDbContextPool` est utilisé. Par conséquent, lors de l'utilisation de ASP.NET Core, la journalisation doit être configurée comme décrit dans la [documentation de ASP.net Core](#).

## Autres applications

EF Core la journalisation requiert un `ILoggerFactory` qui est lui-même configuré avec un ou plusieurs fournisseurs de journalisation. Les fournisseurs courants sont fournis dans les packages suivants :

- [Microsoft.extensions.Logging.console](#): un simple journal de console.
- [Microsoft.extensions.Logging.AzureAppServices](#): prend en charge les fonctionnalités « journaux de diagnostic » et « flux de journal » des Services de Azure App.
- [Microsoft.extensions.Logging.Debug](#): se connecte à un moniteur du débogueur à l'aide de `System.Diagnostics.Debug.WriteLine()`.
- [Microsoft.extensions.Logging.EventLog](#): enregistre dans le journal des événements Windows.
- [Microsoft.extensions.Logging.EventSource](#): prend en charge `EventSource`/`EventListener`.
- [Microsoft.extensions.Logging.TraceSource](#): se connecte à un écouteur de suivi à l'aide de `System.Diagnostics.TraceSource.TraceEvent()`.

Après l'installation du ou des packages appropriés, l'application doit créer une instance singleton/globale d'un `LoggerFactory`. Par exemple, à l'aide de l'enregistreur d'événements de console :

- [Version 3,0](#)
- [Version 2.x](#)

```
public static readonly ILoggerFactory MyLoggerFactory
    = LoggerFactory.Create(builder => { builder.AddConsole(); });
```

Cette instance de Singleton/global doit ensuite être inscrite auprès de EF Core sur le `DbContextOptionsBuilder`. Par exemple :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder
        .UseLoggerFactory(MyLoggerFactory) // Warning: Do not create a new ILoggerFactory instance each time
        .UseSqlServer(
            @"Server=(localdb)\mssqllocaldb;Database=EFLogging;Trusted_Connection=True;ConnectRetryCount=0");
```

## WARNING

Il est très important que les applications ne créent pas une nouvelle instance `ILoggerFactory` pour chaque instance de contexte. Cela entraînera une fuite de mémoire et des performances médiocres.

## Filtrage des éléments consignés

L'application peut contrôler ce qui est enregistré en configurant un filtre sur le `ILoggerProvider`. Par exemple :

- [Version 3,0](#)
- [Version 2.x](#)

```
public static readonly ILoggerFactory MyLoggerFactory
    = LoggerFactory.Create(builder =>
{
    builder
        .AddFilter((category, level) =>
            category == DbLoggerCategory.Database.Command.Name
                && level == LogLevel.Information)
        .AddConsole();
});
```

Dans cet exemple, le journal est filtré pour renvoyer uniquement les messages :

- dans la catégorie « Microsoft.EntityFrameworkCore.Database.Command »
- au niveau « information »

Par EF Core, les catégories de journalisation sont définies dans la classe `DbLoggerCategory` pour faciliter la recherche des catégories, mais elles sont résolues en chaînes simples.

Pour plus d'informations sur l'infrastructure de journalisation sous-jacente, consultez la [documentation ASP.net Core Logging](#).

# Résilience des connexions

21/02/2019 • 9 minutes to read

Résilience des connexions retente automatiquement les commandes de base de données ayant échoué. La fonctionnalité peut être utilisée avec une base de données en fournissant une « stratégie d'exécution », qui encapsule la logique nécessaire pour détecter les erreurs et réessayez les commandes. Fournisseurs EF Core peuvent fournir des stratégies d'exécution adaptées à leurs conditions d'échec de la base de données spécifique et les stratégies de nouvelle tentative optimal.

Par exemple, le fournisseur SQL Server inclut une stratégie d'exécution qui est conçu spécialement pour SQL Server (y compris SQL Azure). Il connaît les types d'exception qui peuvent être retentées et a des valeurs par défaut sensibles pour le nombre maximal de tentatives, délai entre les nouvelles tentatives, etc.

Une stratégie d'exécution est spécifiée lorsque vous configurez les options pour votre contexte. Il s'agit généralement dans le `OnConfiguring` méthode de votre contexte dérivé :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer(
            @"Server=
(localdb)\mssqllocaldb;Database=EFMiscellaneous.ConnectionResiliency;Trusted_Connection=True;ConnectRetryCount=0",
            options => options.EnableRetryOnFailure());
}
```

ou dans `Startup.cs` pour une application ASP.NET Core :

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<PicnicContext>(
        options => options.UseSqlServer(
            "<connection string>",
            providerOptions => providerOptions.EnableRetryOnFailure()));
}
```

## Stratégie d'exécution personnalisée

Il existe un mécanisme pour inscrire une stratégie d'exécution personnalisée de votre choix si vous souhaitez modifier les valeurs par défaut.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseMyProvider(
            "<connection string>",
            options => options.ExecutionStrategy(...));
}
```

## Transactions et les stratégies d'exécution

Une stratégie d'exécution qui réessaie automatiquement en cas d'échec doit être en mesure de lire chaque

opération dans un bloc de nouvelle tentative échoue. Lorsque de nouvelles tentatives sont activées, chaque opération que vous effectuez par le biais de EF Core devient sa propre opération de nouvelle tentative. Autrement dit, chaque requête et chaque appel à `SaveChanges()` va être retentée en tant qu'unité si une défaillance passagère se produit.

Toutefois, si votre code lance une transaction à l'aide de `BeginTransaction()` vous définissez votre propre groupe d'opérations qui doivent être traités en tant qu'unité, et tous les éléments à l'intérieur de la transaction doit être lu une défaillance intervient. Vous recevrez une exception semblable à ce qui suit si vous essayez d'effectuer lors de l'utilisation d'une stratégie d'exécution :

```
InvalidOperationException: La stratégie d'exécution configurée « SqlServerRetryingExecutionStrategy » ne prend pas en charge les transactions lancées par l'utilisateur. Utilisez la stratégie d'exécution retournée par « DbContext.Database.CreateExecutionStrategy() » pour exécuter toutes les opérations de la transaction en tant qu'ensemble pouvant être retenté.
```

La solution consiste à appeler manuellement la stratégie d'exécution avec un délégué représentant tout ce qui doit être exécutée. Si une défaillance passagère se produit, la stratégie d'exécution appelle à nouveau le délégué.

```
using (var db = new BloggingContext())
{
    var strategy = db.Database.CreateExecutionStrategy();

    strategy.Execute(() =>
    {
        using (var context = new BloggingContext())
        {
            using (var transaction = context.Database.BeginTransaction())
            {
                context.Blogs.Add(new Blog {Url = "http://blogs.msdn.com/dotnet"});
                context.SaveChanges();

                context.Blogs.Add(new Blog {Url = "http://blogs.msdn.com/visualstudio"});
                context.SaveChanges();

                transaction.Commit();
            }
        });
    });
}
```

Cette approche peut également être utilisée avec les transactions ambiantes.

```

using (var context1 = new BloggingContext())
{
    context1.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/visualstudio" });

    var strategy = context1.Database.CreateExecutionStrategy();

    strategy.Execute(() =>
    {
        using (var context2 = new BloggingContext())
        {
            using (var transaction = new TransactionScope())
            {
                context2.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
                context2.SaveChanges();

                context1.SaveChanges();

                transaction.Complete();
            }
        }
    });
}

```

## Échec de validation de transaction et le problème de l'idempotence

En règle générale, en cas d'échec de la connexion la transaction en cours est annulée. Toutefois, si la connexion est perdue pendant la transaction étant validées résultant état de la transaction est inconnu. Consultez ce [billet de blog](#) pour plus d'informations.

Par défaut, la stratégie d'exécution va retenter l'opération comme si la transaction a été annulée, mais si elle n'est pas le cas cela entraîne une exception si le nouvel état de la base de données n'est pas compatible ou pourrait entraîner **une altération des données** si le opération ne repose pas sur un état particulier, par exemple lors de l'insertion d'une nouvelle ligne avec les valeurs de clé générée automatiquement.

Il existe plusieurs façons de gérer cela.

### Option 1 - (presque) nothing

La probabilité d'un échec de connexion lors de la validation de transaction est faible, elle peut être acceptable pour votre application de simplement échouer si cette condition se produit réellement.

Toutefois, vous devez éviter d'utiliser des clés générées par le magasin afin de garantir qu'une exception est levée au lieu d'ajouter une ligne en double. Envisagez d'utiliser une valeur GUID générée par le client ou un générateur de valeur de côté client.

### Option 2 : état de l'application reconstruction

1. Ignorer actuel `DbContext`.
2. Créer un nouveau `DbContext` et restaurer l'état de votre application à partir de la base de données.
3. Informer l'utilisateur que la dernière opération ne peut-être pas terminée avec succès.

### Option 3 : ajouter la vérification de l'état

Pour la plupart des opérations qui modifient l'état de la base de données, il est possible d'ajouter du code qui vérifie si elle a réussi. Entity Framework fournit une méthode d'extension pour simplifier ce processus - `IExecutionStrategy.ExecuteInTransaction`.

Cette méthode commence une transaction est validée et accepte également une fonction dans le `verifySucceeded` paramètre qui est appelé lorsqu'une erreur temporaire se produit pendant la validation de transaction.

```

using (var db = new BloggingContext())
{
    var strategy = db.Database.CreateExecutionStrategy();

    var blogToAdd = new Blog {Url = "http://blogs.msdn.com/dotnet"};
    db.Blogs.Add(blogToAdd);

    strategy.ExecuteInTransaction(db,
        operation: context =>
    {
        context.SaveChanges(acceptAllChangesOnSuccess: false);
    },
    verifySucceeded: context => context.Blogs.AsNoTracking().Any(b => b.BlogId == blogToAdd.BlogId));

    db.ChangeTracker.AcceptAllChanges();
}

```

#### NOTE

Ici `SaveChanges` est appelé avec `acceptAllChangesOnSuccess` définie sur `false` afin d'éviter la modification de l'état de la `Blog` entité à `Unchanged` si `SaveChanges` réussit. Cela permet de retenter l'opération même si la validation échoue et la transaction est restaurée.

#### Option 4 : suivre manuellement la transaction

Si vous avez besoin d'utiliser des clés générées par le magasin ou besoin d'un moyen générique de gestion des échecs de validation qui ne dépend pas de l'opération effectuée chaque transaction peut être attribuée à un ID qui est vérifié lors de la validation échoue.

1. Ajouter une table à la base de données utilisé pour suivre l'état des transactions.
2. Insérer une ligne dans la table au début de chaque transaction.
3. Si la connexion échoue lors de la validation, vérifier la présence de la ligne correspondante dans la base de données.
4. Si la validation est réussie, supprimez la ligne correspondante pour éviter la croissance de la table.

```

using (var db = new BloggingContext())
{
    var strategy = db.Database.CreateExecutionStrategy();

    db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });

    var transaction = new TransactionRow { Id = Guid.NewGuid() };
    db.Transactions.Add(transaction);

    strategy.ExecuteInTransaction(db,
        operation: context =>
    {
        context.SaveChanges(acceptAllChangesOnSuccess: false);
    },
    verifySucceeded: context => context.Transactions.AsNoTracking().Any(t => t.Id == transaction.Id));

    db.ChangeTracker.AcceptAllChanges();
    db.Transactions.Remove(transaction);
    db.SaveChanges();
}

```

**NOTE**

Vérifiez que le contexte utilisé pour la vérification a une stratégie d'exécution définie comme la connexion est susceptible d'échouer lors de la vérification en cas d'échec lors de la validation de transaction.

# Tests de composants avec EF Core

07/11/2019 • 2 minutes to read

Vous pouvez tester les composants en simulant plus ou moins une connexion à la base de données réelle, sans la surcharge liée aux opérations d'E/S réelles de la base de données.

Pour ce faire, vous disposez de deux options :

- Le [mode en mémoire SQLite](#) vous permet d'écrire des tests efficaces par rapport à un fournisseur qui se comporte comme une base de données relationnelle.
- Le [fournisseur InMemory](#) est un fournisseur léger qui a des dépendances minimales, mais qui ne se comporte pas toujours comme une base de données relationnelle.

# Test avec SQLite

14/04/2019 • 5 minutes to read

SQLite a un mode en mémoire qui vous permet d'utiliser SQLite pour écrire des tests par rapport à une base de données relationnelle, sans la surcharge des opérations de base de données réelle.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub

## Exemple de scénario de test

Envisagez le service suivant qui permet au code d'application effectuer certaines opérations liées aux blogs. Il utilise en interne un `DbContext` qui se connecte à une base de données SQL Server. Il serait utile pour le remplacement de ce contexte pour se connecter à une base de données en mémoire SQLite afin que nous pouvons écrire des tests efficaces pour ce service sans avoir à modifier le code, ou vous pouvez faire beaucoup de travail pour créer un test double du contexte.

```
using System.Collections.Generic;
using System.Linq;

namespace BusinessLogic
{
    public class BlogService
    {
        private BloggingContext _context;

        public BlogService(BloggingContext context)
        {
            _context = context;
        }

        public void Add(string url)
        {
            var blog = new Blog { Url = url };
            _context.Blogs.Add(blog);
            _context.SaveChanges();
        }

        public IEnumerable<Blog> Find(string term)
        {
            return _context.Blogs
                .Where(b => b.Url.Contains(term))
                .OrderBy(b => b.Url)
                .ToList();
        }
    }
}
```

## Préparer votre contexte

### Évitez de configurer deux fournisseurs de base de données

Dans vos tests, vous vous apprêtez à configurer en externe le contexte pour utiliser le fournisseur en mémoire. Si vous configurez un fournisseur de base de données en substituant `OnConfiguring` dans votre contexte, vous devez

ensuite ajouter du code conditionnel afin de configurer le fournisseur de base de données uniquement si elle n'a pas déjà été configurée.

#### TIP

Si vous utilisez ASP.NET Core, vous devez inutile ce code dans la mesure où votre fournisseur de base de données est configuré en dehors du contexte (dans Startup.cs).

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=EFProviders.InMemory;Trusted_Connection=True;ConnectRetryCount=0");
    }
}
```

#### Ajoutez un constructeur pour le test

Pour activer le test par rapport à une autre base de données le plus simple consiste à modifier votre contexte pour exposer un constructeur qui accepte un `DbContextOptions<TContext>`.

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    { }

    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }
```

#### TIP

`DbContextOptions<TContext>` Indique le contexte à tous ses paramètres, tels que la base de données pour se connecter à. Il s'agit de l'objet qui est généré en exécutant la méthode `OnConfiguring` dans votre contexte.

## Écriture de tests

La clé à tester avec ce fournisseur est la possibilité d'indiquer le contexte à utiliser SQLite et contrôler la portée de la base de données en mémoire. L'étendue de la base de données est contrôlé par l'ouverture et fermeture de la connexion. La base de données est limitée à la durée pendant laquelle la connexion est ouverte. En général, vous souhaitez une nouvelle base de données pour chaque méthode de test.

#### TIP

Pour utiliser `SqliteConnection()` et `.UseSqlite()` référence le package NuGet, méthode d'extension [Microsoft.EntityFrameworkCore.Sqlite](#).

```
using BusinessLogic;
using Microsoft.Data.Sqlite;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using Xunit;

namespace EFTesting.TestProject.SQLite
```

```

{
    public class BlogServiceTests
    {
        [Fact]
        public void Add_writes_to_database()
        {
            // In-memory database only exists while the connection is open
            var connection = new SqliteConnection("DataSource=:memory:");
            connection.Open();

            try
            {
                var options = new DbContextOptionsBuilder<BloggingContext>()
                    .UseSqlite(connection)
                    .Options;

                // Create the schema in the database
                using (var context = new BloggingContext(options))
                {
                    context.Database.EnsureCreated();
                }

                // Run the test against one instance of the context
                using (var context = new BloggingContext(options))
                {
                    var service = new BlogService(context);
                    service.Add("https://example.com");
                    context.SaveChanges();
                }

                // Use a separate instance of the context to verify correct data was saved to database
                using (var context = new BloggingContext(options))
                {
                    Assert.Equal(1, context.Blogs.Count());
                    Assert.Equal("https://example.com", context.Blogs.Single().Url);
                }
            }
            finally
            {
                connection.Close();
            }
        }

        [Fact]
        public void Find_searches_url()
        {
            // In-memory database only exists while the connection is open
            var connection = new SqliteConnection("DataSource=:memory:");
            connection.Open();

            try
            {
                var options = new DbContextOptionsBuilder<BloggingContext>()
                    .UseSqlite(connection)
                    .Options;

                // Create the schema in the database
                using (var context = new BloggingContext(options))
                {
                    context.Database.EnsureCreated();
                }

                // Insert seed data into the database using one instance of the context
                using (var context = new BloggingContext(options))
                {
                    context.Blogs.Add(new Blog { Url = "https://example.com/cats" });
                    context.Blogs.Add(new Blog { Url = "https://example.com/catfish" });
                    context.Blogs.Add(new Blog { Url = "https://example.com/dogs" });
                    context.SaveChanges();
                }
            }
        }
    }
}

```

```
}

// Use a clean instance of the context to run the test
using (var context = new BloggingContext(options))
{
    var service = new BlogService(context);
    var result = service.Find("cat");
    Assert.Equal(2, result.Count());
}

finally
{
    connection.Close();
}

}

}
```

# Test avec InMemory

10/01/2020 • 6 minutes to read

Le fournisseur d'InMemory est utile lorsque vous souhaitez tester des composants à l'aide d'un composant qui se rapproche de la connexion à la base de données réelle, sans la surcharge des opérations de base de données réelles.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## La mémoire n'est pas une base de données relationnelle

EF Core les fournisseurs de base de données n'ont pas besoin d'être des bases de données relationnelles. La mémoire est conçue comme une base de données à usage général à des fins de test et n'est pas conçue pour imiter une base de données relationnelle.

Voici quelques exemples :

- InMemory vous permet d'enregistrer des données qui enfreignent les contraintes d'intégrité référentielle dans une base de données relationnelle.
- Si vous utilisez `DefaultValueSql (String)` pour une propriété dans votre modèle, il s'agit d'une API de base de données relationnelle qui n'aura aucun effet lors de l'exécution sur InMemory.
- La [concurrence via l'horodateur/la version de ligne](#) (`[Timestamp]` ou `IsRowVersion`) n'est pas prise en charge. Aucune `DbUpdateConcurrencyException` n'est levée si une mise à jour est effectuée à l'aide d'un ancien jeton d'accès concurrentiel.

## TIP

Dans de nombreux cas de test, ces différences n'ont pas d'importance. Toutefois, si vous souhaitez tester des éléments qui se comportent davantage comme une véritable base de données relationnelle, envisagez d'utiliser [le mode en mémoire SQLite](#).

## Exemple de scénario de test

Examinez le service suivant qui permet au code de l'application d'effectuer des opérations liées aux blogs. En interne, elle utilise un `DbContext` qui se connecte à une base de données SQL Server. Il serait utile de permute ce contexte pour se connecter à une base de données InMemory afin que nous puissions écrire des tests efficaces pour ce service sans avoir à modifier le code, ou faire beaucoup de travail pour créer un double de test du contexte.

```

using System.Collections.Generic;
using System.Linq;

namespace BusinessLogic
{
    public class BlogService
    {
        private BloggingContext _context;

        public BlogService(BloggingContext context)
        {
            _context = context;
        }

        public void Add(string url)
        {
            var blog = new Blog { Url = url };
            _context.Blogs.Add(blog);
            _context.SaveChanges();
        }

        public IEnumerable<Blog> Find(string term)
        {
            return _context.Blogs
                .Where(b => b.Url.Contains(term))
                .OrderBy(b => b.Url)
                .ToList();
        }
    }
}

```

## Préparez votre contexte

### Évitez de configurer deux fournisseurs de base de données

Dans vos tests, vous allez configurer en externe le contexte pour utiliser le fournisseur d'InMemory. Si vous configurez un fournisseur de base de données en remplaçant `OnConfiguring` dans votre contexte, vous devez ajouter du code conditionnel pour vous assurer que vous ne configurez le fournisseur de base de données que s'il n'a pas déjà été configuré.

```

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        optionsBuilder.UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=EFProviders.InMemory;Trusted_Connection=True;ConnectRetryCount=0");
    }
}

```

#### TIP

Si vous utilisez ASP.NET Core, vous n'avez pas besoin de ce code, car votre fournisseur de base de données est déjà configuré en dehors du contexte (dans Startup.cs).

## Ajouter un constructeur pour le test

La façon la plus simple d'activer les tests sur une autre base de données consiste à modifier votre contexte pour exposer un constructeur qui accepte un `DbContextOptions<TContext>`.

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    { }

    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }
```

**TIP**

`DbContextOptions<TContext>` indique au contexte tous ses paramètres, tels que la base de données à laquelle se connecter. Il s'agit du même objet que celui généré par l'exécution de la méthode `OnConfiguring` dans votre contexte.

## Écrire des tests

La clé à tester avec ce fournisseur est la possibilité d'indiquer au contexte d'utiliser le fournisseur d'`InMemory` et de contrôler l'étendue de la base de données en mémoire. En général, vous souhaitez une base de données propre pour chaque méthode de test.

Voici un exemple de classe de test qui utilise la base de données `InMemory`. Chaque méthode de test spécifie un nom de base de données unique, ce qui signifie que chaque méthode possède sa propre base de données `InMemory`.

**TIP**

Pour utiliser la méthode d'extension `.UseInMemoryDatabase()`, référez-vous au package NuGet [Microsoft.EntityFrameworkCore.InMemory](#).

```

using BusinessLogic;
using Microsoft.EntityFrameworkCore;
using System.Linq;
using Xunit;

namespace EFTesting.TestProject.InMemory
{
    public class BlogServiceTests
    {
        [Fact]
        public void Add_writes_to_database()
        {
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseInMemoryDatabase(databaseName: "Add_writes_to_database")
                .Options;

            // Run the test against one instance of the context
            using (var context = new BloggingContext(options))
            {
                var service = new BlogService(context);
                service.Add("https://example.com");
                context.SaveChanges();
            }

            // Use a separate instance of the context to verify correct data was saved to database
            using (var context = new BloggingContext(options))
            {
                Assert.Equal(1, context.Blogs.Count());
                Assert.Equal("https://example.com", context.Blogs.Single().Url);
            }
        }

        [Fact]
        public void Find_searches_url()
        {
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseInMemoryDatabase(databaseName: "Find_searches_url")
                .Options;

            // Insert seed data into the database using one instance of the context
            using (var context = new BloggingContext(options))
            {
                context.Blogs.Add(new Blog { Url = "https://example.com/cats" });
                context.Blogs.Add(new Blog { Url = "https://example.com/catfish" });
                context.Blogs.Add(new Blog { Url = "https://example.com/dogs" });
                context.SaveChanges();
            }

            // Use a clean instance of the context to run the test
            using (var context = new BloggingContext(options))
            {
                var service = new BlogService(context);
                var result = service.Find("cat");
                Assert.Equal(2, result.Count());
            }
        }
    }
}

```

# Configuration d'un DbContext

07/11/2019 • 11 minutes to read

Cet article présente les modèles de base pour la configuration d'un `DbContext` via une `DbContextOptions` pour se connecter à une base de données à l'aide d'un fournisseur de EF Core spécifique et de comportements facultatifs.

## Configuration de DbContext au moment du design

EF Core des outils au moment du design, tels que les [migrations](#), doivent être en mesure de découvrir et de créer une instance de travail d'un type `DbContext` afin de rassembler des détails sur les types d'entité de l'application et la façon dont ils sont mappés à un schéma de base de données. Ce processus peut être automatique tant que l'outil peut facilement créer le `DbContext` de manière à ce qu'il soit configuré de façon similaire à la façon dont il est configuré au moment de l'exécution.

Alors que tout modèle qui fournit les informations de configuration nécessaires au `DbContext` peut fonctionner au moment de l'exécution, les outils qui nécessitent l'utilisation d'une `DbContext` au moment du design peuvent uniquement fonctionner avec un nombre limité de modèles. Celles-ci sont traitées plus en détail dans la section [création de contexte au moment du design](#).

## Configuration de DbContextOptions

`DbContext` doit avoir une instance de `DbContextOptions` afin d'effectuer n'importe quel travail. L'instance `DbContextOptions` contient des informations de configuration telles que :

- Fournisseur de base de données à utiliser, généralement sélectionné en appelant une méthode telle que `UseSqlServer` ou `UseSqlite`. Ces méthodes d'extension nécessitent le package du fournisseur correspondant, tel que `Microsoft.EntityFrameworkCore.SqlServer` ou `Microsoft.EntityFrameworkCore.Sqlite`. Les méthodes sont définies dans l'espace de noms `Microsoft.EntityFrameworkCore`.
- Toute chaîne de connexion ou tout identificateur nécessaire de l'instance de base de données, généralement passé comme argument à la méthode de sélection du fournisseur mentionnée ci-dessus
- Tous les sélecteurs de comportements facultatifs au niveau du fournisseur, généralement chaînés à l'intérieur de l'appel à la méthode de sélection du fournisseur
- Tous les sélecteurs de comportements EF Core généraux, en général chaînés après ou avant la méthode du sélecteur du fournisseur

L'exemple suivant configure la `DbContextOptions` pour utiliser le fournisseur de SQL Server, une connexion contenue dans la variable `connectionString`, un délai d'attente de commande au niveau du fournisseur et un sélecteur de comportement EF Core qui effectue toutes les requêtes exécutées dans le `DbContext` sans suivi . par défaut :

```
optionsBuilder
    .UseSqlServer(connectionString, providerOptions=>providerOptions.CommandTimeout(60))
    .UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking);
```

## NOTE

Les méthodes de sélection de fournisseur et d'autres méthodes de sélecteur de comportements mentionnées ci-dessus sont des méthodes d'extension sur `DbContextOptions` ou des classes d'options spécifiques au fournisseur. Pour pouvoir accéder à ces méthodes d'extension, vous devrez peut-être avoir un espace de noms (généralement `Microsoft.EntityFrameworkCore`) dans la portée et inclure des dépendances de package supplémentaires dans le projet.

Le `DbContextOptions` peut être fourni à la `DbContext` en substituant la méthode `OnConfiguring` ou en externe à l'aide d'un argument de constructeur.

Si les deux sont utilisés, `OnConfiguring` est appliqué en dernier et peut remplacer les options fournies à l'argument du constructeur.

### Argument de constructeur

Votre constructeur peut simplement accepter un `DbContextOptions` comme suit :

```
public class BloggingContext : DbContext
{
    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }

    public DbSet<Blog> Blogs { get; set; }
}
```

## TIP

Le constructeur de base de `DbContext` accepte également la version non générique de `DbContextOptions`, mais l'utilisation de la version non générique n'est pas recommandée pour les applications avec plusieurs types de contexte.

Votre application peut maintenant passer le `DbContextOptions` lors de l'instanciation d'un contexte, comme suit :

```
var optionsBuilder = new DbContextOptionsBuilder<BloggingContext>();
optionsBuilder.UseSqlite("Data Source=blog.db");

using (var context = new BloggingContext(optionsBuilder.Options))
{
    // do stuff
}
```

### OnConfiguring

Vous pouvez également initialiser le `DbContextOptions` dans le contexte lui-même. Bien que vous puissiez utiliser cette technique pour la configuration de base, vous devrez généralement obtenir certains détails de configuration de l'extérieur, par exemple une chaîne de connexion de base de données. Pour ce faire, vous pouvez utiliser une API de configuration ou tout autre moyen.

Pour initialiser `DbContextOptions` dans le contexte, substituez la méthode `OnConfiguring` et appelez les méthodes sur le `DbContextOptionsBuilder` fourni :

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite("Data Source=blog.db");
    }
}
```

Une application peut simplement instancier ce type de contexte sans passer quoi que ce soit à son constructeur :

```
using (var context = new BloggingContext())
{
    // do stuff
}
```

#### TIP

Cette approche ne se prête pas au test, à moins que les tests ciblent la base de données complète.

## Utilisation de `DbContext` avec l'injection de dépendances

EF Core prend en charge l'utilisation de `DbContext` avec un conteneur d'injection de dépendance. Votre type `DbContext` peut être ajouté au conteneur de service à l'aide de la méthode `AddDbContext<TContext>`.

`AddDbContext<TContext>` fera à la fois votre type `DbContext`, `TContext` et le `DbContextOptions<TContext>` correspondant disponible pour l'injection à partir du conteneur de service.

Pour plus d'informations sur l'injection de dépendances, voir [plus de lectures](#) ci-dessous.

Ajout du `DbContext` à l'injection de dépendances :

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<BloggingContext>(options => options.UseSqlite("Data Source=blog.db"));
}
```

Cela nécessite l'ajout d'un [argument de constructeur](#) à votre type `DbContext` qui accepte `DbContextOptions<TContext>`.

Code de contexte :

```
public class BloggingContext : DbContext
{
    public BloggingContext(DbContextOptions<BloggingContext> options)
        :base(options)
    { }

    public DbSet<Blog> Blogs { get; set; }
}
```

Code d'application (dans ASP.NET Core) :

```
public class MyController
{
    private readonly BloggingContext _context;

    public MyController(BloggingContext context)
    {
        _context = context;
    }

    ...
}
```

Code d'application (à l'aide de ServiceProvider directement, moins courant) :

```
using (var context = serviceProvider.GetService<BloggingContext>())
{
    // do stuff
}

var options = serviceProvider.GetService<DbContextOptions<BloggingContext>>();
```

## Éviter les problèmes de thread DbContext

Entity Framework Core ne prend pas en charge l'exécution de plusieurs opérations parallèles sur la même instance `DbContext`. Cela comprend à la fois l'exécution parallèle des requêtes Async et toute utilisation simultanée explicite à partir de plusieurs threads. Par conséquent, toujours `await` appels asynchrones, ou utiliser des instances `DbContext` distinctes pour les opérations qui s'exécutent en parallèle.

Lorsque EF Core détecte une tentative d'utilisation d'une instance `DbContext` simultanément, vous voyez un `InvalidOperationException` avec un message semblable à celui-ci :

Une deuxième opération a démarré sur ce contexte avant la fin d'une opération précédente. Cela est généralement dû à différents threads utilisant la même instance de `DbContext`, mais il n'est pas garanti que les membres d'instance soient thread-safe.

Lorsque l'accès simultané n'est pas détecté, cela peut entraîner un comportement non défini, des blocages d'application et des données endommagées.

Il existe des erreurs courantes qui peuvent entraîner par inadvertance un accès simultané sur la même instance de `DbContext` :

### Oubli de l'attente de la fin d'une opération asynchrone avant le démarrage d'une autre opération sur le même `DbContext`

Les méthodes asynchrones permettent à EF Core de lancer des opérations qui accèdent à la base de données de façon non bloquante. Toutefois, si un appelant n'attend pas la fin de l'une de ces méthodes et poursuit l'exécution d'autres opérations sur le `DbContext`, l'état de la `DbContext` peut être, (et très probablement) endommagé.

Attendez toujours EF Core les méthodes asynchrones immédiatement.

### Partage implicite d'instances de `DbContext` sur plusieurs threads via l'injection de dépendances

La méthode d'extension `AddDbContext` enregistre les types `DbContext` avec une `durée de vie limitée` par défaut.

Cela ne pose pas de problèmes d'accès simultané dans les applications ASP.NET Core, car il n'existe qu'un seul thread qui exécute chaque demande client à un moment donné, et parce que chaque demande obtient une étendue d'injection de dépendances distincte (et par conséquent une instance `DbContext` distincte).

Toutefois, tout code qui exécute explicitement plusieurs threads en parallèle doit s'assurer que les instances `DbContext` ne sont jamais accessibles simultanément.

À l'aide de l'injection de dépendances, cela peut être obtenu en inscrivant le contexte comme étant étendu et en créant des étendues (à l'aide de `IServiceScopeFactory`) pour chaque thread, ou en inscrivant le `DbContext` comme transitoire (à l'aide de la surcharge de `AddDbContext` qui accepte un paramètre `ServiceLifetime`).

## Plus de lectures

- Lisez [injection de dépendances](#) pour en savoir plus sur l'utilisation de di.
- Pour plus d'informations, consultez le [test](#).

# Utilisation des types de référence Nullable

09/01/2020 • 9 minutes to read

C#8 a introduit une nouvelle fonctionnalité appelée [types de référence Nullable](#), ce qui permet d'annoter les types de référence, ce qui indique s'il est valide qu'ils contiennent ou non null. Si vous débutez avec cette fonctionnalité, il est recommandé de vous familiariser avec elle en lisant les C# documents.

Cette page présente la prise en charge de EF Core pour les types de référence Nullable et décrit les pratiques recommandées pour l'utiliser.

## Propriétés obligatoires et facultatives

La documentation principale sur les propriétés obligatoires et facultatives et leur interaction avec les types de référence Nullable sont les pages de [Propriétés obligatoires et facultatives](#). Il est recommandé de commencer par lire cette page en premier.

### NOTE

Soyez prudent lorsque vous activez les types de référence Nullable sur un projet existant : les propriétés de type de référence qui ont été précédemment configurées comme étant facultatives sont maintenant configurées comme obligatoires, sauf si elles sont explicitement annotées comme nullables. Lors de la gestion d'un schéma de base de données relationnelle, des migrations peuvent être générées, ce qui modifie la possibilité de valeur null de la colonne de base de données.

## DbContext et DbSet

Lorsque les types de référence Nullable sont activés, le C# compilateur émet des avertissements pour toute propriété non Nullable non initialisée, car celles-ci contiennent la valeur null. Par conséquent, la pratique courante consistant à définir un `DbSet` qui n'autorise pas les valeurs NULL sur un contexte génère désormais un avertissement. Toutefois, EF Core Initialise toujours toutes les propriétés `DbSet` sur les types dérivés de `DbContext`. il est donc garanti qu'ils ne sont jamais null, même si le compilateur ne connaît pas ce. Par conséquent, il est recommandé de conserver vos propriétés de `DbSet` qui n'acceptent pas les valeurs NULL, ce qui vous permet d'y accéder sans vérification de valeur null, et de mettre en silence les avertissements du compilateur en les définissant explicitement sur null avec l'aide de l'opérateur null-indulgent avec (!) :

```
public class NullableReferenceTypesContext : DbContext
{
    public DbSet<Customer> Customers { get; set; } = null!;
    public DbSet<Order> Orders { get; set; } = null!;

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        => optionsBuilder
            .UseSqlServer(@"Server=
(localdb)\mssqllocaldb;Database=EFNullableReferenceTypes;Trusted_Connection=True;ConnectRetryCount=0");
}
```

## Propriétés et initialisation non Nullable

Les avertissements du compilateur pour les types de référence non Nullable non initialisés sont également un problème pour les propriétés régulières sur vos types d'entité. Dans l'exemple ci-dessus, nous avons évité ces avertissements à l'aide de la [liaison de constructeur](#), une fonctionnalité qui fonctionne parfaitement avec les

propriétés non Nullable, garantissant qu'elles sont toujours initialisées. Toutefois, dans certains scénarios, la liaison de constructeur n'est pas une option : les propriétés de navigation, par exemple, ne peuvent pas être initialisées de cette manière.

Les propriétés de navigation requises présentent une difficulté supplémentaire : bien qu'une fonction dépendante existe toujours pour un principal donné, elle peut ou non être chargée par une requête particulière, en fonction des besoins à ce stade du programme ([voir les différents modèles de chargement des données](#)). En même temps, il n'est pas souhaitable de rendre ces propriétés Nullable, car cela force tous les accès à vérifier la valeur null, même si elles sont requises.

Une façon de traiter ces scénarios consiste à avoir une propriété qui n'accepte pas les valeurs NULL avec un [champ de stockage](#) Nullable :

```
public Address ShippingAddress
{
    set => _shippingAddress = value;
    get => _shippingAddress
        ?? throw new InvalidOperationException("Uninitialized property: " + nameof(ShippingAddress));
}
```

Étant donné que la propriété de navigation n'accepte pas les valeurs NULL, une navigation requise est configurée ; tant que la navigation est chargée correctement, le dépendant est accessible via la propriété. Toutefois, si vous accédez à la propriété sans avoir au préalable chargé l'entité associée, une exception InvalidOperationException est levée, car le contrat d'API a été utilisé incorrectement.

En guise d'alternative terse, il est possible d'initialiser simplement la propriété avec la valeur null à l'aide de l'opérateur null-indulgent avec ( !) :

```
public Product Product { get; set; } = null!;
```

Une valeur null réelle ne sera jamais observée, sauf en raison d'un bogue de programmation, par exemple l'accès à la propriété de navigation sans charger correctement l'entité associée au préalable.

#### NOTE

Les navigations de collection, qui contiennent des références à plusieurs entités associées, doivent toujours être non Nullable. Une collection vide signifie qu'il n'existe aucune entité associée, mais que la liste elle-même ne doit jamais être null.

## Navigation et inclusion des relations Nullable

Lorsque vous traitez des relations facultatives, il est possible de rencontrer des avertissements du compilateur dans lesquels une exception de référence null réelle serait impossible. Lors de la traduction et de l'exécution de vos requêtes LINQ, EF Core garantit que si une entité associée facultative n'existe pas, toute navigation vers celle-ci sera simplement ignorée, au lieu d'être levée. Toutefois, le compilateur ne tient pas compte de cette EF Core garantie et génère des avertissements comme si la requête LINQ avait été exécutée en mémoire, avec LINQ to Objects. Par conséquent, il est nécessaire d'utiliser l'opérateur null-indulgent avec ( !) pour informer le compilateur qu'une valeur null réelle n'est pas possible :

```
Console.WriteLine(order.OptionalInfo!.ExtraAdditionalInfo!.SomeExtraAdditionalInfo);
```

Un problème similaire se produit lors de l'inclusion de plusieurs niveaux de relations dans les navigations facultatives :

```
var order = context.Orders
    .Include(o => o.OptionalInfo!)
    .ThenInclude(op => op.ExtraAdditionalInfo)
    .Single();
```

Si vous effectuez cette tâche beaucoup et que les types d'entité en question sont principalement utilisés dans les requêtes EF Core, envisagez de rendre les propriétés de navigation non Nullable et de les configurer comme facultatives via l'API Fluent ou les annotations de données. Cela supprimera tous les avertissements du compilateur tout en gardant la relation facultative ; Toutefois, si vos entités sont parcourues en dehors de EF Core, vous pouvez observer des valeurs NULL, même si les propriétés sont annotées comme n'acceptant pas les valeurs NULL.

## Génération de modèles automatique

La C# fonctionnalité de type référence [Nullable 8](#) n'est actuellement pas prise en charge dans l'ingénierie à C# rebours : EF Core génère toujours du code qui suppose que la fonctionnalité est désactivée. Par exemple, les colonnes de texte Nullable seront échafaudées en tant que propriété de type `string`, et non pas `string?`, avec l'API Fluent ou les annotations de données utilisées pour configurer si une propriété est requise ou non. Vous pouvez modifier le code de génération de modèles automatique et C# les remplacer par des annotations de possibilité de valeur null. La prise en charge de la génération de modèles automatique pour les types de référence Nullable est suivie par le problème [#15520](#).

# Création et configuration d'un modèle

05/12/2019 • 2 minutes to read

Entity Framework utilise un ensemble de conventions pour créer un modèle basé sur la forme de vos classes d'entité. Vous pouvez spécifier une configuration supplémentaire pour compléter et/ou remplacer ce qui a été découvert par convention.

Cet article traite de la configuration qui peut être appliquée à un modèle ciblant n'importe quel magasin de données et qui peut être appliquée pendant le ciblage d'une base de données relationnelle. Les fournisseurs peuvent également activer la configuration qui est spécifique à un magasin de données particulier. Pour plus d'informations sur la configuration spécifique du fournisseur, consultez la section [Fournisseurs de base de données](#).

## TIP

Vous pouvez voir l' [exemple](#) de cet article sur GitHub.

## Utiliser l'API Fluent pour configurer un modèle

Vous pouvez substituer la méthode `OnModelCreating` dans votre contexte dérivé et utiliser [ModelBuilder API](#) pour configurer votre modèle. Il s'agit de la méthode de configuration la plus puissante, qui permet de spécifier une configuration sans modifier les classes d'entité. Dotée du niveau de priorité le plus élevé, la configuration de l'API Fluent remplace les conventions et les annotations de données.

```
using Microsoft.EntityFrameworkCore;

namespace EFModeling.FluentAPI.Required
{
    class MyContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }

        #region Required
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Blog>()
                .Property(b => b.Url)
                .IsRequired();
        }
        #endregion
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Url { get; set; }
    }
}
```

## Utiliser des annotations de données pour configurer un modèle

Vous pouvez également appliquer des attributs (également appelés annotations de données) à vos classes et propriétés. Les annotations de données remplacent les conventions, mais elles sont remplacées par la

configuration de l'API Fluent.

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel.DataAnnotations;

namespace EFModeling.DataAnnotations.Required
{
    class MyContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
    }

    #region Required
    public class Blog
    {
        public int BlogId { get; set; }
        [Required]
        public string Url { get; set; }
    }
    #endregion
}
```

# Types d'entité

09/01/2020 • 4 minutes to read

L'inclusion d'un DbSet d'un type dans votre contexte signifie qu'il est inclus dans le modèle de EF Core ; Nous faisons généralement référence à un type de ce type en tant qu'*entité*. EF Core pouvez lire et écrire des instances d'entité à partir de/vers la base de données, et si vous utilisez une base de données relationnelle, EF Core pouvez créer des tables pour vos entités via des migrations.

## Inclusion de types dans le modèle

Par Convention, les types qui sont exposés dans les propriétés DbSet de votre contexte sont inclus dans le modèle en tant qu'entités. Les types d'entités qui sont spécifiés dans la méthode `OnModelCreating` sont également inclus, comme tous les types trouvés en explorant de manière récursive les propriétés de navigation d'autres types d'entités découverts.

Dans l'exemple de code ci-dessous, tous les types sont inclus :

- `Blog` est inclus, car il est exposé dans une propriété DbSet sur le contexte.
- `Post` est inclus, car il est découvert via la propriété de navigation `Blog.Posts`.
- `AuditEntry`, car il est spécifié dans `OnModelCreating`.

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<AuditEntry>();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

public class AuditEntry
{
    public int AuditEntryId { get; set; }
    public string Username { get; set; }
    public string Action { get; set; }
}
```

## Exclusion de types du modèle

Si vous ne souhaitez pas qu'un type soit inclus dans le modèle, vous pouvez l'exclure :

- [Annotations de données](#)
- [API Fluent](#)

```
[NotMapped]
public class BlogMetadata
{
    public DateTime LoadedFromDatabase { get; set; }
}
```

## Nom du tableau

Par Convention, chaque type d'entité sera configuré pour être mappé à une table de base de données portant le même nom que la propriété DbSet qui expose l'entité. S'il n'existe aucun DbSet pour l'entité donnée, le nom de la classe est utilisé.

Vous pouvez configurer manuellement le nom de la table :

- [Annotations de données](#)
- [API Fluent](#)

```
[Table("blogs")]
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

## Schéma de table

Lorsque vous utilisez une base de données relationnelle, les tables sont par convention créées dans le schéma par défaut de votre base de données. Par exemple, Microsoft SQL Server utilise le schéma de `dbo` (SQLite ne prend pas en charge les schémas).

Vous pouvez configurer des tables à créer dans un schéma spécifique comme suit :

- [Annotations de données](#)
- [API Fluent](#)

```
[Table("blogs", Schema = "blogging")]
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Au lieu de spécifier le schéma pour chaque table, vous pouvez également définir le schéma par défaut au niveau du modèle à l'aide de l'API Fluent :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasDefaultSchema("blogging");
}
```

Notez que la définition du schéma par défaut affectera également d'autres objets de base de données, tels que les séquences.

# Propriétés d'entité

09/01/2020 • 9 minutes to read

Chaque type d'entité dans votre modèle a un ensemble de propriétés, qui EF Core lit et écrit à partir de la base de données. Si vous utilisez une base de données relationnelle, les propriétés d'entité sont mappées à des colonnes de table.

## Propriétés incluses et exclues

Par Convention, toutes les propriétés publiques avec un accesseur get et un accesseur Set seront incluses dans le modèle.

Les propriétés spécifiques peuvent être exclues comme suit :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    [NotMapped]
    public DateTime LoadedFromDatabase { get; set; }
}
```

## Noms des colonnes

Par Convention, lors de l'utilisation d'une base de données relationnelle, les propriétés d'entité sont mappées à des colonnes de table portant le même nom que la propriété.

Si vous préférez configurer vos colonnes avec des noms différents, vous pouvez le faire comme suit :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    [Column("blog_id")]
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

## Types de données de colonne

Lors de l'utilisation d'une base de données relationnelle, le fournisseur de base de données sélectionne un type de données en fonction du type .NET de la propriété. Il prend également en compte d'autres métadonnées, telles que la [longueur maximale](#) configurée, si la propriété fait partie d'une clé primaire, etc.

Par exemple, SQL Server mappe des propriétés de `DateTime` à des colonnes `datetime2(7)`, et `string` des propriétés à des colonnes `nvarchar(max)` (ou à `nvarchar(450)` pour les propriétés utilisées comme clé).

Vous pouvez également configurer vos colonnes pour spécifier un type de données exact pour une colonne. Par exemple, le code suivant configure `Url` en tant que chaîne non-Unicode avec une longueur maximale de `200` et `Rating` en tant que valeur décimale avec une précision de `5` et une échelle de `2` :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    public int BlogId { get; set; }
    [Column(TypeName = "varchar(200)")]
    public string Url { get; set; }
    [Column(TypeName = "decimal(5, 2)")]
    public decimal Rating { get; set; }
}
```

## Longueur maximale

La configuration d'une longueur maximale fournit aux fournisseurs de bases de données un indicateur sur le type de données de colonne approprié à choisir pour une propriété donnée. La longueur maximale s'applique uniquement aux types de données de tableau, tels que `string` et `byte[]`.

### NOTE

Entity Framework n'effectue aucune validation de longueur maximale avant de transmettre des données au fournisseur. Il revient au fournisseur ou au magasin de données de valider le cas échéant. Par exemple, lorsque vous ciblez SQL Server, le dépassement de la longueur maximale entraîne une exception, car le type de données de la colonne sous-jacente n'autorise pas le stockage des données excédentaires.

Dans l'exemple suivant, la configuration d'une longueur maximale de 500 entraîne la création d'une colonne de type `nvarchar(500)` sur SQL Server :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    public int BlogId { get; set; }
    [MaxLength(500)]
    public string Url { get; set; }
}
```

## Propriétés obligatoires et facultatives

Une propriété est considérée comme facultative si elle est valide pour contenir `null`. Si `null` n'est pas une valeur valide à assigner à une propriété, elle est considérée comme étant une propriété obligatoire. Lors du mappage à un schéma de base de données relationnelle, les propriétés requises sont créées en tant que colonnes n'acceptant pas les valeurs NULL, et les propriétés facultatives sont créées en tant que colonnes Nullable.

## Conventions

Par convention, une propriété dont le type .NET peut contenir une valeur null sera configurée comme étant facultative, alors que les propriétés dont le type .NET ne peut pas contenir de valeur null seront configurées selon les besoins. Par exemple, toutes les propriétés avec des types valeur .NET (`int`, `decimal`, `bool`, etc.) sont configurées en fonction des besoins, et toutes les propriétés avec des types valeur .NET Nullable (`int?`, `decimal?`, `bool?`, etc.) sont configurées comme facultatives.

C#8 a introduit une nouvelle fonctionnalité appelée [types de référence Nullable](#), qui permet d'annoter des types de référence, ce qui indique s'il est valide qu'ils contiennent ou non des valeurs NULL. Cette fonctionnalité est désactivée par défaut et, si elle est activée, elle modifie le comportement de EF Core de la façon suivante :

- Si les types de référence Nullable sont désactivés (valeur par défaut), toutes les propriétés avec des types de référence .NET sont configurées comme étant facultatives par convention (par exemple, `string`).
- Si les types de référence Nullable sont activés, les propriétés seront configurées en fonction de la C# possibilité de valeur null de leur type .net : `string?` sera configuré comme étant facultatif, tandis que `string` sera configuré comme requis.

L'exemple suivant montre un type d'entité avec des propriétés obligatoires et facultatives, avec la fonctionnalité de référence Nullable désactivée (valeur par défaut) et activée :

- [Sans types référence Nullable \(valeur par défaut\)](#)
- [Avec les types de référence Nullable](#)

```
public class CustomerWithoutNullableReferenceTypes
{
    public int Id { get; set; }

    [Required] // Data annotations needed to configure as required
    public string FirstName { get; set; }

    [Required]
    public string LastName { get; set; } // Data annotations needed to configure as required
    public string MiddleName { get; set; } // Optional by convention
}
```

L'utilisation de types de référence Nullable est recommandée, car elle transmet la C# possibilité de valeur null exprimée dans le code à EF Core modèle et à la base de données, et évite l'utilisation de l'API Fluent ou des annotations de données pour exprimer deux fois le même concept.

#### NOTE

Soyez prudent lorsque vous activez les types de référence Nullable sur un projet existant : les propriétés de type de référence qui ont été précédemment configurées comme étant facultatives sont maintenant configurées comme obligatoires, sauf si elles sont explicitement annotées comme nullables. Lors de la gestion d'un schéma de base de données relationnelle, des migrations peuvent être générées, ce qui modifie la possibilité de valeur null de la colonne de base de données.

Pour plus d'informations sur les types de référence Nullable et leur utilisation avec EF Core, [consultez la page de documentation dédiée pour cette fonctionnalité](#).

#### Configuration explicite

Une propriété qui serait facultative par convention peut être configurée comme suit :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    public int BlogId { get; set; }

    [Required]
    public string Url { get; set; }
}
```

# Touches

09/01/2020 • 7 minutes to read

Une clé sert d'identificateur unique pour chaque instance d'entité. La plupart des entités dans EF ont une clé unique, qui correspond au concept d'une *clé primaire* dans les bases de données relationnelles (pour les entités sans clés, consultez la rubrique [ne pas utiliser les entités](#)). Les entités peuvent avoir des clés supplémentaires au-delà de la clé primaire (pour plus d'informations, consultez [autres clés](#) ).

Par Convention, une propriété nommée `Id` ou `<type name>Id` sera configurée comme clé primaire d'une entité.

```
class Car
{
    public string Id { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}

class Truck
{
    public string TruckId { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

## NOTE

Les [types d'entités détenues](#) utilisent des règles différentes pour définir des clés.

Vous pouvez configurer une propriété unique comme clé primaire d'une entité comme suit :

- [Annotations de données](#)
- [API Fluent](#)

```
class Car
{
    [Key]
    public string LicensePlate { get; set; }

    public string Make { get; set; }
    public string Model { get; set; }
}
```

Vous pouvez également configurer plusieurs propriétés comme clé d'une entité. il s'agit d'une clé composite. Les clés composites ne peuvent être configurées qu'à l'aide de l'API Fluent. les conventions ne configureront jamais de clé composite et vous ne pourrez pas utiliser d'annotations de données pour en configurer une.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Car>()
        .HasKey(c => new { c.State, c.LicensePlate });
}
```

## Nom de la clé primaire

Par Convention, sur les bases de données relationnelles, les clés primaires sont créées avec le nom `PK_<type name>`.

Vous pouvez configurer le nom de la contrainte de clé primaire comme suit :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasKey(b => b.BlogId)
        .HasName("PrimaryKey_BlogId");
}
```

## Types et valeurs de clé

Si EF Core prend en charge l'utilisation de propriétés de n'importe quel type primitif comme clé primaire, y compris `string`, `Guid`, `byte[]` et autres, toutes les bases de données ne prennent pas en charge tous les types en tant que clés. Dans certains cas, les valeurs de clé peuvent être converties automatiquement en un type pris en charge ; sinon, la conversion doit être [spécifiée manuellement](#).

Les propriétés de clé doivent toujours avoir une valeur non définie par défaut lors de l'ajout d'une nouvelle entité au contexte, mais certains types sont [générés par la base de données](#). Dans ce cas, EF tente de générer une valeur temporaire lorsque l'entité est ajoutée à des fins de suivi. Une fois `SaveChanges` appelé, la valeur temporaire est remplacée par la valeur générée par la base de données.

### IMPORTANT

Si une propriété de clé a sa valeur générée par la base de données et qu'une valeur non définie par défaut est spécifiée lors de l'ajout d'une entité, EF suppose que l'entité existe déjà dans la base de données et essaiera de la mettre à jour au lieu d'en insérer une nouvelle. Pour éviter cela, désactivez la génération de valeur ou consultez [comment spécifier des valeurs explicites pour les propriétés générées](#).

## Clés secondaires

Une autre clé sert d'identificateur unique alternatif pour chaque instance d'entité en plus de la clé primaire. Il peut être utilisé comme cible d'une relation. Lors de l'utilisation d'une base de données relationnelle, cela correspond au concept d'index/de contrainte unique sur la ou les colonnes clés de remplacement et une ou plusieurs contraintes de clé étrangère qui réfèrent la ou les colonnes.

### TIP

Si vous souhaitez simplement garantir l'unicité d'une colonne, définissez un index unique plutôt qu'une autre clé (voir [index](#)).

Dans EF, les clés secondaires sont en lecture seule et fournissent une sémantique supplémentaire sur les index uniques, car elles peuvent être utilisées comme cible d'une clé étrangère.

Des clés alternatives sont généralement introduites pour vous si nécessaire et vous n'avez pas besoin de les configurer manuellement. Par Convention, une clé secondaire est introduite pour vous lorsque vous identifiez une propriété qui n'est pas la clé primaire comme la cible d'une relation.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogUrl)
            .HasPrincipalKey(b => b.Url);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public string BlogUrl { get; set; }
    public Blog Blog { get; set; }
}

```

Vous pouvez également configurer une seule propriété pour qu'elle soit une clé secondaire :

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Car>()
        .HasAlternateKey(c => c.LicensePlate);
}

```

Vous pouvez également configurer plusieurs propriétés en tant que clé secondaire (appelée clé de remplacement composite) :

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Car>()
        .HasAlternateKey(c => new { c.State, c.LicensePlate });
}

```

Enfin, par Convention, l'index et la contrainte introduits pour une clé secondaire sont nommés

`AK_<type name>_<property name>` (pour les clés de remplacement composites `<property name>` devient une liste de noms de propriété séparés par un trait de soulignement). Vous pouvez configurer le nom de l'index et de la contrainte unique de la clé secondaire :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Car>()
        .HasAlternateKey(c => c.LicensePlate)
        .HasName("AlternateKey_LicensePlate");
}
```

# Valeurs générées

09/01/2020 • 10 minutes to read

## Modèles de génération de valeur

Il existe trois modèles de génération de valeur qui peuvent être utilisés pour les propriétés :

- Aucune génération de valeur
- Valeur générée lors de l'ajout
- Valeur générée lors de l'ajout ou de la mise à jour

### Aucune génération de valeur

Aucune génération de valeur ne signifie que vous fournissez toujours une valeur valide à enregistrer dans la base de données. Cette valeur valide doit être assignée aux nouvelles entités avant d'être ajoutées au contexte.

### Valeur générée lors de l'ajout

La valeur générée lors de l'ajout signifie qu'une valeur est générée pour les nouvelles entités.

Selon le fournisseur de base de données utilisé, les valeurs peuvent être générées côté client par EF ou dans la base de données. Si la valeur est générée par la base de données, EF peut affecter une valeur temporaire lorsque vous ajoutez l'entité au contexte. Cette valeur temporaire sera ensuite remplacée par la valeur générée par la base de données pendant `SaveChanges()`.

Si vous ajoutez une entité au contexte qui a une valeur affectée à la propriété, EF tente d'insérer cette valeur au lieu d'en générer une nouvelle. Une propriété est considérée comme ayant une valeur affectée si elle n'est pas affectée à la valeur CLR par défaut (`null` pour `string`, `0` pour `int`, `Guid.Empty` pour `Guid`, etc.). Pour plus d'informations, consultez [valeurs explicites pour les propriétés générées](#).

#### WARNING

La façon dont la valeur est générée pour les entités ajoutées dépend du fournisseur de base de données utilisé. Les fournisseurs de base de données peuvent configurer automatiquement la génération de valeur pour certains types de propriété, mais d'autres peuvent vous obliger à configurer manuellement la manière dont la valeur est générée.

Par exemple, lors de l'utilisation de SQL Server, les valeurs sont générées automatiquement pour les propriétés de `GUID` (à l'aide de l'algorithme de GUID séquentiel SQL Server). Toutefois, si vous spécifiez qu'une propriété `DateTime` est générée lors de l'ajout, vous devez configurer un moyen de générer les valeurs. Pour ce faire, vous pouvez configurer une valeur par défaut de `GETDATE()`, voir [valeurs par défaut](#).

### Valeur générée lors de l'ajout ou de la mise à jour

La valeur générée à l'ajout ou à la mise à jour signifie qu'une nouvelle valeur est générée chaque fois que l'enregistrement est enregistré (Insert ou Update).

Comme `value generated on add`, si vous spécifiez une valeur pour la propriété sur une instance nouvellement ajoutée d'une entité, cette valeur est insérée au lieu d'une valeur générée. Il est également possible de définir une valeur explicite lors de la mise à jour. Pour plus d'informations, consultez [valeurs explicites pour les propriétés générées](#).

## WARNING

La façon dont la valeur est générée pour les entités ajoutées et mises à jour dépend du fournisseur de base de données utilisé. Les fournisseurs de base de données peuvent configurer automatiquement la génération de valeur pour certains types de propriété, tandis que d'autres vous obligent à configurer manuellement la façon dont la valeur est générée.

Par exemple, lors de l'utilisation de SQL Server, `byte[]` propriétés définies comme générées lors de l'ajout ou de la mise à jour et marquées comme jetons d'accès concurrentiel, seront configurées avec le type de données `rowversion` -afin que les valeurs soient générées dans la base de données. Toutefois, si vous spécifiez qu'une propriété `DateTime` est générée lors de l'ajout ou de la mise à jour, vous devez configurer un moyen de générer les valeurs. Une façon de procéder consiste à configurer une valeur par défaut de `GETDATE()` (voir les [valeurs par défaut](#)) pour générer des valeurs pour les nouvelles lignes. Vous pouvez ensuite utiliser un déclencheur de base de données pour générer des valeurs lors des mises à jour (par exemple, le déclencheur suivant).

```
CREATE TRIGGER [dbo].[Blogs_UPDATE] ON [dbo].[Blogs]
    AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF ((SELECT TRIGGER_NESTLEVEL()) > 1) RETURN;

    DECLARE @Id INT

    SELECT @Id = INSERTED.BlogId
    FROM INSERTED

    UPDATE dbo.Blogs
    SET LastUpdated = GETDATE()
    WHERE BlogId = @Id
END
```

## Valeur générée lors de l'ajout

Par convention, les clés primaires non composites de type short, int, long ou GUID sont configurées pour avoir des valeurs générées pour les entités insérées, si une valeur n'est pas fournie par l'application. Votre fournisseur de base de données prend généralement en charge la configuration requise ; par exemple, une clé primaire numérique dans SQL Server est automatiquement configurée pour être une colonne d'identité.

Vous pouvez configurer n'importe quelle propriété pour que sa valeur soit générée pour les entités insérées comme suit :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public DateTime Inserted { get; set; }
}
```

## WARNING

Cela permet simplement à EF de savoir que les valeurs sont générées pour les entités ajoutées. cela ne garantit pas qu'EF configure le mécanisme réel pour générer des valeurs. Pour plus d'informations, consultez la section [valeur générée sur l'ajout](#).

## Valeurs par défaut

Sur les bases de données relationnelles, une colonne peut être configurée avec une valeur par défaut ; Si une ligne est insérée sans valeur pour cette colonne, la valeur par défaut est utilisée.

Vous pouvez configurer une valeur par défaut sur une propriété :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property(b => b.Rating)
        .HasDefaultValue(3);
}
```

Vous pouvez également spécifier un fragment SQL utilisé pour calculer la valeur par défaut :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property(b => b.Created)
        .HasDefaultValueSql("getdate()");
}
```

La spécification d'une valeur par défaut configure implicitement la propriété en tant que valeur générée lors de l'ajout.

## Valeur générée lors de l'ajout ou de la mise à jour

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    public DateTime LastUpdated { get; set; }
}
```

### WARNING

Cela permet simplement à EF de savoir que les valeurs sont générées pour les entités ajoutées ou mises à jour, mais elle ne garantit pas que EF configure le mécanisme réel pour générer des valeurs. Pour plus d'informations, consultez la section [valeur générée dans la section Ajouter ou mettre à jour](#).

## Colonnes calculées

Sur certaines bases de données relationnelles, une colonne peut être configurée de manière à ce que sa valeur soit calculée dans la base de données, généralement avec une expression faisant référence à d'autres colonnes :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Person>()
        .Property(p => p.DisplayName)
        .HasComputedColumnSql("[LastName] + ', ' + [FirstName]");
}
```

#### NOTE

Dans certains cas, la valeur de la colonne est calculée chaque fois qu'elle est extraite (parfois appelée colonnes *virtuelles* ), et dans d'autres, elle est calculée à chaque mise à jour de la ligne et stockée (parfois appelée colonnes *stockées* ou *conservées* ). Cela varie d'un fournisseur de base de données à l'autre.

## Aucune génération de valeur

La désactivation de la génération de valeur sur une propriété est généralement nécessaire si une convention la configure pour la génération de valeur. Par exemple, si vous avez une clé primaire de type int, elle sera définie implicitement comme valeur générée lors de l'ajout. Vous pouvez désactiver ce code à l'aide des éléments suivants :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

# Jetons d'accès concurrentiel

10/01/2020 • 2 minutes to read

## NOTE

Cette page décrit comment configurer des jetons d'accès concurrentiel. Consultez [gestion des conflits d'accès concurrentiel](#) pour obtenir une explication détaillée du fonctionnement du contrôle d'accès concurrentiel sur EF Core et des exemples montrant comment gérer les conflits d'accès concurrentiel dans votre application.

Les propriétés configurées en tant que jetons d'accès concurrentiel permettent d'implémenter le contrôle d'accès concurrentiel optimiste.

## Configuration

- [Annotations de données](#)
- [API Fluent](#)

```
public class Person
{
    public int PersonId { get; set; }

    [ConcurrencyCheck]
    public string LastName { get; set; }

    public string FirstName { get; set; }
}
```

## Horodateur/rowversion

Un timestamp/rowversion est une propriété pour laquelle une nouvelle valeur est générée automatiquement par la base de données chaque fois qu'une ligne est insérée ou mise à jour. La propriété est également traitée comme un jeton d'accès concurrentiel, s'assurant que vous recevez une exception si une ligne que vous mettez à jour a changé depuis que vous l'avez interrogée. Les détails précis dépendent du fournisseur de base de données utilisé ; par SQL Server, une propriété `Byte[]` est généralement utilisée, qui sera configurée en tant que colonne `rowversion` dans la base de données.

Vous pouvez configurer une propriété pour qu'elle soit de type timestamp/rowversion comme suit :

- [Annotations de données](#)
- [API Fluent](#)

```
public class Blog
{
    public int BlogId { get; set; }

    public string Url { get; set; }

    [Timestamp]
    public byte[] Timestamp { get; set; }
}
```

# Propriétés cachées

10/01/2020 • 4 minutes to read

Les propriétés Shadow sont des propriétés qui ne sont pas définies dans votre classe d'entité .NET, mais qui sont définies pour ce type d'entité dans le modèle EF Core. La valeur et l'état de ces propriétés sont gérés uniquement dans le dispositif de suivi des modifications. Les propriétés Shadow sont utiles lorsque la base de données contient des données qui ne doivent pas être exposées sur les types d'entités mappés.

## Propriétés de l'ombre de la clé étrangère

Les propriétés Shadow sont le plus souvent utilisées pour les propriétés de clé étrangère, où la relation entre deux entités est représentée par une valeur de clé étrangère dans la base de données, mais la relation est gérée sur les types d'entité à l'aide des propriétés de navigation entre l'entité modes. Par Convention, EF introduit une propriété Shadow lorsqu'une relation est détectée, mais qu'aucune propriété de clé étrangère n'est trouvée dans la classe d'entité dépendante.

La propriété sera nommée `<navigation property name><principal key property name>` (la navigation sur l'entité dépendante, qui pointe vers l'entité principale, est utilisée pour le nommage). Si le nom de la propriété de clé principale comprend le nom de la propriété de navigation, le nom sera simplement `<principal key property name>`. S'il n'existe aucune propriété de navigation sur l'entité dépendante, le nom du type de principal est utilisé à la place.

Par exemple, la liste de code suivante entraînera l'introduction d'une propriété Shadow `BlogId` dans l'entité `Post` :

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    // Since there is no CLR property which holds the foreign
    // key for this relationship, a shadow property is created.
    public Blog Blog { get; set; }
}
```

## Configuration des propriétés Shadow

Vous pouvez utiliser l'API Fluent pour configurer les propriétés Shadow. Une fois que vous avez appelé la surcharge de chaîne de `Property`, vous pouvez chaîner n'importe quel appel de configuration que vous feriez

pour d'autres propriétés. Dans l'exemple suivant, étant donné que `Blog` n'a aucune propriété CLR nommée `LastUpdated`, une propriété Shadow est créée :

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property<DateTime>("LastUpdated");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Si le nom fourni à la méthode `Property` correspond au nom d'une propriété existante (une propriété Shadow ou une propriété Shadow définie sur la classe d'entité), le code configurera cette propriété existante au lieu d'introduire une nouvelle propriété Shadow.

## Accès aux propriétés Shadow

Les valeurs de propriété Shadow peuvent être obtenues et modifiées par le biais de l'API `ChangeTracker` :

```
context.Entry(myBlog).Property("LastUpdated").CurrentValue = DateTime.Now;
```

Les propriétés Shadow peuvent être référencées dans des requêtes LINQ via la méthode statique `EF.Property` :

```
var blogs = context.Blogs
    .OrderBy(b => EF.Property<DateTime>(b, "LastUpdated"));
```

# Relations

09/01/2020 • 25 minutes to read

Une relation définit la relation entre deux entités. Dans une base de données relationnelle, il est représenté par une contrainte de clé étrangère.

## NOTE

La plupart des exemples de cet article utilisent une relation un-à-plusieurs pour illustrer les concepts. Pour obtenir des exemples de relations un-à-un et plusieurs-à-plusieurs, consultez la section [autres modèles de relation](#) à la fin de l'article.

## Définition des termes

Il existe un certain nombre de termes utilisés pour décrire les relations

- **Entité dépendante** : Il s'agit de l'entité qui contient les propriétés de clé étrangère. Parfois appelé « enfant » de la relation.
- **Entité principale** : Il s'agit de l'entité qui contient les propriétés de clé primaire/secondeaire. Parfois appelé « parent » de la relation.
- **Clé étrangère** : Propriétés de l'entité dépendante utilisées pour stocker les valeurs de clé principale de l'entité associée.
- **Clé principale** : Propriétés qui identifient de façon unique l'entité principale. Il peut s'agir de la clé primaire ou d'une clé secondaire.
- **Propriété de navigation** : Propriété définie sur le principal et/ou sur l'entité dépendante qui référence l'entité associée.
  - **Propriété de navigation de la collection** : Propriété de navigation qui contient des références à de nombreuses entités associées.
  - **Propriété de navigation de référence** : Propriété de navigation qui contient une référence à une entité associée unique.
  - **Propriété de navigation inverse** : Lorsque vous discutez d'une propriété de navigation particulière, ce terme fait référence à la propriété de navigation à l'autre extrémité de la relation.
- **Relation à référence automatique** : Relation dans laquelle les types d'entités dépendant et principal sont identiques.

Le code suivant illustre une relation un-à-plusieurs entre `Blog` et `Post`

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

- `Post` est l'entité dépendante
- `Blog` est l'entité principale
- `Post.BlogId` est la clé étrangère
- `Blog.BlogId` est la clé principale (dans le cas présent, il s'agit d'une clé primaire plutôt que d'une clé secondaire)
- `Post.Blog` est une propriété de navigation de référence
- `Blog.Posts` est une propriété de navigation de collection
- `Post.Blog` est la propriété de navigation inverse de `Blog.Posts` (et vice versa)

## Conventions

Par défaut, une relation est créée lorsqu'une propriété de navigation est détectée sur un type. Une propriété est considérée comme une propriété de navigation si le type vers lequel elle pointe ne peut pas être mappé en tant que type scalaire par le fournisseur de base de données actuel.

### NOTE

Les relations découvertes par convention cibleront toujours la clé primaire de l'entité principale. Pour cibler une clé secondaire, une configuration supplémentaire doit être effectuée à l'aide de l'API Fluent.

### Relations entièrement définies

Le modèle le plus courant pour les relations est d'avoir des propriétés de navigation définies aux deux extrémités de la relation et une propriété de clé étrangère définie dans la classe d'entité dépendante.

- Si une paire de propriétés de navigation est trouvée entre deux types, ils sont configurés en tant que propriétés de navigation inverse de la même relation.
- Si l'entité dépendante contient une propriété dont le nom correspond à l'un de ces modèles, elle est configurée en tant que clé étrangère :

- <navigation property name><principal key property name>
- <navigation property name>Id
- <principal entity name><principal key property name>
- <principal entity name>Id

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

Dans cet exemple, les propriétés mises en surbrillance sont utilisées pour configurer la relation.

#### NOTE

Si la propriété est la clé primaire ou si elle est d'un type qui n'est pas compatible avec la clé principale, elle ne sera pas configurée en tant que clé étrangère.

#### NOTE

Avant le EF Core 3.0, la propriété nommée exactement comme la propriété de clé principale [était également mise en correspondance comme clé étrangère](#)

### Aucune propriété de clé étrangère

Bien qu'il soit recommandé d'avoir une propriété de clé étrangère définie dans la classe d'entité dépendante, elle n'est pas obligatoire. Si aucune propriété de clé étrangère n'est trouvée, une [propriété de clé étrangère Shadow](#) est introduite avec le nom `<navigation property name><principal key property name>` ou `<principal entity name><principal key property name>` si aucune navigation n'est présente sur le type dépendant.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

Dans cet exemple, la clé étrangère Shadow est `BlogId` car l'attente du nom de navigation est redondante.

## NOTE

Si une propriété du même nom existe déjà, le nom de la propriété Shadow sera suivi d'un nombre.

## Propriété de navigation unique

L'inclusion d'une seule propriété de navigation (aucune navigation inverse et aucune propriété de clé étrangère) suffit à avoir une relation définie par Convention. Vous pouvez également avoir une propriété de navigation unique et une propriété de clé étrangère.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}
```

## Limitations

Lorsqu'il existe plusieurs propriétés de navigation définies entre deux types (autrement dit, plusieurs paires de navigation qui pointent les unes vers les autres), les relations représentées par les propriétés de navigation sont ambiguës. Vous devrez les configurer manuellement pour résoudre l'ambiguïté.

## Suppression en cascade

Par Convention, la suppression en cascade sera définie sur *cascade* pour les relations requises et *ClientSetNull* pour les relations facultatives. *Cascade* signifie que les entités dépendantes sont également supprimées.

*ClientSetNull* signifie que les entités dépendantes qui ne sont pas chargées en mémoire restent inchangées et doivent être supprimées manuellement ou mises à jour pour pointer vers une entité principale valide. Pour les entités chargées en mémoire, EF Core tente de définir les propriétés de clé étrangère sur la valeur null.

Consultez la section [relations obligatoires et facultatives](#) pour connaître la différence entre les relations obligatoires et facultatives.

Pour plus d'informations sur les différents comportements de suppression et les valeurs par défaut utilisées par Convention, consultez [suppression en cascade](#).

## Configuration manuelle

- [API Fluent](#)
- [Annotations de données](#)

Pour configurer une relation dans l'API Fluent, commencez par identifier les propriétés de navigation qui composent la relation. `HasOne` ou `HasMany` identifie la propriété de navigation sur le type d'entité sur lequel vous commencez la configuration. Vous enchaînez ensuite un appel à `WithOne` ou `WithMany` pour identifier la navigation inverse. `HasOne` / `WithOne` sont utilisés pour les propriétés de navigation de référence et `HasMany` / `WithMany` sont utilisés pour les propriétés de navigation de collection.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

## Propriété de navigation unique

Si vous n'avez qu'une seule propriété de navigation, il y a des surcharges sans paramètre de `WithOne` et `WithMany`. Cela indique qu'il existe conceptuellement une référence ou une collection à l'autre extrémité de la relation, mais qu'il n'y a aucune propriété de navigation incluse dans la classe d'entité.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasMany(b => b.Posts)
            .WithOne();
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
}

```

## Clé étrangère

- [API Fluent \(clé simple\)](#)
- [API Fluent \(clé composite\)](#)
- [Annotations de données \(clé simple\)](#)

Vous pouvez utiliser l'API Fluent pour configurer la propriété à utiliser comme propriété de clé étrangère pour une relation donnée :

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey(p => p.BlogForeignKey);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogForeignKey { get; set; }
    public Blog Blog { get; set; }
}
```

### Masquer la clé étrangère

Vous pouvez utiliser la surcharge de chaîne de `HasForeignKey(...)` pour configurer une propriété Shadow comme clé étrangère (pour plus d'informations, consultez [Propriétés Shadow](#) ). Nous vous recommandons d'ajouter explicitement la propriété Shadow au modèle avant de l'utiliser comme clé étrangère (comme indiqué ci-dessous).

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        // Add the shadow property to the model
        modelBuilder.Entity<Post>()
            .Property<int>("BlogForeignKey");

        // Use the shadow property as a foreign key
        modelBuilder.Entity<Post>()
            .HasOne(p => p.Blog)
            .WithMany(b => b.Posts)
            .HasForeignKey("BlogForeignKey");
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

```

### Nom de la contrainte de clé étrangère

Par Convention, lorsque vous ciblez une base de données relationnelle, les contraintes de clé étrangère sont nommées `FK_`. Pour les clés étrangères composites devient une liste de noms de propriétés de clé étrangère séparés par un trait de soulignement.

Vous pouvez également configurer le nom de la contrainte comme suit :

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasOne(p => p.Blog)
        .WithMany(b => b.Posts)
        .HasForeignKey(p => p.BlogId)
        .HasConstraintName("ForeignKey_Post_Blog");
}

```

### Sans propriété de navigation

Vous n'avez pas nécessairement besoin de fournir une propriété de navigation. Vous pouvez simplement fournir une clé étrangère d'un côté de la relation.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Post>()
            .HasOne<Blog>()
            .WithMany()
            .HasForeignKey(p => p.BlogId);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
}

```

## Clé principale

Si vous souhaitez que la clé étrangère fasse référence à une propriété autre que la clé primaire, vous pouvez utiliser l'API Fluent pour configurer la propriété de clé principale de la relation. La propriété que vous configurez en tant que clé principale est automatiquement configurée en tant que [clé secondaire](#).

- [Clé simple](#)
- [Clé composite](#)

```

class MyContext : DbContext
{
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<RecordOfSale>()
            .HasOne(s => s.Car)
            .WithMany(c => c.SaleHistory)
            .HasForeignKey(s => s.CarLicensePlate)
            .HasPrincipalKey(c => c.LicensePlate);
    }
}

public class Car
{
    public int CarId { get; set; }
    public string LicensePlate { get; set; }
    public string Make { get; set; }
    public string Model { get; set; }

    public List<RecordOfSale> SaleHistory { get; set; }
}

public class RecordOfSale
{
    public int RecordOfSaleId { get; set; }
    public DateTime DateSold { get; set; }
    public decimal Price { get; set; }

    public string CarLicensePlate { get; set; }
    public Car Car { get; set; }
}

```

## Relations obligatoires et facultatives

Vous pouvez utiliser l'API Fluent pour déterminer si la relation est obligatoire ou facultative. Au final, cela contrôle si la propriété de clé étrangère est obligatoire ou facultative. Cela est particulièrement utile lorsque vous utilisez une clé étrangère d'État Shadow. Si vous avez une propriété de clé étrangère dans votre classe d'entité, le caractère requis de la relation est déterminé selon que la propriété de clé étrangère est obligatoire ou facultative (pour plus d'informations, consultez [propriétés requises et facultatives](#) ).

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasOne(p => p.Blog)
        .WithMany(b => b.Posts)
        .IsRequired();
}

```

### NOTE

L'appel de `IsRequired(false)` rend également la propriété de clé étrangère facultative, sauf si elle est configurée dans le cas contraire.

## Suppression en cascade

Vous pouvez utiliser l'API Fluent pour configurer explicitement le comportement de suppression en cascade pour une relation donnée.

Pour une présentation détaillée de chaque option, consultez [suppression en cascade](#) .

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasOne(p => p.Blog)
        .WithMany(b => b.Posts)
        .OnDelete(DeleteBehavior.Cascade);
}
```

## Autres modèles de relation

### Un-à-un

Les relations un-à-un ont une propriété de navigation de référence des deux côtés. Ils suivent les mêmes conventions que les relations un-à-plusieurs, mais un index unique est introduit sur la propriété de clé étrangère pour s'assurer qu'un seul dépendant est lié à chaque principal.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }
}

public class BlogImage
{
    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

#### NOTE

EF choisit l'une des entités à dépendre en fonction de sa capacité à détecter une propriété de clé étrangère. Si l'entité incorrecte est choisie comme dépendant, vous pouvez utiliser l'API Fluent pour corriger ce problème.

Quand vous configurez la relation avec l'API Fluent, vous utilisez les méthodes `HasOne` et `WithOne`.

Quand vous configurez la clé étrangère, vous devez spécifier le type d'entité dépendant. Notez le paramètre générique fourni à `HasForeignKey` dans la liste ci-dessous. Dans une relation un-à-plusieurs, il est clair que l'entité avec la navigation de référence est la dépendance et celle avec la collection est le principal. Mais ce n'est pas le cas dans une relation un-à-un, c'est pourquoi il est nécessaire de la définir explicitement.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<BlogImage> BlogImages { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .HasOne(b => b.BlogImage)
            .WithOne(i => i.Blog)
            .HasForeignKey<BlogImage>(b => b.BlogForeignKey);
    }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public BlogImage BlogImage { get; set; }
}

public class BlogImage
{
    public int BlogImageId { get; set; }
    public byte[] Image { get; set; }
    public string Caption { get; set; }

    public int BlogForeignKey { get; set; }
    public Blog Blog { get; set; }
}

```

## Plusieurs-à-plusieurs

Les relations plusieurs-à-plusieurs sans classe d'entité pour représenter la table de jointure ne sont pas encore prises en charge. Toutefois, vous pouvez représenter une relation plusieurs-à-plusieurs en incluant une classe d'entité pour la table de jointure et en mappant deux relations un-à-plusieurs distinctes.

```
class MyContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public DbSet<Tag> Tags { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<PostTag>()
            .HasKey(t => new { t.PostId, t.TagId });

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Post)
            .WithMany(p => p.PostTags)
            .HasForeignKey(pt => pt.PostId);

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Tag)
            .WithMany(t => t.PostTags)
            .HasForeignKey(pt => pt.TagId);
    }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public List<PostTag> PostTags { get; set; }
}

public class Tag
{
    public string TagId { get; set; }

    public List<PostTag> PostTags { get; set; }
}

public class PostTag
{
    public int PostId { get; set; }
    public Post Post { get; set; }

    public string TagId { get; set; }
    public Tag Tag { get; set; }
}
```

# Index

09/01/2020 • 5 minutes to read

Les index sont un concept commun entre de nombreux magasins de données. Bien que leur implémentation dans le magasin de données puisse varier, elles sont utilisées pour rendre les recherches basées sur une colonne (ou un ensemble de colonnes) plus efficaces.

Les index ne peuvent pas être créés à l'aide d'annotations de données. Vous pouvez utiliser l'API Fluent pour spécifier un index sur une seule colonne comme suit :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasIndex(b => b.Url);
}
```

Vous pouvez également spécifier un index sur plusieurs colonnes :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Person>()
        .HasIndex(p => new { p.FirstName, p.LastName });
}
```

## NOTE

Par Convention, un index est créé dans chaque propriété (ou ensemble de propriétés) utilisée comme clé étrangère.

EF Core ne prend en charge qu'un seul index par jeu de propriétés distinct. Si vous utilisez l'API Fluent pour configurer un index sur un ensemble de propriétés pour lequel un index est déjà défini, soit par Convention, soit par configuration précédente, vous allez modifier la définition de cet index. Cela est utile si vous souhaitez configurer davantage un index qui a été créé par Convention.

## Unicité de l'index

Par défaut, les index ne sont pas uniques : plusieurs lignes peuvent avoir la même valeur (s) pour le jeu de colonnes de l'index. Vous pouvez rendre un index unique comme suit :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasIndex(b => b.Url)
        .IsUnique();
}
```

Toute tentative d'insertion de plusieurs entités avec les mêmes valeurs pour le jeu de colonnes de l'index entraîne la levée d'une exception.

## Nom d'index

Par Convention, les index créés dans une base de données relationnelle sont nommés

`IX_<type name>_<property name>`. Pour les index composites, `<property name>` devient une liste de noms de propriétés séparés par un trait de soulignement.

Vous pouvez utiliser l'API Fluent pour définir le nom de l'index créé dans la base de données :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasIndex(b => b.Url)
        .Name("Index_Url");
}
```

## Filtre d'index

Certaines bases de données relationnelles vous permettent de spécifier un index filtré ou partiel. Cela vous permet d'indexer uniquement un sous-ensemble des valeurs d'une colonne, en réduisant la taille de l'index et en améliorant l'utilisation des performances et de l'espace disque. Pour plus d'informations sur les index filtrés SQL Server, [consultez la documentation](#).

Vous pouvez utiliser l'API Fluent pour spécifier un filtre sur un index, fourni sous la forme d'une expression SQL :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasIndex(b => b.Url)
        .Filter("[Url] IS NOT NULL");
}
```

Lorsque vous utilisez le SQL Server le fournisseur EF ajoute un filtre `'IS NOT NULL'` pour toutes les colonnes Nullable qui font partie d'un index unique. Pour remplacer cette Convention, vous pouvez fournir une valeur `null`

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasIndex(b => b.Url)
        .IsUnique()
        .Filter(null);
}
```

## Colonnes incluses

Certaines bases de données relationnelles vous permettent de configurer un ensemble de colonnes qui sont incluses dans l'index, mais qui ne font pas partie de sa « clé ». Cela peut améliorer considérablement les performances des requêtes lorsque toutes les colonnes de la requête sont incluses dans l'index en tant que colonnes clés ou non-clés, car la table elle-même n'a pas besoin d'être accessible. Pour plus d'informations sur SQL Server colonnes incluses, [consultez la documentation](#).

Dans l'exemple suivant, la colonne `Url` fait partie de la clé d'index, de sorte que tout filtrage de requête sur cette colonne peut utiliser l'index. En outre, les requêtes qui accèdent uniquement aux colonnes `Title` et `PublishedOn` n'ont pas besoin d'accéder à la table et s'exécutent plus efficacement :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasIndex(p => p.Url)
        .IncludeProperties(p => new
    {
        p.Title,
        p.PublishedOn
    });
}
```

# Héritage

09/01/2020 • 5 minutes to read

EF peut mapper une hiérarchie de types .NET à une base de données. Cela vous permet d'écrire vos entités .NET dans le code comme d'habitude, à l'aide de types de base et dérivés, et de faire en sorte que EF crée en toute transparence le schéma de base de données approprié, les requêtes d'émission, etc. Les détails réels de la façon dont une hiérarchie de types est mappée dépendent du fournisseur ; Cette page décrit la prise en charge de l'héritage dans le contexte d'une base de données relationnelle.

Pour le moment, EF Core ne prend en charge que le modèle TPH (table par hiérarchie). TPH utilise une table unique pour stocker les données de tous les types dans la hiérarchie, et une colonne de discriminateur est utilisée pour identifier le type représenté par chaque ligne.

## NOTE

La table par type (TPT) et la table par type (TPC), qui sont prises en charge par EF6, ne sont pas encore prises en charge par EF Core. TPT est une fonctionnalité majeure prévue pour EF Core 5,0.

## Mappage de la hiérarchie des types d'entités

Par Convention, EF ne configure que l'héritage si au moins deux types hérités sont explicitement inclus dans le modèle. EF ne recherche pas automatiquement les types de base ou dérivés qui ne sont pas inclus dans le modèle.

Vous pouvez inclure des types dans le modèle en exposant un DbSet pour chaque type dans la hiérarchie d'héritage :

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<RssBlog> RssBlogs { get; set; }
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}

public class RssBlog : Blog
{
    public string RssUrl { get; set; }
}
```

Ce modèle est mappé au schéma de base de données suivant (Notez la colonne de *discriminateur* créée implicitement, qui identifie le type de *blog* stocké dans chaque ligne) :

| Results |        |               |                               |
|---------|--------|---------------|-------------------------------|
|         | BlogId | Discriminator | Url                           |
| 1       | 1      | Blog          | http://blogs.msdn.com/dotnet  |
| 2       | 2      | RssBlog       | http://blogs.msdn.com/ado.net |

#### NOTE

Les colonnes de base de données deviennent automatiquement Nullable si nécessaire lors de l'utilisation du mappage TPH. Par exemple, la colonne `RssUrl` accepte les valeurs NULL, car les instances de `blog` ordinaires n'ont pas cette propriété.

Si vous ne souhaitez pas exposer un `DbSet` pour une ou plusieurs entités dans la hiérarchie, vous pouvez également utiliser l'API Fluent pour vous assurer qu'elles sont incluses dans le modèle.

#### TIP

Si vous ne vous fiez pas aux conventions, vous pouvez spécifier explicitement le type de base à l'aide de `HasBaseType`. Vous pouvez également utiliser `.HasBaseType((Type)null)` pour supprimer un type d'entité de la hiérarchie.

## Configuration de discriminateur

Vous pouvez configurer le nom et le type de la colonne de discriminateur et les valeurs utilisées pour identifier chaque type dans la hiérarchie :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasDiscriminator<string>("blog_type")
        .HasValue<Blog>("blog_base")
        .HasValue<RssBlog>("blog_rss");
}
```

Dans les exemples ci-dessus, EF a ajouté le discriminateur implicitement en tant que [propriété Shadow](#) sur l'entité de base de la hiérarchie. Cette propriété peut être configurée comme n'importe quelle autre :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .Property("Discriminator")
        .HasMaxLength(200);
}
```

Enfin, le discriminateur peut également être mappé à une propriété .NET normale dans votre entité :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>()
        .HasDiscriminator(b => b.BlogType);

    modelBuilder.Entity<Blog>()
        .Property(e => e.BlogType)
        .HasMaxLength(200)
        .HasColumnName("blog_type");
}
```

## Colonnes partagées

Par défaut, lorsque deux types d'entités frères dans la hiérarchie ont une propriété portant le même nom, ils sont mappés à deux colonnes distinctes. Toutefois, si leur type est identique, ils peuvent être mappés à la même colonne de base de données :

```
public class MyContext : DbContext
{
    public DbSet<BlogBase> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .HasColumnName("Url");

        modelBuilder.Entity<RssBlog>()
            .Property(b => b.Url)
            .HasColumnName("Url");
    }
}

public abstract class BlogBase
{
    public int BlogId { get; set; }
}

public class Blog : BlogBase
{
    public string Url { get; set; }
}

public class RssBlog : BlogBase
{
    public string Url { get; set; }
}
```

# Séquences

09/01/2020 • 2 minutes to read

## NOTE

Les séquences sont une fonctionnalité généralement prise en charge uniquement par les bases de données relationnelles. Si vous utilisez une base de données non relationnelle telle que Cosmos, consultez la documentation de votre base de données sur la génération de valeurs uniques.

Une séquence génère des valeurs numériques séquentielles uniques dans la base de données. Les séquences ne sont pas associées à une table spécifique, et plusieurs tables peuvent être configurées pour dessiner des valeurs à partir de la même séquence.

## Utilisation de base

Vous pouvez configurer une séquence dans le modèle, puis l'utiliser pour générer des valeurs pour les propriétés :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasSequence<int>("OrderNumbers");

    modelBuilder.Entity<Order>()
        .Property(o => o.OrderNo)
        .HasDefaultValueSql("NEXT VALUE FOR shared.OrderNumbers");
}
```

Notez que le SQL spécifique utilisé pour générer une valeur à partir d'une séquence est spécifique à la base de données ; l'exemple ci-dessus fonctionne sur SQL Server mais échoue sur les autres bases de données. Pour plus d'informations, consultez la documentation de votre base de données spécifique.

## Configuration des paramètres de séquence

Vous pouvez également configurer d'autres aspects de la séquence, tels que son schéma, sa valeur de départ, son incrément, etc. :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasSequence<int>("OrderNumbers", schema: "shared")
        .StartsAt(1000)
        .IncrementsBy(5);
}
```

# Champs de stockage

25/10/2019 • 4 minutes to read

## NOTE

Cette fonctionnalité est une nouveauté de EF Core 1.1.

Les champs de stockage permettent à EF de lire et/ou d'écrire dans un champ plutôt que sur une propriété. Cela peut être utile lorsque l'encapsulation dans la classe est utilisée pour limiter l'utilisation de et/ou améliorer la sémantique de l'accès aux données par le code d'application, mais la valeur doit être lue et/ou écrite dans la base de données sans utiliser ces restrictions/ améliorations.

## Conventions

Par Convention, les champs suivants sont découverts en tant que champs de stockage pour une propriété donnée (liste dans l'ordre de priorité). Les champs sont uniquement détectés pour les propriétés incluses dans le modèle. Pour plus d'informations sur les propriétés incluses dans le modèle, consultez [inclusion de & exclusion de propriétés](#).

- `_<camel-cased property name>`
- `_<property name>`
- `m_<camel-cased property name>`
- `m_<property name>`

```
public class Blog
{
    private string _url;

    public int BlogId { get; set; }

    public string Url
    {
        get { return _url; }
        set { _url = value; }
    }
}
```

Lorsqu'un champ de stockage est configuré, EF écrit directement dans ce champ lors de la matérialisation des instances d'entité de la base de données (au lieu d'utiliser l'accesseur Set de propriété). Si EF doit lire ou écrire la valeur à d'autres moments, il utilisera la propriété si possible. Par exemple, si EF doit mettre à jour la valeur d'une propriété, il utilisera la méthode setter de propriété si celle-ci est définie. Si la propriété est en lecture seule, elle écrit dans le champ.

## Annotations de données

Les champs de stockage ne peuvent pas être configurés avec des annotations de données.

## API Fluent

Vous pouvez utiliser l'API Fluent pour configurer un champ de stockage pour une propriété.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .HasField("_validatedUrl");
    }
}

public class Blog
{
    private string _validatedUrl;

    public int BlogId { get; set; }

    public string Url
    {
        get { return _validatedUrl; }
    }

    public void SetUrl(string url)
    {
        using (var client = new HttpClient())
        {
            var response = client.GetAsync(url).Result;
            response.EnsureSuccessStatusCode();
        }

        _validatedUrl = url;
    }
}

```

## Contrôle de l'utilisation du champ

Vous pouvez configurer le moment où EF utilise le champ ou la propriété. Consultez l' [énumération PropertyAccessMode](#) pour connaître les options prises en charge.

```

modelBuilder.Entity<Blog>()
    .Property(b => b.Url)
    .HasField("_validatedUrl")
    .UsePropertyAccessMode(PropertyAccessMode.Field);

```

## Champs sans propriété

Vous pouvez également créer une propriété conceptuelle dans votre modèle qui n'a pas de propriété CLR correspondante dans la classe d'entité, mais utilise à la place un champ pour stocker les données dans l'entité. Cela diffère des [Propriétés Shadow](#), où les données sont stockées dans le dispositif de suivi des modifications. Cela est généralement utilisé si la classe d'entité utilise des méthodes pour récupérer/définir des valeurs.

Vous pouvez indiquer à EF le nom du champ dans l'API `Property(...)`. S'il n'existe aucune propriété avec le nom donné, EF recherche un champ.

```

class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property("validatedUrl");
    }
}

public class Blog
{
    private string _validatedUrl;

    public int BlogId { get; set; }

    public string GetUrl()
    {
        return _validatedUrl;
    }

    public void SetUrl(string url)
    {
        using (var client = new HttpClient())
        {
            var response = client.GetAsync(url).Result;
            response.EnsureSuccessStatusCode();
        }

        _validatedUrl = url;
    }
}

```

Lorsqu'il n'y a aucune propriété dans la classe d'entité, vous pouvez utiliser la méthode `EF.Property(...)` dans une requête LINQ pour faire référence à la propriété qui fait partie conceptuellement du modèle.

```
var blogs = db.blogs.OrderBy(b => EF.Property<string>(b, "validatedUrl"));
```

# Conversions de valeurs

07/11/2019 • 8 minutes to read

## NOTE

Cette fonctionnalité est une nouveauté d'EF Core 2.1.

Les convertisseurs de valeurs autorisent la conversion des valeurs de propriété lors de la lecture ou de l'écriture dans la base de données. Cette conversion peut être d'une valeur à une autre du même type (par exemple, le chiffrement de chaînes) ou d'une valeur d'un type à une valeur d'un autre type (par exemple, la conversion de valeurs enum vers et à partir de chaînes dans la base de données).

## Notions de base

Les convertisseurs de valeurs sont spécifiés en termes d'`ModelClrType` et de `ProviderClrType`. Le type de modèle est le type .NET de la propriété dans le type d'entité. Le type de fournisseur est le type .NET compris par le fournisseur de base de données. Par exemple, pour enregistrer des enums en tant que chaînes dans la base de données, le type de modèle est le type de l'énumération, et le type de fournisseur est `String`. Ces deux types peuvent être identiques.

Les conversions sont définies à l'aide de deux `Func` arborescences d'expressions : une de `ModelClrType` à `ProviderClrType` et l'autre de `ProviderClrType` à `ModelClrType`. Les arborescences d'expressions sont utilisées afin qu'elles puissent être compilées dans le code d'accès à la base de données pour des conversions efficaces. Pour les conversions complexes, l'arborescence de l'expression peut être un simple appel à une méthode qui effectue la conversion.

## Configuration d'un convertisseur de valeurs

Les conversions de valeurs sont définies sur les propriétés dans le `OnModelCreating` de votre `DbContext`. Prenons l'exemple d'une énumération et d'un type d'entité définis comme suit :

```
public class Rider
{
    public int Id { get; set; }
    public EquineBeast Mount { get; set; }
}

public enum EquineBeast
{
    Donkey,
    Mule,
    Horse,
    Unicorn
}
```

Les conversions peuvent ensuite être définies dans `OnModelCreating` pour stocker les valeurs d'énumération sous forme de chaînes (par exemple, « Donkey », « mule »,...) dans la base de données :

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder
        .Entity<Rider>()
        .Property(e => e.Mount)
        .HasConversion(
            v => v.ToString(),
            v => (EquineBeast)Enum.Parse(typeof(EquineBeast), v));
}

```

#### NOTE

Une valeur de `null` ne sera jamais passée à un convertisseur de valeurs. Cela rend l'implémentation des conversions plus facile et permet de les partager entre des propriétés Nullable et non Nullable.

## La classe ValueConverter

L'appel de `HasConversion` comme indiqué ci-dessus crée une instance `ValueConverter` et la définit sur la propriété. Le `valueConverter` peut à la place être créé explicitement. Exemple :

```

var converter = new ValueConverter<EquineBeast, string>(
    v => v.ToString(),
    v => (EquineBeast)Enum.Parse(typeof(EquineBeast), v));

modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion(converter);

```

Cela peut être utile lorsque plusieurs propriétés utilisent la même conversion.

#### NOTE

Il n'existe actuellement aucun moyen de spécifier à un endroit que chaque propriété d'un type donné doit utiliser le même convertisseur de valeur. Cette fonctionnalité sera prise en compte pour une version ultérieure.

## Convertisseurs intégrés

EF Core est fourni avec un ensemble de classes de `ValueConverter` prédéfinies, qui se trouvent dans l'espace de noms `Microsoft.EntityFrameworkCore.Storage.ValueConversion`. Ces équivalents sont :

- `BoolToZeroOneConverter` -bool à zéro et un
- `BoolToStringConverter` -bool à des chaînes telles que « Y » et « N »
- `BoolToTwoValuesConverter` -bool à deux valeurs quelconques
- Tableau `BytesToStringConverter` -octet en chaîne encodée en base64
- conversions de `CastingConverter` qui requièrent uniquement un cast de type
- `CharToStringConverter` -char en chaîne de caractères uniques
- valeur 64 bits encodée en binaire `DateTimeOffsetToBinaryConverter` -`DateTimeOffset`
- `DateTimeOffsetToBytesConverter` -`DateTimeOffset` au tableau d'octets
- `DateTimeOffsetToStringConverter` -`DateTimeOffset` en chaîne
- `DateTimeToBinaryConverter` -`DateTime` à la valeur 64 bits, y compris `DateTimeKind`
- `DateTimeToStringConverter` -`DateTime` en chaîne

- `DateTimeToTicksConverter` -DateTime aux battements
- `EnumToNumberConverter` -enum au nombre sous-jacent
- `EnumToStringConverter` -enum en chaîne
- `GuidToBytesConverter` -GUID au tableau d'octets
- `GuidToStringConverter` -GUID en chaîne
- `NumberToBytesConverter` -toute valeur numérique à un tableau d'octets
- `NumberToStringConverter` -toute valeur numérique à chaîne
- `StringToBytesConverter` -chaîne en octets UTF8
- `TimeSpanToStringConverter` -TimeSpan à chaîne
- `TimeSpanToTicksConverter` -intervalle de cycles

Notez que `EnumToStringConverter` est inclus dans cette liste. Cela signifie qu'il n'est pas nécessaire de spécifier explicitement la conversion, comme indiqué ci-dessus. Au lieu de cela, utilisez simplement le convertisseur intégré :

```
var converter = new EnumToStringConverter<EquineBeast>();

modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion(converter);
```

Notez que tous les convertisseurs intégrés sont sans État et qu'une seule instance peut être partagée en toute sécurité par plusieurs propriétés.

## Conversions prédéfinies

Pour les conversions courantes pour lesquelles un convertisseur intégré existe, il n'est pas nécessaire de spécifier explicitement le convertisseur. Au lieu de cela, il vous suffit de configurer le type de fournisseur à utiliser et EF utilisera automatiquement le convertisseur intégré approprié. Les conversions de type enum en chaînes sont utilisées comme exemple ci-dessus, mais EF effectue cette opération automatiquement si le type de fournisseur est configuré :

```
modelBuilder
    .Entity<Rider>()
    .Property(e => e.Mount)
    .HasConversion<string>();
```

La même chose peut être obtenue en spécifiant explicitement le type de colonne. Par exemple, si le type d'entité est défini comme suit :

```
public class Rider
{
    public int Id { get; set; }

    [Column(TypeName = "nvarchar(24)")]
    public EquineBeast Mount { get; set; }
}
```

Les valeurs enum sont ensuite enregistrées sous forme de chaînes dans la base de données sans aucune configuration supplémentaire dans `OnModelCreating`.

## Limitations

Il existe quelques limitations actuelles connues du système de conversion de valeurs :

- Comme indiqué ci-dessus, `null` ne peut pas être converti.
- Il n'existe actuellement aucun moyen de répartir la conversion d'une propriété en plusieurs colonnes, ou vice versa.
- L'utilisation de conversions de valeurs peut avoir un impact sur la capacité de EF Core à convertir des expressions en SQL. Un avertissement sera consigné dans ce cas. La suppression de ces limitations est prise en compte pour une version ultérieure.

# Amorçage des données

07/11/2019 • 7 minutes to read

L'amorçage de données est le processus de remplissage d'une base de données avec un jeu de données initial.

Pour ce faire, vous pouvez procéder de plusieurs façons dans EF Core :

- Données de valeur de départ du modèle
- Personnalisation manuelle de la migration
- Logique d'initialisation personnalisée

## Données de valeur de départ du modèle

### NOTE

Cette fonctionnalité est une nouveauté d'EF Core 2.1.

Contrairement à EF6, dans EF Core, les données d'amorçage peuvent être associées à un type d'entité dans le cadre de la configuration du modèle. Ensuite EF Core [migrations](#) peut calculer automatiquement les opérations d'insertion, de mise à jour ou de suppression qui doivent être appliquées lors de la mise à niveau de la base de données vers une nouvelle version du modèle.

### NOTE

Les migrations considèrent uniquement les modifications de modèle lors de la détermination de l'opération à effectuer pour que les données de départ soient à l'état souhaité. Par conséquent, toute modification apportée aux données effectuée en dehors des migrations peut être perdue ou provoquer une erreur.

Par exemple, cela permet de configurer les données de départ pour une `Blog` dans `OnModelCreating` :

```
modelBuilder.Entity<Blog>().HasData(new Blog { BlogId = 1, Url = "http://sample.com" });
```

Pour ajouter des entités qui ont une relation, les valeurs de clé étrangère doivent être spécifiées :

```
modelBuilder.Entity<Post>().HasData(
    new Post() { BlogId = 1, PostId = 1, Title = "First post", Content = "Test 1" });
```

Si le type d'entité a des propriétés dans l'état d'ombre, une classe anonyme peut être utilisée pour fournir les valeurs :

```
modelBuilder.Entity<Post>().HasData(
    new { BlogId = 1, PostId = 2, Title = "Second post", Content = "Test 2" });
```

Les types d'entités détenues peuvent être amorcés de la même façon :

```
modelBuilder.Entity<Post>().OwnsOne(p => p.AuthorName).HasData(
    new { PostId = 1, First = "Andriy", Last = "Svyryd" },
    new { PostId = 2, First = "Diego", Last = "Vega" });
```

Pour plus de contexte, consultez l' [exemple de projet complet](#).

Une fois les données ajoutées au modèle, des [migrations](#) doivent être utilisées pour appliquer les modifications.

#### TIP

Si vous devez appliquer des migrations dans le cadre d'un déploiement automatisé, vous pouvez créer un [script SQL](#) qui peut être visualisé avant l'exécution.

Vous pouvez également utiliser `context.Database.EnsureCreated()` pour créer une base de données contenant les données de départ, par exemple pour une base de données de test ou lorsque vous utilisez le fournisseur en mémoire ou une base de données non relation. Notez que si la base de données existe déjà, `EnsureCreated()` ne met pas à jour le schéma et n'amorce pas les données dans la base de données. Pour les bases de données relationnelles, vous ne devez pas appeler `EnsureCreated()` si vous envisagez d'utiliser des migrations.

#### Limitations des données de la valeur initiale du modèle

Ce type de données de départ est géré par des migrations et le script permettant de mettre à jour les données qui se trouvent déjà dans la base de données doit être généré sans connexion à la base de données. Cela impose des restrictions :

- La valeur de clé primaire doit être spécifiée même si elle est généralement générée par la base de données. Il sera utilisé pour détecter les modifications de données entre les migrations.
- Les données précédemment amorcées seront supprimées si la clé primaire est modifiée d'une façon ou d'une autre.

Par conséquent, cette fonctionnalité est très utile pour les données statiques qui ne sont pas censées changer en dehors des migrations et ne dépend pas d'autres éléments de la base de données, par exemple des codes POSTaux.

Si votre scénario comprend l'un des éléments suivants, il est recommandé d'utiliser une logique d'initialisation personnalisée décrite dans la dernière section :

- Données temporaires pour le test
- Données qui dépendent de l'état de la base de données
- Données qui ont besoin de valeurs clés pour être générées par la base de données, y compris les entités qui utilisent d'autres clés comme identité
- Données qui requièrent une transformation personnalisée (qui n'est pas gérée par les [conversions de valeurs](#)), comme un hachage de mot de passe
- Données nécessitant des appels à l'API externe, par exemple ASP.NET Core des rôles d'identité et la création d'utilisateurs

## Personnalisation manuelle de la migration

Lors de l'ajout d'une migration, les modifications apportées aux données spécifiées avec `HasData` sont transformées en appels à `InsertData()`, `UpdateData()` et `DeleteData()`. L'une des façons de contourner certaines des limitations de `HasData` consiste à ajouter manuellement ces appels ou [opérations personnalisées](#) à la migration.

```
migrationBuilder.InsertData(
    table: "Blogs",
    columns: new[] { "Url" },
    values: new object[] { "http://generated.com" });
```

## Logique d'initialisation personnalisée

Une façon simple et efficace d'effectuer un amorçage de données consiste à utiliser `DbContext.SaveChanges()` avant le début de l'exécution de la logique principale de l'application.

```
using (var context = new DataSeedingContext())
{
    context.Database.EnsureCreated();

    var testBlog = context.Blogs.FirstOrDefault(b => b.Url == "http://test.com");
    if (testBlog == null)
    {
        context.Blogs.Add(new Blog { Url = "http://test.com" });
    }
    context.SaveChanges();
}
```

### WARNING

Le code d'amorçage ne doit pas faire partie de l'exécution normale de l'application, car cela peut entraîner des problèmes de concurrence lorsque plusieurs instances sont en cours d'exécution et que l'application a également l'autorisation de modifier le schéma de la base de données.

Selon les contraintes de votre déploiement, le code d'initialisation peut être exécuté de différentes manières :

- Exécution locale de l'application d'initialisation
- Déploiement de l'application d'initialisation avec l'application principale, appel de la routine d'initialisation et désactivation ou suppression de l'application d'initialisation.

Cela peut généralement être automatisé à l'aide de [profils de publication](#).

# Types d'entité avec constructeurs

25/10/2019 • 11 minutes to read

## NOTE

Cette fonctionnalité est une nouveauté d'EF Core 2.1.

À compter de EF Core 2.1, il est maintenant possible de définir un constructeur avec des paramètres et d'avoir EF Core appeler ce constructeur lors de la création d'une instance de l'entité. Les paramètres de constructeur peuvent être liés à des propriétés mappées ou à différents genres de services pour faciliter les comportements tels que le chargement différé.

## NOTE

À partir de EF Core 2.1, toute la liaison de constructeur est par Convention. La configuration de constructeurs spécifiques à utiliser est prévue pour une version ultérieure.

## Lier à des propriétés mappées

Prenons l'exemple d'un modèle de blog/publication classique :

```
public class Blog
{
    public int Id { get; set; }

    public string Name { get; set; }
    public string Author { get; set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public int Id { get; set; }

    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

    public Blog Blog { get; set; }
}
```

Lorsque EF Core crée des instances de ces types, par exemple pour les résultats d'une requête, il commence par appeler le constructeur sans paramètre par défaut, puis affecte à chaque propriété la valeur de la base de données. Toutefois, si EF Core trouve un constructeur paramétrable avec des noms et des types de paramètres qui correspondent à ceux des propriétés mappées, il appellera plutôt le constructeur paramétrable avec des valeurs pour ces propriétés et ne définira pas explicitement chaque propriété. Exemple :

```

public class Blog
{
    public Blog(int id, string name, string author)
    {
        Id = id;
        Name = name;
        Author = author;
    }

    public int Id { get; set; }

    public string Name { get; set; }
    public string Author { get; set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public Post(int id, string title, DateTime postedOn)
    {
        Id = id;
        Title = title;
        PostedOn = postedOn;
    }

    public int Id { get; set; }

    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

    public Blog Blog { get; set; }
}

```

Voici quelques points à noter :

- Toutes les propriétés ne doivent pas avoir de paramètres de constructeur. Par exemple, la propriété poster. Content n'est pas définie par un paramètre de constructeur, donc EF Core la définira après avoir appelé le constructeur de manière normale.
- Les types et les noms de paramètres doivent correspondre aux types de propriétés et aux noms, à ceci près que les propriétés peuvent respecter la casse Pascal alors que les paramètres sont en casse mixte.
- EF Core ne pouvez pas définir de propriétés de navigation (telles que les blogs ou les billets ci-dessus) à l'aide d'un constructeur.
- Le constructeur peut être public, privé ou avoir une autre accessibilité. Toutefois, les proxies à chargement différé requièrent que le constructeur soit accessible à partir de la classe proxy qui hérite. Cela signifie généralement qu'il devient public ou protégé.

### **Propriétés en lecture seule**

Une fois les propriétés définies par le biais du constructeur, il peut être judicieux de les rendre accessibles en lecture seule. EF Core prend cela en charge, mais il y a quelques éléments à consulter :

- Les propriétés sans Setter ne sont pas mappées par Convention. (Cela a tendance à mapper les propriétés qui ne doivent pas être mappées, telles que les propriétés calculées.)
- L'utilisation de valeurs de clés générées automatiquement nécessite une propriété de clé en lecture-écriture, puisque la valeur de clé doit être définie par le générateur de clé lors de l'insertion de nouvelles entités.

Un moyen simple d'éviter ces choses consiste à utiliser des accesseurs set privés. Exemple :

```

public class Blog
{
    public Blog(int id, string name, string author)
    {
        Id = id;
        Name = name;
        Author = author;
    }

    public int Id { get; private set; }

    public string Name { get; private set; }
    public string Author { get; private set; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    public Post(int id, string title, DateTime postedOn)
    {
        Id = id;
        Title = title;
        PostedOn = postedOn;
    }

    public int Id { get; private set; }

    public string Title { get; private set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; private set; }

    public Blog Blog { get; set; }
}

```

EF Core voit une propriété avec un accesseur Set privé en lecture-écriture, ce qui signifie que toutes les propriétés sont mappées comme avant et que la clé peut toujours être générée par le magasin.

Une alternative à l'utilisation des accesseurs set privés consiste à définir des propriétés en lecture seule et à ajouter des mappages plus explicites dans OnModelCreating. De même, certaines propriétés peuvent être complètement supprimées et remplacées par des champs uniquement. Par exemple, considérez les types d'entités suivants :

```

public class Blog
{
    private int _id;

    public Blog(string name, string author)
    {
        Name = name;
        Author = author;
    }

    public string Name { get; }
    public string Author { get; }

    public ICollection<Post> Posts { get; } = new List<Post>();
}

public class Post
{
    private int _id;

    public Post(string title, DateTime postedOn)
    {
        Title = title;
        PostedOn = postedOn;
    }

    public string Title { get; }
    public string Content { get; set; }
    public DateTime PostedOn { get; }

    public Blog Blog { get; set; }
}

```

Et cette configuration dans OnModelCreating :

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>(
        b =>
        {
            b.HasKey("_id");
            b.Property(e => e.Author);
            b.Property(e => e.Name);
        });

    modelBuilder.Entity<Post>(
        b =>
        {
            b.HasKey("_id");
            b.Property(e => e.Title);
            b.Property(e => e.PostedOn);
        });
}

```

Points à noter :

- La clé « Property » est désormais un champ. Ce n'est pas un champ `readonly` afin que les clés générées par le magasin puissent être utilisées.
- Les autres propriétés sont des propriétés en lecture seule définies uniquement dans le constructeur.
- Si la valeur de clé primaire n'est jamais définie par EF ou lue à partir de la base de données, il n'est pas nécessaire de l'inclure dans le constructeur. Cela laisse la clé « Property » en tant que champ simple et précise qu'elle ne doit pas être définie explicitement lors de la création de nouveaux blogs ou publications.

#### NOTE

Ce code génère l'avertissement du compilateur '169', indiquant que le champ n'est jamais utilisé. Cela peut être ignoré, car en réalité EF Core utilise le champ d'une manière extralinguistique.

## Injection de services

EF Core pouvez également injecter des « services » dans le constructeur d'un type d'entité. Par exemple, vous pouvez injecter les éléments suivants :

- `DbContext` -l'instance de contexte actuelle, qui peut également être typée comme votre type DbContext dérivé
- `ILazyLoader` -le service de chargement différé--consultez la [documentation sur le chargement différé](#) pour plus d'informations
- `Action<object, string>` -un délégué de chargement différé--consultez la [documentation sur le chargement différé](#) pour plus d'informations
- `IEntityType` -les métadonnées de EF Core associées à ce type d'entité

#### NOTE

À partir de EF Core 2.1, seuls les services connus par EF Core peuvent être injectés. La prise en charge de l'injection des services d'application est prise en compte pour une version ultérieure.

Par exemple, un DbContext injecté peut être utilisé pour accéder de manière sélective à la base de données afin d'obtenir des informations sur les entités associées sans les charger toutes. Dans l'exemple ci-dessous, il est utilisé pour obtenir le nombre de publications dans un blog sans charger les publications :

```

public class Blog
{
    public Blog()
    {
    }

    private Blog(BloggingContext context)
    {
        Context = context;
    }

    private BloggingContext Context { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }
    public string Author { get; set; }

    public ICollection<Post> Posts { get; set; }

    public int PostsCount
        => Posts?.Count
        ?? Context?.Set<Post>().Count(p => Id == EF.Property<int?>(p, "BlogId"))
        ?? 0;
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public DateTime PostedOn { get; set; }

    public Blog Blog { get; set; }
}

```

Voici quelques points à noter :

- Le constructeur est privé, car il n'est jamais appelé par EF Core, et il existe un autre constructeur public pour une utilisation générale.
- Le code qui utilise le service injecté (autrement dit, le contexte) est défensif contre la `null` pour gérer les cas où EF Core ne crée pas l'instance.
- Étant donné que le service est stocké dans une propriété en lecture/écriture, il est réinitialisé lorsque l'entité est attachée à une nouvelle instance de contexte.

#### **WARNING**

L'injection de `DbContext` comme celle-ci est souvent considérée comme un anti-modèle dans la mesure où elle couple vos types d'entité directement à EF Core. Examinez attentivement toutes les options avant d'utiliser l'injection de service comme celle-ci.

# Fractionnement de table

10/01/2020 • 3 minutes to read

EF Core permet de mapper deux entités ou plus sur une seule ligne. C'est ce que l'on appelle le *fractionnement de table* ou le *partage de table*.

## Configuration

Pour utiliser le fractionnement de table, les types d'entités doivent être mappés à la même table, les clés primaires sont mappées aux mêmes colonnes et au moins une relation configurée entre la clé primaire d'un type d'entité et une autre dans la même table.

Un scénario courant de fractionnement de table n'utilise qu'un sous-ensemble des colonnes de la table pour améliorer les performances ou l'encapsulation.

Dans cet exemple `Order` représente un sous-ensemble de `DetailedOrder`.

```
public class Order
{
    public int Id { get; set; }
    public OrderStatus? Status { get; set; }
    public DetailedOrder DetailedOrder { get; set; }
}
```

```
public class DetailedOrder
{
    public int Id { get; set; }
    public OrderStatus? Status { get; set; }
    public string BillingAddress { get; set; }
    public string ShippingAddress { get; set; }
    public byte[] Version { get; set; }
}
```

Outre la configuration requise, nous appelons `Property(o => o.Status).HasColumnName("Status")` pour mapper `DetailedOrder.Status` à la même colonne que `Order.Status`.

```
modelBuilder.Entity<DetailedOrder>(dob =>
{
    dob.ToTable("Orders");
    dob.Property(o => o.Status).HasColumnName("Status");
});

modelBuilder.Entity<Order>(ob =>
{
    ob.ToTable("Orders");
    ob.Property(o => o.Status).HasColumnName("Status");
    obhasOne(o => o.DetailedOrder).WithOne()
        .HasForeignKey<DetailedOrder>(o => o.Id);
});
```

**TIP**

Pour plus de contexte, consultez l' [exemple de projet complet](#) .

## Contrôle

L'enregistrement et l'interrogation d'entités à l'aide du fractionnement de table s'effectuent de la même façon que les autres entités :

```
using (var context = new TableSplittingContext())
{
    context.Database.EnsureDeleted();
    context.Database.EnsureCreated();

    context.Add(new Order
    {
        Status = OrderStatus.Pending,
        DetailedOrder = new DetailedOrder
        {
            Status = OrderStatus.Pending,
            ShippingAddress = "221 B Baker St, London",
            BillingAddress = "11 Wall Street, New York"
        }
    });
}

context.SaveChanges();
}

using (var context = new TableSplittingContext())
{
    var pendingCount = context.Orders.Count(o => o.Status == OrderStatus.Pending);
    Console.WriteLine($"Current number of pending orders: {pendingCount}");
}

using (var context = new TableSplittingContext())
{
    var order = context.DetailedOrders.First(o => o.Status == OrderStatus.Pending);
    Console.WriteLine($"First pending order will ship to: {order.ShippingAddress}");
}
```

## Entité dépendante facultative

**NOTE**

Cette fonctionnalité a été introduite dans EF Core 3,0.

Si toutes les colonnes utilisées par une entité dépendante sont `NULL` dans la base de données, aucune instance n'est créée lors de la requête. Cela permet de modéliser une entité dépendante facultative, où la propriété de relation sur le principal est null. Notez que cela se produit également lorsque toutes les propriétés dépendantes sont facultatives et définies sur `null`, ce qui n'est peut-être pas prévu.

## Jetons d'accès concurrentiel

Si l'un des types d'entité partageant une table a un jeton d'accès concurrentiel, il doit également être inclus dans tous les autres types d'entités. Cela est nécessaire pour éviter une valeur de jeton d'accès concurrentiel obsolète lorsqu'une seule des entités mappées à la même table est mise à jour.

Pour éviter d'exposer le jeton d'accès concurrentiel au code consommateur, il est possible de créer un en tant que propriété **Shadow**:

```
modelBuilder.Entity<Order>()
    .Property<byte[]>("Version").IsRowVersion().HasColumnName("Version");

modelBuilder.Entity<DetailedOrder>()
    .Property(o => o.Version).IsRowVersion().HasColumnName("Version");
```

# Types d'entité détenus

10/01/2020 • 16 minutes to read

EF Core vous permet de modéliser des types d'entité qui peuvent uniquement apparaître dans les propriétés de navigation d'autres types d'entités. Il s'agit de *types d'entités détenues*. L'entité contenant un type d'entité détenu est son *propriétaire*.

Les entités détenues sont essentiellement une partie du propriétaire et ne peuvent pas exister sans elles, elles sont conceptuellement similaires aux [agrégats](#). Cela signifie que le type détenu est par définition sur le côté dépendant de la relation avec le propriétaire.

## Configuration explicite

Les types d'entités détenus ne sont jamais inclus par EF Core dans le modèle par Convention. Vous pouvez utiliser la méthode `OwnsOne` dans `OnModelCreating` ou annoter le type avec `OwnedAttribute` (nouveauté de EF Core 2.1) pour configurer le type en tant que type détenu.

Dans cet exemple, `StreetAddress` est un type sans propriété d'identité. Il est utilisé comme propriété du type `Order` pour spécifier l'adresse d'expédition d'une commande particulière.

Nous pouvons utiliser la `OwnedAttribute` pour la traiter comme une entité possédée lorsqu'elle est référencée à partir d'un autre type d'entité :

```
[Owned]  
public class StreetAddress  
{  
    public string Street { get; set; }  
    public string City { get; set; }  
}
```

```
public class Order  
{  
    public int Id { get; set; }  
    public StreetAddress ShippingAddress { get; set; }  
}
```

Il est également possible d'utiliser la méthode `OwnsOne` dans `OnModelCreating` pour spécifier que la propriété `ShippingAddress` est une entité appartenant au type d'entité `Order` et pour configurer des facettes supplémentaires si nécessaire.

```
modelBuilder.Entity<Order>().OwnsOne(p => p.ShippingAddress);
```

Si la propriété `ShippingAddress` est privée dans le type `Order`, vous pouvez utiliser la version de chaîne de la méthode `OwnsOne` :

```
modelBuilder.Entity<Order>().OwnsOne(typeof(StreetAddress), "ShippingAddress");
```

Pour plus de contexte, consultez l' [exemple de projet complet](#) .

## Clés implicites

Les types détenus configurés avec `OwnsOne` ou découverts par le biais d'une navigation de référence ont toujours une relation un-à-un avec le propriétaire, donc ils n'ont pas besoin de leurs propres valeurs de clés, car les valeurs de clé étrangère sont uniques. Dans l'exemple précédent, le type de `StreetAddress` n'a pas besoin de définir une propriété de clé.

Pour comprendre comment EF Core effectue le suivi de ces objets, il est utile de savoir qu'une clé primaire est créée en tant que [propriété Shadow](#) pour le type détenu. La valeur de la clé d'une instance du type détenu sera identique à la valeur de la clé de l'instance propriétaire.

## Collections de types détenus

### NOTE

Cette fonctionnalité est une nouveauté d'EF Core 2.2.

Pour configurer une collection de types détenus, utilisez `OwnsMany` dans `OnModelCreating`.

Les types détenus nécessitent une clé primaire. S'il n'existe aucune propriété candidate correcte sur le type .NET, EF Core pouvez essayer d'en créer un. Toutefois, lorsque des types détenus sont définis par le biais d'une collection, il suffit de créer une propriété Shadow pour agir à la fois comme clé étrangère dans le propriétaire et la clé primaire de l'instance appartenant, comme nous le faisons pour `OwnsOne` : il peut y avoir plusieurs instances de type détenu pour chaque propriétaire et, par conséquent, la clé du propriétaire n'est pas suffisante pour fournir une identité unique pour

Les deux solutions les plus simples à ce niveau sont les suivantes :

- Définition d'une clé primaire de substitution sur une nouvelle propriété indépendante de la clé étrangère qui pointe vers le propriétaire. Les valeurs contenues doivent être uniques pour l'ensemble des propriétaires (par exemple, si le {1} parent a des {1}enfants, alors {2} parent ne peut pas avoir d'{1}enfant), de sorte que la valeur n'a pas de signification inhérente. Étant donné que la clé étrangère ne fait pas partie de la clé primaire, ses valeurs peuvent être modifiées. vous pouvez donc déplacer un enfant d'un parent à un autre, mais cela est généralement dû à une sémantique d'agrégation.
- En utilisant la clé étrangère et une propriété supplémentaire comme clé composite. La valeur de propriété supplémentaire ne doit désormais être unique que pour un parent donné (par conséquent, si le parent {1} a un enfant {1,1}, le {2} parent peut toujours avoir des {2,1}enfants). En faisant de la clé étrangère de la clé primaire, la relation entre le propriétaire et l'entité détenu devient immuable et reflète mieux la sémantique d'agrégation. C'est ce que EF Core par défaut.

Dans cet exemple, nous allons utiliser la classe `Distributor` :

```
public class Distributor
{
    public int Id { get; set; }
    public ICollection<StreetAddress> ShippingCenters { get; set; }
}
```

Par défaut, la clé primaire utilisée pour le type détenu référencé par l'intermédiaire de la propriété de navigation `ShippingCenters` est `("DistributorId", "Id")` où `"DistributorId"` est le FK et `"Id"` est une valeur `int` unique.

Pour configurer un autre appel de PK `HasKey` :

```
modelBuilder.Entity<Distributor>().OwnsMany(p => p.ShippingCenters, a =>
{
    a.WithOwner().HasForeignKey("OwnerId");
    a.Property<int>("Id");
    a.HasKey("Id");
});
```

#### NOTE

Avant de EF Core 3,0 `WithOwner()` méthode n'existe pas, cet appel doit être supprimé. En outre, la clé primaire n'a pas été détectée automatiquement. elle a donc toujours été spécifiée.

## Mappage de types détenus avec le fractionnement de table

Lorsque vous utilisez des bases de données relationnelles, par défaut, les types appartenant à la référence sont mappés à la même table que le propriétaire. Cela nécessite le fractionnement de la table en deux : certaines colonnes seront utilisées pour stocker les données du propriétaire, et certaines colonnes seront utilisées pour stocker les données de l'entité détenue. Il s'agit d'une fonctionnalité courante connue sous le nom de [fractionnement de table](#).

Par défaut, EF Core nommera les colonnes de base de données pour les propriétés du type d'entité détenue, en suivant le modèle `Navigation_OwnedEntityProperty`. Par conséquent, les propriétés du `StreetAddress` s'affichent dans la table Orders avec les noms « `ShippingAddress_Street` » et « `ShippingAddress_City` ».

Vous pouvez utiliser la méthode `HasColumnName` pour renommer ces colonnes :

```
modelBuilder.Entity<Order>().OwnsOne(
    o => o.ShippingAddress,
    sa =>
{
    sa.Property(p => p.Street).HasColumnName("ShipsToStreet");
    sa.Property(p => p.City).HasColumnName("ShipsToCity");
});
```

#### NOTE

La plupart des méthodes de configuration de type d'entité normales telles que `ignore` peuvent être appelées de la même façon.

## Partage du même type .NET entre plusieurs types détenus

Un type d'entité détenu peut être du même type .NET qu'un autre type d'entité détenue. par conséquent, le type .NET peut ne pas être suffisant pour identifier un type détenue.

Dans ce cas, la propriété qui pointe du propriétaire vers l'entité détenue devient la définition de la *navigation* du type d'entité détenue. Du point de vue de EF Core, la navigation de définition fait partie de l'identité du type en même temps que le type .NET.

Par exemple, dans la classe suivante `ShippingAddress` et `BillingAddress` sont tous deux du même type .NET, `StreetAddress` :

```

public class OrderDetails
{
    public DetailedOrder Order { get; set; }
    public StreetAddress BillingAddress { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}

```

Pour comprendre comment EF Core doit distinguer les instances suivies de ces objets, il peut être utile de penser que la navigation de définition est devenue partie intégrante de la clé de l'instance parallèlement à la valeur de la clé du propriétaire et du type .NET du type détenu.

## Types détenus imbriqués

Dans cet exemple `OrderDetails` possède `BillingAddress` et `ShippingAddress`, qui sont tous deux des types de `StreetAddress`. `OrderDetails` est alors détenu par le type `DetailedOrder`.

```

public class DetailedOrder
{
    public int Id { get; set; }
    public OrderDetails OrderDetails { get; set; }
    public OrderStatus Status { get; set; }
}

```

```

public enum OrderStatus
{
    Pending,
    Shipped
}

```

Chaque navigation vers un type détenu définit un type d'entité distinct avec une configuration totalement indépendante.

En plus des types détenus imbriqués, un type détenu peut faire référence à une entité normale, il peut s'agir du propriétaire ou d'une entité différente tant que l'entité qui en est la propriété est sur le côté dépendant. Cette fonctionnalité définit les types d'entités détenues, à l'exception des types complexes dans EF6.

```

public class OrderDetails
{
    public DetailedOrder Order { get; set; }
    public StreetAddress BillingAddress { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}

```

Il est possible de chaîner la méthode `OwesOne` dans un appel Fluent pour configurer ce modèle :

```

modelBuilder.Entity<DetailedOrder>().OwesOne(p => p.OrderDetails, od =>
{
    od.WithOwner(d => d.Order);
    od.OwesOne(c => c.BillingAddress);
    od.OwesOne(c => c.ShippingAddress);
});

```

Notez le `WithOwner` appel utilisé pour configurer la propriété de navigation qui pointe vers le propriétaire. Pour configurer une navigation vers le type d'entité propriétaire qui ne fait pas partie de la relation de propriété `WithOwner()` devez être appelé sans argument.

Il est possible d'obtenir le résultat à l'aide de `OwnedAttribute` à la fois sur `OrderDetails` et `StreetAddress`.

## Stockage des types détenus dans des tables distinctes

Contrairement aux types complexes EF6, les types détenus peuvent être stockés dans une table distincte du propriétaire. Pour remplacer la Convention qui mappe un type détenu à la même table que le propriétaire, vous pouvez simplement appeler `ToTable` et fournir un autre nom de table. L'exemple suivant mappe `OrderDetails` et ses deux adresses à une table distincte à partir de `DetailedOrder`:

```
modelBuilder.Entity<DetailedOrder>().OwnsOne(p => p.OrderDetails, od =>
{
    od.ToTable("OrderDetails");
});
```

Il est également possible d'utiliser le `TableAttribute` pour y parvenir, mais notez que cela échouera s'il existe plusieurs navigations dans le type détenu dans la mesure où, dans ce cas, plusieurs types d'entité seraient mappés à la même table.

## Interrogation des types détenus

Quand le propriétaire fait l'objet d'une interrogation, les types détenus sont inclus par défaut. Il n'est pas nécessaire d'utiliser la méthode `Include`, même si les types détenus sont stockés dans une table distincte. En fonction du modèle décrit précédemment, la requête suivante obtiendra `Order`, `OrderDetails` et les deux `StreetAddresses` appartenant à la base de données :

```
var order = context.DetailedOrders.First(o => o.Status == OrderStatus.Pending);
Console.WriteLine($"First pending order will ship to: {order.OrderDetails.ShippingAddress.City}");
```

## Limitations

Certaines de ces limitations sont essentielles à la façon dont les types d'entités détenus fonctionnent, mais d'autres sont des restrictions que nous pouvons être en mesure de supprimer dans les versions ultérieures :

### Restrictions par conception

- Vous ne pouvez pas créer un `DbSet<T>` pour un type détenu
- Vous ne pouvez pas appeler `Entity<T>()` avec un type détenu sur `ModelBuilder`

### Lacunes actuelles

- Les types d'entités détenues ne peuvent pas avoir de hiérarchies d'héritage
- Les navigations de référence vers les types d'entité détenus ne peuvent pas avoir la valeur null, sauf si elles sont explicitement mappées à une table distincte du propriétaire
- Les instances de types d'entité possédées ne peuvent pas être partagées par plusieurs propriétaires (il s'agit d'un scénario connu pour les objets de valeur qui ne peuvent pas être implémentés à l'aide des types d'entités détenus)

### Lacunes dans les versions précédentes

- Dans EF Core 2,0, les navigations vers les types d'entités détenues ne peuvent pas être déclarées dans des types d'entité dérivés, à moins que les entités détenues soient explicitement mappées à une table distincte de la hiérarchie de propriétaire. Cette limitation a été supprimée dans EF Core 2,1
- Dans EF Core 2,0 et 2,1, seules les navigations de référence vers les types détenus étaient prises en charge. Cette limitation a été supprimée dans EF Core 2,2

# Types d'entité sans clé

05/12/2019 • 7 minutes to read

## NOTE

Cette fonctionnalité a été ajoutée à EF Core 2,1 sous le nom des types de requêtes. Dans EF Core 3,0, le concept a été renommé en types d'entité sans clé.

En plus des types d'entités standard, un modèle de EF Core peut contenir des \_types d'entité sans\_clé, qui peuvent être utilisés pour exécuter des requêtes de base de données sur des données qui ne contiennent pas de valeurs de clés.

## Caractéristiques des types d'entité sans clé

Les types d'entité sans clé prennent en charge un grand nombre des mêmes fonctionnalités de mappage que les types d'entités standard, tels que le mappage d'héritage et les propriétés de navigation. Sur des magasins relationnels, ils peuvent configurer les objets de base de données cible et les colonnes par le biais de méthodes de l'API fluent ou des annotations de données.

Toutefois, ils diffèrent des types d'entités standard en ce qu'ils :

- Impossible d'avoir une clé définie.
- Ne sont jamais suivis pour les modifications apportées à `DbContext` et, par conséquent, ne sont jamais insérées, mises à jour ou supprimées dans la base de données.
- Ne sont jamais découverts par convention.
- Ne prennent en charge qu'un sous-ensemble de fonctionnalités de mappage de navigation, notamment :
  - Ils ne peuvent jamais agir en tant que la terminaison principale d'une relation.
  - Ils n'ont peut-être pas de navigation vers les entités détenues
  - Ils peuvent uniquement contenir des propriétés de navigation de référence pointant vers des entités normales.
  - Les entités ne peuvent pas contenir de propriétés de navigation pour les types d'entités Keyless.
- Doit être configuré avec `.HasNoKey()` appel de méthode.
- Peut être mappé à une *requête de définition*. Une requête de définition est une requête déclarée dans le modèle qui joue le rôle d'une source de données pour un type d'entité sans clé.

## Scénarios d'utilisation

Voici quelques-uns des principaux scénarios d'utilisation pour les types d'entité sans clé :

- Servir de type de retour pour les [requêtes SQL brutes](#).
- Mappage à des vues de base de données qui ne contiennent pas de clé primaire.
- Mappage de tables qui n'ont pas d'une clé primaire définie.
- Mappage à des requêtes définies dans le modèle.

## Mappage aux objets de base de données

Le mappage d'un type d'entité clé-inférieur à un objet de base de données s'effectue à l'aide de l'API Fluent `ToTable` ou `ToView`. Du point de vue d'EF Core, l'objet de base de données spécifié dans cette méthode est un

*vue*, ce qui signifie qu'il est traité comme une source de la requête en lecture seule et ne peut pas être la cible de mise à jour, insérer ou supprimer des opérations. Toutefois, cela ne signifie pas que l'objet de base de données est réellement requis pour être une vue de base de données. Il peut également s'agir d'une table de base de données qui sera traitée en lecture seule. À l'inverse, pour les types d'entités standard, EF Core suppose qu'un objet de base de données spécifié dans la méthode `ToTable` peut être traité comme une *table*, ce qui signifie qu'il peut être utilisé comme source de requête, mais également ciblé par les opérations Update, DELETE et insert. En fait, vous pouvez spécifier le nom d'une vue de base de données dans `ToTable` et tout devrait fonctionner correctement tant que la vue est configurée pour être mis à jour sur la base de données.

#### NOTE

`ToView` suppose que l'objet existe déjà dans la base de données et qu'il n'est pas créé par des migrations.

## Exemple

L'exemple suivant montre comment utiliser les types d'entités Keyless pour interroger une vue de base de données.

#### TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

Tout d'abord, nous définissons un modèle simple de Blog et Post :

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
}
```

Ensuite, nous définissons une vue de base de données simple qui nous permettra d'interroger le nombre de messages associée à chaque blog :

```
db.Database.ExecuteSqlRaw(
    @"CREATE VIEW View_BlogPostCounts AS
        SELECT b.Name, Count(p.PostId) as PostCount
        FROM Blogs b
        JOIN Posts p on p.BlogId = b.BlogId
        GROUP BY b.Name");
```

Ensuite, nous définissons une classe pour contenir le résultat de la vue de base de données :

```
public class BlogPostsCount
{
    public string BlogName { get; set; }
    public int PostCount { get; set; }
}
```

Ensuite, vous configurez le type d'entité « sans clé » dans `OnModelCreating` à l'aide de l'API `HasNoKey`. Nous utilisons l'API de configuration Fluent pour configurer le mappage pour le type d'entité « sans clé » :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder
        .Entity<BlogPostsCount>(eb =>
    {
        eb.HasNoKey();
        eb.ToView("View_BlogPostCounts");
        eb.Property(v => v.BlogName).HasColumnName("Name");
    });
}
```

Ensuite, nous configurons le `DbContext` pour inclure le `DbSet<T>` :

```
public DbSet<BlogPostsCount> BlogPostCounts { get; set; }
```

Enfin, nous pouvons interroger la vue de base de données de manière standard :

```
var postCounts = db.BlogPostCounts.ToList();

foreach (var postCount in postCounts)
{
    Console.WriteLine($"{postCount.BlogName} has {postCount.PostCount} posts.");
    Console.WriteLine();
}
```

#### TIP

Notez que nous avons également défini une propriété de requête au niveau du contexte (DbSet) pour agir en tant que racine pour les requêtes sur ce type.

# Alternance entre plusieurs modèles ayant le même type DbContext

10/01/2020 • 2 minutes to read

Le modèle intégré `OnModelCreating` peut utiliser une propriété du contexte pour modifier la façon dont le modèle est généré. Par exemple, supposons que vous souhaitez configurer une entité différemment en fonction de certaines propriétés :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    if (UseIntProperty)
    {
        modelBuilder.Entity<ConfigurableEntity>().Ignore(e => e.StringProperty);
    }
    else
    {
        modelBuilder.Entity<ConfigurableEntity>().Ignore(e => e.IntProperty);
    }
}
```

Malheureusement, ce code ne fonctionnerait pas tel quel, car EF génère le modèle et s'exécute `OnModelCreating` une seule fois, en mettant en cache le résultat pour des raisons de performances. Toutefois, vous pouvez vous connecter au mécanisme de mise en cache du modèle pour que EF prenne en charge la propriété qui produit des modèles différents.

## IModelCacheKeyFactory

EF utilise la `IModelCacheKeyFactory` pour générer des clés de cache pour les modèles ; par défaut, EF suppose que, pour tout type de contexte donné, le modèle sera le même, l'implémentation par défaut de ce service retourne donc une clé qui contient simplement le type de contexte. Pour produire différents modèles à partir du même type de contexte, vous devez remplacer le service `IModelCacheKeyFactory` par l'implémentation correcte. La clé générée est comparée à d'autres clés de modèle à l'aide de la méthode `Equals`, en tenant compte de toutes les variables qui affectent le modèle :

L'implémentation suivante prend en compte les `IgnoreIntProperty` lors de la génération d'une clé de cache de modèle :

```
public class DynamicModelCacheKeyFactory : IModelCacheKeyFactory
{
    public object Create(DbContext context)
        => context is DynamicContext dynamicContext
            ? (context.GetType(), dynamicContext.UseIntProperty)
            : (object)context.GetType();
}
```

Enfin, inscrivez votre nouveau `IModelCacheKeyFactory` dans le `OnConfiguring` de votre contexte :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder
        .UseInMemoryDatabase("DynamicContext")
        .ReplaceService<IModelCacheKeyFactory, DynamicModelCacheKeyFactory>();
```

Pour plus de contexte, consultez l' [exemple de projet complet](#) .

# Données spatiales

09/01/2020 • 14 minutes to read

## NOTE

Cette fonctionnalité a été ajoutée dans EF Core 2.2.

Les données spatiales représentent l'emplacement physique et la forme des objets. De nombreuses bases de données prennent en charge ce type de données afin qu'elles puissent être indexées et interrogées avec d'autres données. Les scénarios courants incluent l'interrogation d'objets situés à une distance donnée à partir d'un emplacement ou la sélection de l'objet dont la bordure contient un emplacement donné. EF Core prend en charge le mappage aux types de données spatiales à l'aide de la bibliothèque spatiale [NetTopologySuite](#).

## Installation de .

Pour pouvoir utiliser les données spatiales avec EF Core, vous devez installer le package NuGet de prise en charge approprié. Le package que vous devez installer dépend du fournisseur que vous utilisez.

| FOURNISSEUR EF CORE                     | PACKAGE NUGET SPATIAL  |
|---|--|
| Microsoft.EntityFrameworkCore.SqlServer | <a href="#">Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite</a> |
| Microsoft.EntityFrameworkCore.Sqlite    | <a href="#">Microsoft.EntityFrameworkCore.sqlite.NetTopologySuite</a>    |
| Microsoft.EntityFrameworkCore.InMemory  | <a href="#">NetTopologySuite</a>   |
| Npgsql.EntityFrameworkCore.PostgreSQL   | <a href="#">Npgsql.EntityFrameworkCore.PostgreSQL.NetTopologySuite</a>   |

## Reconstitution de la logique des produits

Les packages NuGet spatiaux activent également les modèles d'[ingénierie à rebours](#) avec des propriétés spatiales, mais vous devez installer le package **avant** d'exécuter `Scaffold-DbContext` ou `dotnet ef dbcontext scaffold`. Si vous ne le souhaitez pas, vous recevrez des avertissements sur les mappages de type pour les colonnes et les colonnes seront ignorées.

## NetTopologySuite (NTS)

NetTopologySuite est une bibliothèque spatiale pour .NET. EF Core permet le mappage aux types de données spatiales dans la base de données à l'aide des types NTS dans votre modèle.

Pour activer le mappage aux types spatiaux via NTS, appelez la méthode `UseNetTopologySuite` sur le générateur d'options `DbContext` du fournisseur. Par exemple, avec SQL Server vous l'appellerez comme ceci.

```
optionsBuilder.UseSqlServer(  
    @"Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=WideWorldImporters",  
    x => x.UseNetTopologySuite());
```

Il existe plusieurs types de données spatiales. Le type que vous utilisez dépend des types de formes que vous souhaitez autoriser. Voici la hiérarchie des types NTS que vous pouvez utiliser pour les propriétés de votre modèle.

Ils sont situés dans l'espace de noms `NetTopologySuite.Geometries`.

- Géométrie
  - rapport
  - LineString
  - Polygone
  - GeometryCollection
    - MultiPoint
    - MultiLineString
    - MultiPolygon

#### WARNING

`CircularString`, `CompoundCurve` et `CurePolygon` ne sont pas pris en charge par NTS.

L'utilisation du type `Geometry` de base permet à n'importe quel type de forme d'être spécifié par la propriété.

Les classes d'entité suivantes peuvent être utilisées pour mapper à des tables dans l'[exemple de base de données grand WorldImporters](#).

```
[Table("Cities", Schema = "Application")]
class City
{
    public int CityID { get; set; }

    public string CityName { get; set; }

    public Point Location { get; set; }
}

[Table("Countries", Schema = "Application")]
class Country
{
    public int CountryID { get; set; }

    public string CountryName { get; set; }

    // Database includes both Polygon and MultiPolygon values
    public Geometry Border { get; set; }
}
```

#### Création de valeurs

Vous pouvez utiliser des constructeurs pour créer des objets `Geometry` ; Toutefois, NTS recommande d'utiliser à la place une fabrique de géométrie. Cela vous permet de spécifier un SRID par défaut (le système de référence spatiale utilisé par les coordonnées) et vous donne le contrôle sur des éléments plus avancés tels que le modèle de précision (utilisé pendant les calculs) et la séquence de coordonnées (détermine les ordonnées-dimensions mesures et disponibles).

```
var geometryFactory = NtsGeometryServices.Instance.CreateGeometryFactory(srid: 4326);
var currentLocation = geometryFactory.CreatePoint(-122.121512, 47.6739882);
```

#### NOTE

4326 fait référence à WGS 84, une norme utilisée dans le GPS et d'autres systèmes géographiques.

## Longitude et Latitude

Les coordonnées en NTS sont exprimées en valeurs X et Y. Pour représenter la longitude et la latitude, utilisez X pour la longitude et Y pour la latitude. Notez que **c'est à partir du format** `latitude, longitude` dans lequel vous voyez généralement ces valeurs.

## SRID ignoré pendant les opérations du client

NTS ignore les valeurs SRID lors des opérations. Il part du principe qu'il s'agit d'un système de coordonnées planaire. Cela signifie que si vous spécifiez des coordonnées en termes de longitude et de latitude, certaines valeurs évaluées par le client, telles que la distance, la longueur et la zone, seront en degrés, et non en mètres. Pour les valeurs plus significatives, vous devez d'abord projeter les coordonnées dans un autre système de coordonnées à l'aide d'une bibliothèque comme [ProjNet4GeoAPI](#) avant de calculer ces valeurs.

Si une opération est évaluée par le serveur par EF Core via SQL, l'unité du résultat est déterminée par la base de données.

Voici un exemple d'utilisation de ProjNet4GeoAPI pour calculer la distance entre deux villes.

```
static class GeometryExtensions
{
    static readonly CoordinateSystemServices _coordinateSystemServices
        = new CoordinateSystemServices(
            new CoordinateSystemFactory(),
            new CoordinateTransformationFactory(),
            new Dictionary<int, string>
            {
                // Coordinate systems:

                [4326] = GeographicCoordinateSystem.WGS84.WKT,

                // This coordinate system covers the area of our data.
                // Different data requires a different coordinate system.
                [2855] =
                @"
                    PROJCS[""NAD83(HARN) / Washington North"",
                        GEOGCS[""NAD83(HARN)"",
                            DATUM[""NAD83_High_Accuracy_Regional_Network"",
                                SPHEROID[""GRS 1980"",6378137,298.257222101,
                                    AUTHORITY[""EPSG"",""7019""}],
                                AUTHORITY[""EPSG"",""6152""}],
                            PRIMEM[""Greenwich"",
                                AUTHORITY[""EPSG"",""8901""}],
                            UNIT[""degree"",
                                AUTHORITY[""EPSG"",""9122""]],
                                AUTHORITY[""EPSG"",""4152""}],
                            PROJECTION[""Lambert_Conformal_Conic_2SP""],
                            PARAMETER[""standard_parallel_1"",48.73333333333333],
                            PARAMETER[""standard_parallel_2"",47.5],
                            PARAMETER[""latitude_of_origin"",47],
                            PARAMETER[""central_meridian"",-120.833333333333],
                            PARAMETER[""false_easting"",500000],
                            PARAMETER[""false_northing"",0],
                            UNIT[""metre"",
                                AUTHORITY[""EPSG"",""9001""]],
                                AUTHORITY[""EPSG"",""2855""]
                            "
                        });
}

public static Geometry ProjectTo(this Geometry geometry, int srid)
{
    var transformation = _coordinateSystemServices.CreateTransformation(geometry.SRID, srid);

    var result = geometry.Copy();
    result.Apply(new MathTransformFilter(transformation.MathTransform));
}
```

```

        return result;
    }

    class MathTransformFilter : ICoordinateSequenceFilter
    {
        readonly MathTransform _transform;

        public MathTransformFilter(MathTransform transform)
            => _transform = transform;

        public bool Done => false;
        public bool GeometryChanged => true;

        public void Filter(CoordinateSequence seq, int i)
        {
            var result = _transform.Transform(
                new[]
                {
                    seq.GetOrdinate(i, Ordinate.X),
                    seq.GetOrdinate(i, Ordinate.Y)
                });
            seq.SetOrdinate(i, Ordinate.X, result[0]);
            seq.SetOrdinate(i, Ordinate.Y, result[1]);
        }
    }
}

```

```

var seattle = new Point(-122.333056, 47.609722) { SRID = 4326 };
var redmond = new Point(-122.123889, 47.669444) { SRID = 4326 };

var distance = seattle.ProjectTo(2855).Distance(redmond.ProjectTo(2855));

```

## Interrogation des données

Dans LINQ, les méthodes et propriétés NTS disponibles en tant que fonctions de base de données sont traduites en SQL. Par exemple, les méthodes `distance` et `Contains` sont traduites dans les requêtes suivantes. Le tableau à la fin de cet article indique quels membres sont pris en charge par différents fournisseurs de EF Core.

```

var nearestCity = db.Cities
    .OrderBy(c => c.Location.Distance(currentLocation))
    .FirstOrDefault();

var currentCountry = db.Countries
    .FirstOrDefault(c => c.Border.Contains(currentLocation));

```

## SQL Server

Si vous utilisez SQL Server, vous devez connaître certaines choses supplémentaires.

### Geography ou Geometry

Par défaut, les propriétés spatiales sont mappées à des colonnes `geography` dans SQL Server. Pour utiliser `geometry`, [configurez le type de colonne](#) dans votre modèle.

### Anneaux de polygone Geography

Lors de l'utilisation du type de colonne `geography`, SQL Server impose des exigences supplémentaires sur l'anneau extérieur (ou l'interpréteur de commandes) et les anneaux intérieurs (ou trous). L'anneau extérieur doit être orienté vers le sens inverse des aiguilles d'une montre et des anneaux intérieurs. NTS valide cette valeur avant d'envoyer des valeurs à la base de données.

## FullGlobe

SQL Server a un type Geometry non standard pour représenter le globe complet lors de l'utilisation du type de colonne `geography`. Il offre également un moyen de représenter les polygones en fonction du monde entier (sans anneau extérieur). Aucun de ces deux n'est pris en charge par NTS.

### WARNING

Les FullGlobe et les polygones basés sur ce dernier ne sont pas pris en charge par NTS.

## SQLite

Voici quelques informations supplémentaires pour celles qui utilisent SQLite.

### Installation de SpatiaLite

Sur Windows, la bibliothèque de mod\_spatialite native est distribuée en tant que dépendance de package NuGet. D'autres plateformes doivent l'installer séparément. Cette opération s'effectue généralement à l'aide d'un gestionnaire de package logiciel. Par exemple, vous pouvez utiliser la fonction APT sur Ubuntu et homebrew sur MacOS.

```
# Ubuntu  
apt-get install libsqlite3-mod-spatialite  
  
# macOS  
brew install libspatialite
```

### Configuration de SRID

Dans SpatiaLite, les colonnes doivent spécifier un SRID par colonne. La valeur par défaut de SRID est `0`. Spécifiez un autre SRID à l'aide de la méthode `ForSqliteHasSrid`.

```
modelBuilder.Entity<City>().Property(c => c.Location)  
.ForSqliteHasSrid(4326);
```

### Dimension

Comme pour SRID, la dimension d'une colonne (ou ordonnée) est également spécifiée dans le cadre de la colonne. Les ordonnées par défaut sont X et Y. Activez des ordonnées supplémentaires (Z et M) à l'aide de la méthode `ForSqliteHasDimension`.

```
modelBuilder.Entity<City>().Property(c => c.Location)  
.ForSqliteHasDimension(Ordinates.XYZ);
```

## Opérations traduites

Ce tableau montre les membres NTS qui sont convertis en SQL par chaque fournisseur de EF Core.

| NETTOPOLOGYSUITE      | SQL SERVER<br>(GÉOMÉTRIE) | SQL SERVER<br>(GEOGRAPHY) | SQlite | Npgsql |
|-----------------------|---------------------------|---------------------------|--------|--------|
| Geometry. Area        | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. AsBinary () | ✓                         | ✓                         | ✓      | ✓      |

| NETTOPOLOGYSUITE                               | SQL SERVER<br>(GÉOMÉTRIE) | SQL SERVER<br>(GEOGRAPHY) | SQlite | Npgsql |
|--|---------------------------|---------------------------|--------|--------|
| Geometry. AsText ()                            | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. Boundary                             | ✓                         |                           | ✓      | ✓      |
| Geometry. buffer<br>(double)                   | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. buffer<br>(double, int)              |                           |                           | ✓      | ✓      |
| Geometry. Centre de<br>gravité                 | ✓                         |                           | ✓      | ✓      |
| Geometry. Contains<br>(Geometry)               | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>ConvexHull ()                     | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. CoveredBy<br>(Geometry)              |                           |                           | ✓      | ✓      |
| Geometry.<br>couvertures<br>(Geometry)         |                           |                           | ✓      | ✓      |
| Geometry. crosses<br>(Geometry)                | ✓                         |                           | ✓      | ✓      |
| Geometry. difference<br>(Geometry)             | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. dimension                            | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. disjointe<br>(Geometry)              | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. distance<br>(Geometry)               | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. Envelope                             | ✓                         |                           | ✓      | ✓      |
| Geometry.<br>EqualsExact<br>(Geometry)         |                           |                           |        | ✓      |
| Geometry.<br>EqualsTopologically<br>(Geometry) | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>GeometryType                      | ✓                         | ✓                         | ✓      | ✓      |

| NETTOPOLOGYSUITE                                    | SQL SERVER<br>(GÉOMÉTRIE) | SQL SERVER<br>(GEOGRAPHY) | SQlite | Npgsql |
|---|---------------------------|---------------------------|--------|--------|
| Geometry.<br>GetGeometryN (int)                     | ✓                         |                           | ✓      | ✓      |
| Geometry.<br>InteriorPoint                          | ✓                         |                           | ✓      | ✓      |
| Geometry.<br>intersection<br>(Geometry)             | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>intersections<br>(Geometry)            | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. IsEmpty                                   | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. IsSimple                                  | ✓                         |                           | ✓      | ✓      |
| Geometry. IsValid                                   | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>IsWithinDistance<br>(Geometry, double) | ✓                         |                           | ✓      | ✓      |
| Geometry. Length                                    | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>NumGeometries                          | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>NumPoints                              | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>OgcGeometryType                        | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>chevauchements<br>(Geometry)           | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>PointOnSurface                         | ✓                         |                           | ✓      | ✓      |
| Geometry. relate<br>(Geometry, String)              | ✓                         |                           | ✓      | ✓      |
| Geometry. Reverse ()                                |                           |                           | ✓      | ✓      |
| Geometry. SRID                                      | ✓                         | ✓                         | ✓      | ✓      |
| Geometry.<br>SymmetricDifference<br>(Geometry)      | ✓                         | ✓                         | ✓      | ✓      |

| NETTOPOLOGYSUITE                    | SQL SERVER<br>(GÉOMÉTRIE) | SQL SERVER<br>(GEOGRAPHY) | SQlite | Npgsql |
|-------------------------------------|---------------------------|---------------------------|--------|--------|
| Geometry. ToBinary ()               | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. ToText ()                 | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. touche<br>(Geometry)      | ✓                         |                           | ✓      | ✓      |
| Geometry. Union ()                  |                           |                           | ✓      | ✓      |
| Geometry. Union<br>(Geometry)       | ✓                         | ✓                         | ✓      | ✓      |
| Geometry. within<br>(Geometry)      | ✓                         | ✓                         | ✓      | ✓      |
| GeometryCollection.<br>Count        | ✓                         | ✓                         | ✓      | ✓      |
| GeometryCollection<br>[int]         | ✓                         | ✓                         | ✓      | ✓      |
| LineString. Count                   | ✓                         | ✓                         | ✓      | ✓      |
| LineString. point de<br>terminaison | ✓                         | ✓                         | ✓      | ✓      |
| LineString. GetPointN<br>(int)      | ✓                         | ✓                         | ✓      | ✓      |
| LineString. IsClosed                | ✓                         | ✓                         | ✓      | ✓      |
| LineString. IsRing                  | ✓                         |                           | ✓      | ✓      |
| LineString. StartPoint              | ✓                         | ✓                         | ✓      | ✓      |
| MultiLineString.<br>IsClosed        | ✓                         | ✓                         | ✓      | ✓      |
| Point. M                            | ✓                         | ✓                         | ✓      | ✓      |
| Point. X                            | ✓                         | ✓                         | ✓      | ✓      |
| Point. Y                            | ✓                         | ✓                         | ✓      | ✓      |
| Point. Z                            | ✓                         | ✓                         | ✓      | ✓      |
| Polygon. ExteriorRing               | ✓                         | ✓                         | ✓      | ✓      |
| Polygon.<br>GetInteriorRingN (int)  | ✓                         | ✓                         | ✓      | ✓      |

| NETTOPOLOGYSUITE             | SQL SERVER<br>(GÉOMÉTRIE) | SQL SERVER<br>(GEOGRAPHY) | SQlite | Npgsql |
|------------------------------|---------------------------|---------------------------|--------|--------|
| Polygon.<br>NumInteriorRings | ✓                         | ✓                         | ✓      | ✓      |

## Ressources supplémentaires

- [Données spatiales dans SQL Server](#)
- [Page d'accueil SpatiaLite](#)
- [Documentation spatiale npgsql](#)
- [Documentation PostGIS](#)

# Gestion des schémas de base de données

08/11/2019 • 2 minutes to read

EF Core propose deux méthodes pour que votre modèle EF Core et le schéma de base de données restent synchronisés. Pour choisir entre les deux, décidez si votre modèle EF Core ou le schéma de base de données est la source de vérité.

Si vous souhaitez que votre modèle EF Core soit la source de vérité, utilisez [Migrations](#). Quand vous apportez des modifications à votre modèle EF Core, cette approche applique progressivement les modifications de schéma correspondantes à votre base de données afin qu'elle reste compatible avec votre modèle EF Core.

Utilisez [l'ingénierie à rebours](#) si vous souhaitez que votre schéma de base de données soit la source de vérité. Cette approche vous permet de structurer un DbContext et les classes de type d'entité en reconstituant la logique de votre schéma de base de données dans un modèle EF Core.

## NOTE

Les [API de création et de suppression](#) peuvent également créer le schéma de base de données à partir de votre modèle EF Core. Toutefois, elles servent principalement pour des tâches de test, de prototypage et d'autres scénarios où la suppression de la base de données est acceptable.

# Migrations

08/01/2020 • 11 minutes to read

Un modèle de données change au cours du développement et perd sa synchronisation avec la base de données. Vous pouvez supprimer la base de données et laisser EF en créer une qui correspond au modèle, mais cette procédure entraîne la perte de données. La fonctionnalité de migration dans EF Core permet de mettre à jour de manière incrémentielle le schéma de la base de données pour qu'il reste synchronisé avec le modèle de données de l'application tout en conservant les données existantes dans la base de données.

Les migrations incluent des outils en ligne de commande et des API qui aident à effectuer les tâches suivantes :

- [Créer une migration](#). Générez du code qui peut mettre à jour la base de données pour la synchroniser avec un ensemble de modifications du modèle.
- [Mettre à jour la base de données](#). Appliquez des migrations en attente pour mettre à jour le schéma de base de données.
- [Personnaliser le code de migration](#). Le code généré doit parfois être modifié ou complété.
- [Supprimer une migration](#). Supprimez le code généré.
- [Rétablir une migration](#). Annulez les modifications apportées à la base de données.
- [Générer des scripts SQL](#). Vous aurez peut-être besoin d'un script pour mettre à jour une base de données de production ou pour résoudre des problèmes liés au code de migration.
- [Appliquer des migrations au moment de l'exécution](#). Quand les mises à jour au moment du design et l'exécution de scripts ne sont pas les meilleures options,appelez la méthode `Migrate()`.

## TIP

Si `DbContext` se trouve dans un assembly différent du projet de démarrage, vous pouvez spécifier explicitement les projets cible et de démarrage dans les [outils de la console du gestionnaire de package](#) ou dans les [outils CLI .NET Core](#).

## Installer les outils

Installez les [outils en ligne de commande](#) :

- Pour Visual Studio, nous vous recommandons les [outils de la Console du Gestionnaire de package](#).
- Pour d'autres environnements de développement, choisissez les [outils CLI .NET Core](#).

## Créer une migration

Une fois que vous avez [défini votre modèle initial](#), il est temps de créer la base de données. Pour ajouter une migration initiale, exécutez la commande suivante.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef migrations add InitialCreate
```

Trois fichiers sont ajoutés à votre projet sous le répertoire **Migrations** :

- **XXXXXXXXXXXXXX\_InitialCreate.cs** : fichier principal des migrations. Contient les opérations

nécessaires à l'application de la migration (dans `Up()`) et à sa restauration (dans `Down()`).

- **XXXXXXXXXXXXXX\_InitialCreate.Designer.cs** : fichier de métadonnées des migrations. Contient des informations utilisées par EF.
- **MyContextModelSnapshot.cs** : instantané de votre modèle actuel. Permet de déterminer ce qui a changé pendant l'ajout de la migration suivante.

L'horodatage dans le nom des fichiers permet de conserver ces derniers dans l'ordre chronologique et de voir ainsi la progression des modifications.

#### TIP

Vous êtes libre de déplacer les fichiers Migrations et de changer leur espace de noms. Les migrations sont créées en tant que sœurs de la dernière migration.

## Mettre à jour la base de données

Ensuite, appliquez la migration à la base de données pour créer le schéma.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef database update
```

## Personnaliser le code de migration

Une fois votre modèle EF Core modifié, le schéma de base de données risque de ne plus être synchronisé. Pour le mettre à jour, ajoutez une autre migration. Le nom de la migration peut être utilisé comme un message de validation dans un système de gestion de versions. Par exemple, vous pouvez choisir un nom tel que *AjouterÉvaluationsProduit* si la modification est une nouvelle classe d'entité pour les évaluations.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef migrations add AddProductReviews
```

Une fois que la migration a été structurée (que du code a été généré pour elle), vérifiez si le code est exact et ajoutez, supprimez ou modifiez toutes les opérations nécessaires pour pouvoir l'appliquer correctement.

Par exemple, une migration peut contenir les opérations suivantes :

```
migrationBuilder.DropColumn(  
    name: "FirstName",  
    table: "Customer");  
  
migrationBuilder.DropColumn(  
    name: "LastName",  
    table: "Customer");  
  
migrationBuilder.AddColumn<string>(  
    name: "Name",  
    table: "Customer",  
    nullable: true);
```

Bien que ces opérations rendent le schéma de base de données compatible, elles ne conservent pas les noms

de client existants. Pour l'améliorer, réécrivez la migration comme suit.

```
migrationBuilder.AddColumn<string>(
    name: "Name",
    table: "Customer",
    nullable: true);

migrationBuilder.Sql(
@"
    UPDATE Customer
    SET Name = FirstName + ' ' + LastName;
");

migrationBuilder.DropColumn(
    name: "FirstName",
    table: "Customer");

migrationBuilder.DropColumn(
    name: "LastName",
    table: "Customer");
```

#### TIP

Le processus de structuration de migration vous avertit quand une opération peut entraîner une perte de données (par exemple la suppression d'une colonne). Si cet avertissement s'affiche, veillez particulièrement à examiner si le code de migration est exact.

Appliquez la migration à la base de données à l'aide de la commande appropriée.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef database update
```

#### Migrations vides

Il est parfois utile d'ajouter une migration sans apporter de modification au modèle. Dans ce cas, l'ajout d'une nouvelle migration crée des fichiers de code avec des classes vides. Vous pouvez personnaliser cette migration pour effectuer des opérations qui ne sont pas directement liées au modèle EF Core. Voici quelques exemples de ce que vous pouvez gérer de cette façon :

- Recherche en texte intégral
- Fonctions
- Procédures stockées
- Déclencheurs
- Affichages

## Supprimer une migration

Parfois, vous ajoutez une migration et réalisez que vous devez apporter des modifications supplémentaires à votre modèle EF Core avant de l'appliquer. Pour supprimer la dernière migration, utilisez cette commande.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef migrations remove
```

Après avoir supprimé la migration, vous pouvez apporter les modifications supplémentaires au modèle et la rajouter.

## Rétablissement une migration

Si vous avez déjà appliqué une migration (ou plusieurs migrations) à la base de données, mais que vous devez la restaurer, vous pouvez utiliser la même commande que celle servant à appliquer des migrations, mais en spécifiant le nom de la migration à restaurer.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef database update LastGoodMigration
```

## Générer des scripts SQL

Quand vous déboguez vos migrations ou que vous les déployez sur une base de données de production, il est utile de générer un script SQL. Vous pouvez ensuite revoir le script et l'affiner en fonction des besoins d'une base de données de production. Vous pouvez également utiliser le script conjointement avec une technologie de déploiement. La commande de base est la suivante.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef migrations script
```

Il existe plusieurs options pour cette commande.

La migration **from** doit être la dernière migration appliquée à la base de données avant l'exécution du script. Si aucune migration n'a été appliquée, spécifiez `0` (il s'agit de la valeur par défaut).

La migration **to** est la dernière migration à appliquer à la base de données après l'exécution du script. Par défaut, il s'agit de la dernière migration dans votre projet.

Un script **idempotent** peut également être généré. Ce script n'applique les migrations que si elles n'ont pas déjà été appliquées à la base de données. Cela est utile si vous ne savez pas exactement ce que la dernière migration a appliquée à la base de données ou si vous effectuez un déploiement sur plusieurs bases de données pouvant chacune être liée à une migration différente.

## Appliquer des migrations au moment de l'exécution

Certaines applications sont susceptibles d'appliquer des migrations au moment de l'exécution (au démarrage ou à la première exécution). Ces opérations nécessitent l'utilisation de la méthode `Migrate()`.

Cette méthode s'appuie sur le service `IMigrator`, qui peut être utilisé pour des scénarios plus avancés. Utilisez `myDbContext.GetInfrastructure().GetService<IMigrator>()` pour y accéder.

```
myDbContext.Database.Migrate();
```

**WARNING**

- Cette approche ne s'adresse pas à tout un chacun. Bien qu'elle soit idéale pour les applications avec une base de données locale, la plupart des applications nécessitent une stratégie de déploiement plus robuste, telle que la génération de scripts SQL.
- Nappelez pas `EnsureCreated()` avant `Migrate()`. `EnsureCreated()` ignore Migrations pour créer le schéma, ce qui entraîne l'échec de `Migrate()`.

## Étapes suivantes

Pour plus d'informations, consultez [Informations de référence sur les outils Entity Framework Core - EF Core](#).

# Migrations dans les environnements d'équipe

07/11/2019 • 3 minutes to read

Lorsque vous travaillez avec des migrations dans des environnements d'équipe, portez une attention particulière au fichier d'instantané de modèle. Ce fichier peut vous indiquer si la migration de votre coéquipier fusionne correctement avec vous ou si vous devez résoudre un conflit en recréant la migration avant de la partager.

## fusion

Lorsque vous fusionnez des migrations à partir de vos coéquipiers, vous pouvez recevoir des conflits dans votre fichier d'instantané de modèle. Si les deux modifications ne sont pas liées, la fusion est triviale et les deux migrations peuvent coexister. Par exemple, vous pouvez obtenir un conflit de fusion dans la configuration du type d'entité client qui ressemble à ceci :

```
<<<<< Mine
b.Property<bool>("Deactivated");
=====
b.Property<int>("LoyaltyPoints");
>>>>> Theirs
```

Étant donné que ces deux propriétés doivent exister dans le modèle final, effectuez la fusion en ajoutant les deux propriétés. Dans de nombreux cas, votre système de contrôle de version peut automatiquement fusionner ces modifications pour vous.

```
b.Property<bool>("Deactivated");
b.Property<int>("LoyaltyPoints");
```

Dans ce cas, votre migration et la migration de votre coéquipier sont indépendantes les unes des autres. Dans la mesure où l'une d'entre elles peut être appliquée en premier, vous n'avez pas besoin d'apporter des modifications supplémentaires à votre migration avant de la partager avec votre équipe.

## Résolution des conflits

Parfois, vous rencontrez un vrai conflit lors de la fusion du modèle d'instantané de modèle. Par exemple, vous et votre coéquipier pouvez avoir renommé la même propriété.

```
<<<<< Mine
b.Property<string>("Username");
=====
b.Property<string>("Alias");
>>>>> Theirs
```

Si vous rencontrez ce type de conflit, résolvez-le en recréant votre migration. Procédez comme suit :

1. Abandonner la fusion et la restauration dans votre répertoire de travail avant la fusion
2. Supprimer votre migration (mais conserver les modifications de votre modèle)
3. Fusionner les modifications de votre coéquipier dans votre répertoire de travail
4. Rajouter votre migration

Après cela, les deux migrations peuvent être appliquées dans le bon ordre. Leur migration est appliquée en

premier, en renommant la colonne en *alias*. par la suite, votre migration le renomme nom *d'utilisateur*.

Votre migration peut être partagée en toute sécurité avec le reste de l'équipe.

# Opérations de migration personnalisées

25/10/2019 • 3 minutes to read

L'API MigrationBuilder vous permet d'effectuer de nombreux types d'opérations différents au cours d'une migration, mais elle est loin d'être exhaustive. Toutefois, l'API est également extensible, ce qui vous permet de définir vos propres opérations. Il existe deux façons d'étendre l'API : à l'aide de la méthode `Sql()`, ou en définissant des objets `MigrationOperation` personnalisés.

Pour illustrer cela, examinons l'implémentation d'une opération qui crée un utilisateur de base de données à l'aide de chaque approche. Dans nos migrations, nous souhaitons activer l'écriture du code suivant :

```
migrationBuilder.CreateUser("SQLUser1", "Password");
```

## Utilisation de MigrationBuilder. SQL ()

Le moyen le plus simple d'implémenter une opération personnalisée consiste à définir une méthode d'extension qui appelle `MigrationBuilder.Sql()`. Voici un exemple qui génère le Transact-SQL approprié.

```
static MigrationBuilder CreateUser(
    this MigrationBuilder migrationBuilder,
    string name,
    string password)
=> migrationBuilder.Sql($"CREATE USER {name} WITH PASSWORD '{password}');"
```

Si vos migrations doivent prendre en charge plusieurs fournisseurs de bases de données, vous pouvez utiliser la propriété `MigrationBuilder.ActiveProvider`. Voici un exemple qui prend en charge à la fois Microsoft SQL Server et PostgreSQL.

```
static MigrationBuilder CreateUser(
    this MigrationBuilder migrationBuilder,
    string name,
    string password)
{
    switch (migrationBuilder.ActiveProvider)
    {
        case "Npgsql.EntityFrameworkCore.PostgreSQL":
            return migrationBuilder
                .Sql($"CREATE USER {name} WITH PASSWORD '{password}');"

        case "Microsoft.EntityFrameworkCore.SqlServer":
            return migrationBuilder
                .Sql($"CREATE USER {name} WITH PASSWORD = '{password}');"
    }

    return migrationBuilder;
}
```

Cette approche fonctionne uniquement si vous connaissez tous les fournisseurs où votre opération personnalisée sera appliquée.

## Utilisation d'un MigrationOperation

Pour découpler l'opération personnalisée de SQL, vous pouvez définir votre propre `MigrationOperation` pour la représenter. L'opération est ensuite transmise au fournisseur afin de pouvoir déterminer le SQL approprié à générer.

```
class CreateUserOperation : MigrationOperation
{
    public string Name { get; set; }
    public string Password { get; set; }
}
```

Avec cette approche, la méthode d'extension doit simplement ajouter l'une de ces opérations à `MigrationBuilder.Operations`.

```
static MigrationBuilder CreateUser(
    this MigrationBuilder migrationBuilder,
    string name,
    string password)
{
    migrationBuilder.Operations.Add(
        new CreateUserOperation
        {
            Name = name,
            Password = password
        });

    return migrationBuilder;
}
```

Cette approche nécessite que chaque fournisseur sache comment générer SQL pour cette opération dans son service `IMigrationsSqlGenerator`. Voici un exemple qui remplace le générateur de SQL Server pour gérer la nouvelle opération.

```

class MyMigrationsSqlGenerator : SqlServerMigrationsSqlGenerator
{
    public MyMigrationsSqlGenerator(
        MigrationsSqlGeneratorDependencies dependencies,
        IMigrationsAnnotationProvider migrationsAnnotations)
        : base(dependencies, migrationsAnnotations)
    {
    }

    protected override void Generate(
        MigrationOperation operation,
        IModel model,
        MigrationCommandListBuilder builder)
    {
        if (operation is CreateUserOperation createUserOperation)
        {
            Generate(createUserOperation, builder);
        }
        else
        {
            base.Generate(operation, model, builder);
        }
    }

    private void Generate(
        CreateUserOperation operation,
        MigrationCommandListBuilder builder)
    {
        var sqlHelper = Dependencies.SqlGenerationHelper;
        var stringMapping = Dependencies.TypeMappingSource.FindMapping(typeof(string));

        builder
            .Append("CREATE USER ")
            .Append(sqlHelper.DelimitIdentifier(operation.Name))
            .Append(" WITH PASSWORD = ")
            .Append(stringMapping.GenerateSqlLiteral(operation.Password))
            .AppendLine(sqlHelper.StatementTerminator)
            .EndCommand();
    }
}

```

Remplacez le service SQL Generator des migrations par défaut par le service mis à jour.

```

protected override void OnConfiguring(DbContextOptionsBuilder options)
=> options
    .UseSqlServer(connectionString)
    .ReplaceService<IMigrationsSqlGenerator, MyMigrationsSqlGenerator>();

```

# Utilisation d'un projet de migrations distinct

05/12/2019 • 2 minutes to read

Vous souhaiterez peut-être stocker vos migrations dans un autre assembly que celui contenant votre `DbContext`.

Vous pouvez également utiliser cette stratégie pour gérer plusieurs ensembles de migrations, par exemple, un pour le développement et un autre pour les mises à niveau vers la version finale.

Pour...

1. Créez un nouveau projet de bibliothèque de classes.
2. Ajoutez une référence à votre assembly `DbContext`.
3. Déplacez les fichiers de migration et d'instantané de modèle dans la bibliothèque de classes.

## TIP

Si vous n'avez pas de migrations existantes, générez-en une dans le projet contenant le `DbContext`, puis déplacez-le. Cela est important car si l'assembly de migrations ne contient pas de migration existante, la commande `Add-migration` ne pourra pas trouver `DbContext`.

4. Configurez l'assembly des migrations :

```
options.UseSqlServer(  
    connectionString,  
    x => x.MigrationsAssembly("MyApp.Migrations"));
```

5. Ajoutez une référence à votre assembly de migration à partir de l'assembly de démarrage.

- Si cela provoque une dépendance circulaire, mettez à jour le chemin de sortie de la bibliothèque de classes :

```
<PropertyGroup>  
    <OutputPath>..\MyStartupProject\bin\$(Configuration)\</OutputPath>  
</PropertyGroup>
```

Si vous avez tout effectué correctement, vous devez être en mesure d'ajouter de nouvelles migrations au projet.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef migrations add NewMigration --project MyApp.Migrations
```

# Migrations avec plusieurs fournisseurs

09/01/2020 • 3 minutes to read

Les [outils de EF corent](#) uniquement les migrations de l'échafaudage pour le fournisseur actif. Toutefois, il peut arriver que vous souhaitiez utiliser plusieurs fournisseurs (par exemple Microsoft SQL Server et SQLite) avec votre DbContext. Il existe deux façons de gérer cela avec les migrations. Vous pouvez gérer deux ensembles de migrations, un pour chaque fournisseur, ou les fusionner dans un ensemble unique qui peut fonctionner sur les deux.

## Deux ensembles de migration

Dans la première approche, vous générez deux migrations pour chaque modification de modèle.

Pour ce faire, vous pouvez placer chaque ensemble de migration [dans un assembly distinct](#) et basculer manuellement le fournisseur actif (et l'assembly de migration) entre l'ajout des deux migrations.

Une autre approche qui facilite l'utilisation des outils consiste à créer un nouveau type qui dérive de votre DbContext et qui remplace le fournisseur actif. Ce type est utilisé au moment de la conception lors de l'ajout ou de l'application de migrations.

```
class MySqliteDbContext : MyDbContext
{
    protected override void OnConfiguring(DbContextOptionsBuilder options)
        => options.UseSqlite("Data Source=my.db");
}
```

### NOTE

Étant donné que chaque jeu de migration utilise ses propres types DbContext, cette approche ne nécessite pas l'utilisation d'un assembly de migrations distinct.

Lorsque vous ajoutez une nouvelle migration, spécifiez les types de contexte.

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet ef migrations add InitialCreate --context MyDbContext --output-dir Migrations/SqlServerMigrations
dotnet ef migrations add InitialCreate --context MySqliteDbContext --output-dir Migrations/SqliteMigrations
```

### TIP

Vous n'avez pas besoin de spécifier le répertoire de sortie pour les migrations suivantes, car elles sont créées en tant que frères au dernier.

## Un jeu de migration

Si vous n'aimez pas avoir deux ensembles de migrations, vous pouvez les combiner manuellement en un seul ensemble qui peut être appliqué aux deux fournisseurs.

Les annotations peuvent coexister car un fournisseur ignore les annotations qu'il ne comprend pas. Par exemple, une colonne de clé primaire qui fonctionne avec Microsoft SQL Server et SQLite peut se présenter comme suit.

```
Id = table.Column<int>(nullable: false)
    .Annotation("SqlServer:ValueGenerationStrategy", "SqlServerValueGenerationStrategy.IdentityColumn")
    .Annotation("Sqlite:Autoincrement", true),
```

Si les opérations ne peuvent être appliquées que sur un fournisseur (ou si elles sont différentes entre les fournisseurs), utilisez la propriété `ActiveProvider` pour indiquer quel fournisseur est actif.

```
if (migrationBuilder.ActiveProvider == "Microsoft.EntityFrameworkCore.SqlServer")
{
    migrationBuilder.CreateSequence(
        name: "EntityFrameworkHiLoSequence");
}
```

# Table d'historique des migrations personnalisées

07/11/2019 • 2 minutes to read

Par défaut, EF Core effectue le suivi des migrations qui ont été appliquées à la base de données en les enregistrant dans une table nommée `__EFMigrationsHistory`. Pour différentes raisons, vous souhaiterez peut-être personnaliser ce tableau pour mieux répondre à vos besoins.

## IMPORTANT

Si vous personnalisez la table d'historique des migrations *après avoir* appliqué des migrations, vous êtes responsable de la mise à jour de la table existante dans la base de données.

## Nom du schéma et de la table

Vous pouvez modifier le schéma et le nom de la table à l'aide de la méthode `MigrationsHistoryTable()` dans `OnConfiguring()` (ou `ConfigureServices()` sur ASP.NET Core). Voici un exemple d'utilisation du fournisseur SQL Server EF Core.

```
protected override void OnConfiguring(DbContextOptionsBuilder options)
    => options.UseSqlServer(
        connectionString,
        x => x.MigrationsHistoryTable("__MyMigrationsHistory", "mySchema"));
```

## Autres modifications

Pour configurer des aspects supplémentaires de la table, substituez et remplacez le service `IHistoryRepository` spécifique au fournisseur. Voici un exemple de modification du nom de la colonne `MigrationId` en `ID` sur SQL Server.

```
protected override void OnConfiguring(DbContextOptionsBuilder options)
    => options
        .UseSqlServer(connectionString)
        .ReplaceService<IHistoryRepository, MyHistoryRepository>();
```

## WARNING

`SqlServerHistoryRepository` est à l'intérieur d'un espace de noms interne et peut changer dans les versions ultérieures.

```
class MyHistoryRepository : SqlServerHistoryRepository
{
    public MyHistoryRepository(HistoryRepositoryDependencies dependencies)
        : base(dependencies)
    {
    }

    protected override void ConfigureTable(EntityTypeBuilder<HistoryRow> history)
    {
        base.ConfigureTable(history);

        history.Property(h => h.MigrationId).HasColumnName("Id");
    }
}
```

# Créer et supprimer des API

25/10/2019 • 2 minutes to read

Les méthodes `EnsureCreated` et `EnsureDeleted` fournissent une alternative légère aux [migrations](#) pour la gestion du schéma de base de données. Ces méthodes sont utiles dans les scénarios où les données sont temporaires et peuvent être supprimées lorsque le schéma est modifié. Par exemple pendant le prototypage, les tests ou les caches locaux.

Certains fournisseurs (surtout non relationnels) ne prennent pas en charge les migrations. Pour ces fournisseurs, `EnsureCreated` est souvent le moyen le plus simple d'initialiser le schéma de base de données.

## WARNING

`EnsureCreated` et les migrations ne fonctionnent pas correctement ensemble. Si vous utilisez des migrations, n'utilisez pas `EnsureCreated` pour initialiser le schéma.

La transition de `EnsureCreated` à des migrations n'est pas une expérience transparente. La façon la plus simple de procéder consiste à supprimer la base de données et à la recréer à l'aide des migrations. Si vous prévoyez d'utiliser des migrations à l'avenir, il est préférable de commencer par les migrations au lieu d'utiliser `EnsureCreated`.

## EnsureDeleted

La méthode `EnsureDeleted` supprime la base de données, le cas échéant. Si vous ne disposez pas des autorisations appropriées, une exception est levée.

```
// Drop the database if it exists
dbContext.Database.EnsureDeleted();
```

## EnsureCreated

`EnsureCreated` crée la base de données si elle n'existe pas et initialise le schéma de la base de données. Si des tables existent (y compris des tables pour une autre classe `DbContext`), le schéma ne sera pas initialisé.

```
// Create the database if it doesn't exist
dbContext.Database.EnsureCreated();
```

## TIP

Les versions asynchrones de ces méthodes sont également disponibles.

## Script SQL

Pour récupérer le SQL utilisé par `EnsureCreated`, vous pouvez utiliser la méthode `GenerateCreateScript`.

```
var sql = dbContext.Database.GenerateCreateScript();
```

## Plusieurs classes DbContext

EnsureCreated fonctionne uniquement si aucune table n'est présente dans la base de données. Si nécessaire, vous pouvez écrire votre propre vérification pour voir si le schéma doit être initialisé et utiliser le service IRelationalDatabaseCreator sous-jacent pour initialiser le schéma.

```
// TODO: Check whether the schema needs to be initialized

// Initialize the schema for this DbContext
var databaseCreator = dbContext.GetService<IRelationalDatabaseCreator>();
databaseCreator.CreateTables();
```

# Rétroconception

05/12/2019 • 12 minutes to read

L'ingénierie à rebours est le processus de génération de modèles automatique des classes de type d'entité et une classe DbContext basée sur un schéma de base de données. Il peut être effectué à l'aide de la commande `Scaffold-DbContext` des outils de la console du gestionnaire de package EF Core (PMC) ou de la commande `dotnet ef dbcontext scaffold` des outils de l'interface de ligne de commande (CLI) .NET.

## Installation de .

Avant l'ingénierie à rebours, vous devez installer les [Outils PMC](#) (Visual Studio uniquement) ou les [Outils CLI](#). Pour plus d'informations, consultez les liens.

Vous devez également installer un [fournisseur de base de données](#) approprié pour le schéma de base de données que vous souhaitez rétroconcevoir.

## Chaîne de connexion

Le premier argument de la commande est une chaîne de connexion à la base de données. Les outils utilisent cette chaîne de connexion pour lire le schéma de la base de données.

La façon dont vous utilisez les guillemets et les séquences d'échappement de la chaîne de connexion dépend du shell que vous utilisez pour exécuter la commande. Reportez-vous à la documentation de votre shell pour obtenir des informations spécifiques. Par exemple, PowerShell vous oblige à échapper le caractère `$`, mais pas `\`.

```
Scaffold-DbContext 'Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Chinook'  
Microsoft.EntityFrameworkCore.SqlServer
```

```
dotnet ef dbcontext scaffold "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Chinook"  
Microsoft.EntityFrameworkCore.SqlServer
```

## Configuration et secrets de l'utilisateur

Si vous avez un projet ASP.NET Core, vous pouvez utiliser la syntaxe `Name=<connection-string>` pour lire la chaîne de connexion à partir de la configuration.

Cela fonctionne bien avec l' [outil secret Manager](#) pour conserver le mot de passe de votre base de données distinct de votre code base.

```
dotnet user-secrets set ConnectionStrings.Chinook "Data Source=(localdb)\MSSQLLocalDB;Initial  
Catalog=Chinook"  
dotnet ef dbcontext scaffold Name=Chinook Microsoft.EntityFrameworkCore.SqlServer
```

## Nom du fournisseur

Le deuxième argument est le nom du fournisseur. Le nom du fournisseur est généralement identique au nom du package NuGet du fournisseur.

## Spécification de tables

Toutes les tables du schéma de base de données sont rétroconçues dans les types d'entités par défaut. Vous pouvez limiter les tables qui sont rétroconçues en spécifiant des schémas et des tables.

Le paramètre `-Schemas` dans PMC et l'option `--schema` de l'interface CLI peuvent être utilisés pour inclure chaque table dans un schéma.

`-Tables` (PMC) et `--table` (CLI) peuvent être utilisés pour inclure des tables spécifiques.

Pour inclure plusieurs tables dans PMC, utilisez un tableau.

```
Scaffold-DbContext ... -Tables Artist, Album
```

Pour inclure plusieurs tables dans l'interface CLI, spécifiez l'option plusieurs fois.

```
dotnet ef dbcontext scaffold ... --table Artist --table Album
```

## Préservation des noms

Les noms de table et de colonne sont fixes pour mieux correspondre aux conventions de nommage .NET pour les types et les propriétés par défaut. La spécification du commutateur `-UseDatabaseNames` dans PMC ou de l'option `--use-database-names` dans l'interface CLI désactivera ce comportement en conservant autant que possible les noms de bases de données originaux. Les identificateurs .NET non valides seront toujours fixes et les noms synthétisés, tels que les propriétés de navigation, seront toujours conformes aux conventions d'affectation de noms .NET.

## API Fluent ou annotations de données

Par défaut, les types d'entités sont configurés à l'aide de l'API Fluent. Spécifiez `-DataAnnotations` (PMC) ou `--data-annotations` (CLI) pour utiliser à la place des annotations de données lorsque cela est possible.

Par exemple, l'utilisation de l'API Fluent entraîne l'échafaudage suivant :

```
entity.Property(e => e.Title)
    .IsRequired()
    .HasMaxLength(160);
```

Lorsque vous utilisez des annotations de données, vous générez une structure :

```
[Required]
[StringLength(160)]
public string Title { get; set; }
```

## Nom de DbContext

Le nom de classe DbContext généré par génération de modèles automatique sera le nom de la base de données suffixée *par défaut*. Pour spécifier un autre, utilisez `-Context` dans PMC et `--context` dans l'interface CLI.

## Répertoires et espaces de noms

Les classes d'entité et une classe DbContext sont intégrées au répertoire racine du projet et utilisent l'espace de noms par défaut du projet. Vous pouvez spécifier le répertoire dans lequel les classes sont échafaudées à l'aide de `-OutputDir` (PMC) ou `--output-dir` (CLI). L'espace de noms sera l'espace de noms racine, ainsi que les noms de

tous les sous-répertoires sous le répertoire racine du projet.

Vous pouvez également utiliser `-ContextDir` (PMC) et `--context-dir` (CLI) pour générer une structure de la classe `DbContext` dans un répertoire distinct des classes de type d'entité.

```
Scaffold-DbContext ... -ContextDir Data -OutputDir Models
```

```
dotnet ef dbcontext scaffold ... --context-dir Data --output-dir Models
```

## Fonctionnement

L'ingénierie à rebours commence par lire le schéma de base de données. Il lit les informations sur les tables, les colonnes, les contraintes et les index.

Ensuite, il utilise les informations de schéma pour créer un modèle de EF Core. Les tables sont utilisées pour créer des types d'entité ; les colonnes sont utilisées pour créer des propriétés. et les clés étrangères sont utilisées pour créer des relations.

Enfin, le modèle est utilisé pour générer le code. Les classes de type d'entité, l'API Fluent et les annotations de données correspondantes sont échafaudées afin de recréer le même modèle à partir de votre application.

## Limitations

- Tout ce qui concerne un modèle peut être représenté à l'aide d'un schéma de base de données. Par exemple, les informations sur les **hiérarchies d'héritage**, les **types détenus** et le **fractionnement de table** ne sont pas présentes dans le schéma de la base de données. Pour cette raison, ces constructions ne feront jamais l'effet d'une rétroconception.
- En outre, **certains types de colonne** peuvent ne pas être pris en charge par le fournisseur EF Core. Ces colonnes ne sont pas incluses dans le modèle.
- Vous pouvez définir des **jetons d'accès concurrentiel**, dans un modèle EF Core pour empêcher deux utilisateurs de mettre à jour la même entité en même temps. Certaines bases de données ont un type spécial pour représenter ce type de colonne (par exemple, `rowversion` dans SQL Server), auquel cas nous pouvons rétroconcevoir ces informations. Toutefois, les autres jetons d'accès concurrentiel ne feront pas l'être par rétroconception.
- La **C# fonctionnalité de type référence Nullable 8** n'est actuellement pas prise en charge dans l'ingénierie à C# rebours : EF Core génère toujours du code qui suppose que la fonctionnalité est désactivée. Par exemple, les colonnes de texte `Nullable` seront échafaudées en tant que propriété de type `string`, et non pas `string?`, avec l'API Fluent ou les annotations de données utilisées pour configurer si une propriété est requise ou non. Vous pouvez modifier le code de génération de modèles automatique et C# les remplacer par des annotations de possibilité de valeur null. La prise en charge de la génération de modèles automatique pour les types de référence `Nullable` est suivie par le problème [#15520](#).

## Personnalisation du modèle

Le code généré par EF Core est votre code. N'hésitez pas à la modifier. Elle sera régénérée uniquement si vous retrouvez à nouveau le même modèle. Le code de génération de modèles automatique représente *un* modèle qui peut être utilisé pour accéder à la base de données, mais ce n'est certainement pas le *seul* modèle qui peut être utilisé.

Personnalisez les classes de type d'entité et la classe `DbContext` en fonction de vos besoins. Par exemple, vous pouvez choisir de renommer des types et des propriétés, d'introduire des hiérarchies d'héritage ou de fractionner une table en plusieurs entités. Vous pouvez également supprimer des index non uniques, des séquences

inutilisées et des propriétés de navigation, des propriétés scalaires facultatives et des noms de contrainte à partir du modèle.

Vous pouvez également ajouter des constructeurs, des méthodes, des propriétés, etc. supplémentaires. utilisation d'une autre classe partielle dans un fichier séparé. Cette approche fonctionne même lorsque vous envisagez de réactiver le modèle.

## Mise à jour du modèle

Après avoir apporté des modifications à la base de données, vous devrez peut-être mettre à jour votre modèle de EF Core pour refléter ces modifications. Si les modifications de la base de données sont simples, il peut être plus facile d'apporter manuellement les modifications à votre modèle de EF Core. Par exemple, le fait de renommer une table ou une colonne, de supprimer une colonne ou de mettre à jour le type d'une colonne est une modification triviale dans le code.

Toutefois, les modifications les plus importantes ne sont pas aussi faciles à effectuer manuellement. Un flux de travail courant consiste à reconstituer à nouveau le modèle de la base de données à l'aide d'`-Force` (PMC) ou d'`--force` (CLI) pour remplacer le modèle existant par un modèle mis à jour.

Une autre fonctionnalité couramment demandée est la possibilité de mettre à jour le modèle à partir de la base de données tout en préservant la personnalisation, comme les renommages, les hiérarchies de types, etc. Utilisez [#831](#) de problème pour suivre la progression de cette fonctionnalité.

### WARNING

Si vous rérétroconcevez le modèle à partir de la base de données, toutes les modifications que vous avez apportées aux fichiers seront perdues.

# Interrogation des données

11/10/2019 • 2 minutes to read • [Edit Online](#)

Entity Framework Core utilise LINQ (Language Integrated Query) pour interroger les données de la base de données. LINQ vous permet d'utiliser C# (ou le langage .NET de votre choix) pour écrire des requêtes fortement typées. Il utilise vos classes de contexte et d'entité dérivées pour référencer les objets de base de données. EF Core passe une représentation de la requête LINQ au fournisseur de bases de données. Les fournisseurs de bases de données la traduisent à leur tour en langage de requête spécifique aux bases de données (par exemple, SQL pour une base de données relationnelle).

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

Les extraits de code montrent quelques exemples illustrant comment accomplir des tâches courantes avec Entity Framework Core.

## Chargement de toutes les données

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.ToList();
}
```

## Chargement d'une seule entité

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);
}
```

## Filtrage

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Where(b => b.Url.Contains("dotnet"))
        .ToList();
}
```

## Pour aller plus loin

- Découvrir les [expressions de requête LINQ](#)
- Pour plus d'informations sur le traitement d'une requête dans EF Core, consultez [Fonctionnement d'une requête](#).

# Comparaison entre client et serveur

23/11/2019 • 10 minutes to read • [Edit Online](#)

En règle générale, Entity Framework Core tente d'évaluer autant que possible une requête sur le serveur. EF Core convertit des parties de la requête en paramètres, qu'elle peut évaluer côté client. Le reste de la requête (avec les paramètres générés) est donné au fournisseur de base de données pour déterminer la requête de base de données équivalente à évaluer sur le serveur. EF Core prend en charge l'évaluation partielle du client dans la projection de niveau supérieur (essentiellement, le dernier appel à `Select()`). Si la projection de niveau supérieur dans la requête ne peut pas être traduite sur le serveur, EF Core extrait toutes les données requises du serveur et évalue les autres parties de la requête sur le client. Si EF Core détecte une expression, à un emplacement autre que la projection de niveau supérieur, qui ne peut pas être traduite sur le serveur, elle lève une exception Runtime. Découvrez [Comment fonctionne la requête](#) pour comprendre comment EF Core détermine ce qui ne peut pas être traduit sur le serveur.

## NOTE

Avant la version 3,0, Entity Framework Core l'évaluation du client prise en charge n'importe où dans la requête. Pour plus d'informations, consultez la [section versions précédentes](#).

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Évaluation du client dans la projection de niveau supérieur

Dans l'exemple suivant, une méthode d'assistance est utilisée pour normaliser les URL des blogs, qui sont retournées à partir d'une base de données SQL Server. Étant donné que le fournisseur de SQL Server n'a aucune idée de la façon dont cette méthode est implémentée, il n'est pas possible de la traduire en SQL. Tous les autres aspects de la requête sont évalués dans la base de données, mais le passage de la `URL` renvoyée via cette méthode s'effectue sur le client.

```
var blogs = context.Blogs
    .OrderByDescending(blog => blog.Rating)
    .Select(blog => new
    {
        Id = blog.BlogId,
        Url = StandardizeUrl(blog.Url)
    })
    .ToList();
```

```

public static string StandardizeUrl(string url)
{
    url = url.ToLower();

    if (!url.StartsWith("http://"))
    {
        url = string.Concat("http://", url);
    }

    return url;
}

```

## Évaluation du client non prise en charge

Si l'évaluation du client est utile, elle peut entraîner des performances médiocres. Considérez la requête suivante, dans laquelle la méthode d'assistance est désormais utilisée dans un filtre Where. Étant donné que le filtre ne peut pas être appliqué dans la base de données, toutes les données doivent être extraites en mémoire pour appliquer le filtre sur le client. En fonction du filtre et de la quantité de données sur le serveur, l'évaluation du client peut entraîner des performances médiocres. Ainsi Entity Framework Core bloque cette évaluation du client et lève une exception d'exécution.

```

var blogs = context.Blogs
    .Where(blog => StandardizeUrl(blog.Url).Contains("dotnet"))
    .ToList();

```

## Évaluation explicite du client

Vous devrez peut-être forcer l'évaluation du client explicitement dans certains cas, comme suit :

- La quantité de données est faible, de sorte que l'évaluation sur le client n'entraîne pas une baisse considérable des performances.
- L'opérateur LINQ utilisé n'a pas de traduction côté serveur.

Dans ce cas, vous pouvez choisir explicitement l'évaluation du client en appelant des méthodes comme `AsEnumerable` ou `ToList` (`AsAsyncEnumerable` ou `ToToListAsync` pour `Async`). En utilisant `AsEnumerable` vous diffuseriez les résultats, mais l'utilisation de `ToList` entraînerait une mise en mémoire tampon en créant une liste, ce qui prend également davantage de mémoire. Toutefois, si vous énumérez plusieurs fois, le stockage des résultats dans une liste est plus utile, car il n'existe qu'une seule requête dans la base de données. En fonction de l'utilisation particulière, vous devez déterminer la méthode qui est plus utile pour le cas.

```

var blogs = context.Blogs
    .AsEnumerable()
    .Where(blog => StandardizeUrl(blog.Url).Contains("dotnet"))
    .ToList();

```

## Fuite de mémoire potentielle dans l'évaluation du client

Étant donné que la traduction et la compilation des requêtes sont coûteuses, EF Core met en cache le plan de requête compilé. Le délégué mis en cache peut utiliser le code client lors de l'évaluation du client de la projection de niveau supérieur. EF Core génère des paramètres pour les parties évaluées par le client de l'arborescence et réutilise le plan de requête en remplaçant les valeurs des paramètres. Toutefois, certaines constantes de l'arborescence de l'expression ne peuvent pas être converties en paramètres. Si le délégué mis en cache contient des constantes, ces objets ne peuvent pas être récupérés par le garbage collector, car ils sont toujours référencés. Si un tel objet contient un `DbContext` ou d'autres services qu'il contient, cela peut entraîner une augmentation de

l'utilisation de la mémoire de l'application au fil du temps. Ce comportement est généralement le signe d'une fuite de mémoire. EF Core lève une exception chaque fois qu'il s'agit d'une constante d'un type qui ne peut pas être mappé à l'aide du fournisseur de base de données actuel. Les causes courantes et leurs solutions sont les suivantes :

- **Utilisation d'une méthode d'instance:** lors de l'utilisation de méthodes d'instance dans une projection cliente, l'arborescence de l'expression contient une constante de l'instance. Si votre méthode n'utilise pas de données de l'instance, envisagez de rendre la méthode statique. Si vous avez besoin de données d'instance dans le corps de la méthode, transmettez les données spécifiques en tant qu'argument à la méthode.
- **Passage d'arguments de constante à la méthode:** ce cas se produit généralement à l'aide de `this` dans un argument de la méthode cliente. Envisagez de fractionner l'argument dans en plusieurs arguments scalaires, qui peuvent être mappés par le fournisseur de base de données.
- **Autres constantes:** si une constante est parvenue dans un autre cas, vous pouvez évaluer si la constante est nécessaire dans le traitement. S'il est nécessaire d'avoir la constante, ou si vous ne pouvez pas utiliser une solution des cas ci-dessus, créez une variable locale pour stocker la valeur et utilisez la variable locale dans la requête. EF Core convertira la variable locale en paramètre.

## Versions antérieures

La section suivante s'applique aux versions de EF Core antérieures à 3,0.

Les anciennes versions de EF Core prises en charge pour l'évaluation du client dans n'importe quelle partie de la requête, et pas seulement la projection de niveau supérieur. C'est pourquoi les requêtes similaires à une publication sous la section d'[évaluation du client non prise en charge](#) fonctionnaient correctement. Dans la mesure où ce comportement peut provoquer des problèmes de performances invisibles, EF Core journalisé un avertissement d'évaluation du client. Pour plus d'informations sur l'affichage de la sortie de journalisation, consultez [journalisation](#).

Si vous le souhaitez, vous pouvez EF Core modifier le comportement par défaut pour lever une exception ou ne rien faire lors de l'évaluation du client (à l'exception de dans la projection). Le comportement de levée d'exception le rendrait similaire au comportement dans 3,0. Pour modifier le comportement, vous devez configurer des avertissements lors de la configuration des options de votre contexte (généralement dans

`DbContext.OnConfiguring` ou dans `Startup.cs` si vous utilisez ASP.NET Core.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder
        .UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=EFQuerying;Trusted_Connection=True;")
        .ConfigureWarnings(warnings => warnings.Throw(RelationalEventId.QueryClientEvaluationWarning));
}
```

# Suivi et requêtes sans suivi

17/10/2019 • 8 minutes to read • [Edit Online](#)

Suivi des contrôles de comportement si Entity Framework Core conservera des informations sur une instance d'entité dans son dispositif de suivi des modifications. Si une entité est suivie, toutes les modifications détectées dans l'entité sont rendues persistantes dans la base de données pendant `SaveChanges()`. EF Core corrigera également les propriétés de navigation entre les entités dans un résultat de requête de suivi et les entités qui se trouvent dans le dispositif de suivi des modifications.

## NOTE

Les types d'entité sans clé ne sont jamais suivis. Chaque fois que cet article mentionne des types d'entités, il fait référence aux types d'entité qui ont une clé définie.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Requêtes avec suivi

Par défaut, les requêtes qui retournent des types d'entités ont le suivi activé. Cela signifie que vous pouvez apporter des modifications à ces instances d'entité et rendre ces modifications persistantes par `SaveChanges()`. Dans l'exemple suivant, la modification de l'évaluation des blogs sera détectée et rendue persistante dans la base de données pendant `SaveChanges()`.

```
var blog = context.Blogs.SingleOrDefault(b => b.BlogId == 1);
blog.Rating = 5;
context.SaveChanges();
```

## Pas de suivi des requêtes

Les requêtes sans suivi sont utiles lorsque les résultats sont utilisés dans un scénario en lecture seule. Elles sont plus rapides à exécuter, car il n'est pas nécessaire de configurer les informations de suivi des modifications. Si vous n'avez pas besoin de mettre à jour les entités récupérées de la base de données, une requête de non-suivi doit être utilisée. Vous pouvez permuter une requête individuelle pour qu'elle soit sans suivi.

```
var blogs = context.Blogs
    .AsNoTracking()
    .ToList();
```

Vous pouvez également modifier le comportement de suivi par défaut au niveau de l'instance du contexte :

```
context.ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;

var blogs = context.Blogs.ToList();
```

## Résolution de l'identité

Dans la mesure où une requête de suivi utilise le suivi des modifications, EF Core effectue la résolution d'identité dans une requête de suivi. Lors de la matérialisation d'une entité, EF Core retourne la même instance d'entité à partir du dispositif de suivi des modifications si elle fait déjà l'objet d'un suivi. Si le résultat contient plusieurs fois la même entité, vous obtenez la même instance pour chaque occurrence. Les requêtes de non-suivi n'utilisent pas le dispositif de suivi des modifications et n'effectuent pas de résolution d'identité. Ainsi, vous obtenez une nouvelle instance de l'entité même lorsque la même entité est contenue plusieurs fois dans le résultat. Ce comportement est différent dans les versions antérieures à EF Core 3,0, consultez [versions précédentes](#).

## Suivi et projections personnalisées

Même si le type de résultat de la requête n'est pas un type d'entité, EF Core effectue toujours le suivi des types d'entité contenus dans le résultat par défaut. Dans la requête suivante, qui retourne un type anonyme, les instances de `Blog` dans le jeu de résultats sont suivies.

```
var blog = context.Blogs
    .Select(b =>
    new
    {
        Blog = b,
        PostCount = b.Posts.Count()
    });
}
```

Si le jeu de résultats contient des types d'entités provenant de la composition LINQ, EF Core les suit.

```
var blog = context.Blogs
    .Select(b =>
    new
    {
        Blog = b,
        Post = b.Posts.OrderBy(p => p.Rating).LastOrDefault()
    });
}
```

Si le jeu de résultats ne contient aucun type d'entité, aucun suivi n'est effectué. Dans la requête suivante, nous retournons un type anonyme avec certaines des valeurs de l'entité (mais aucune instance du type d'entité réel). Aucune entité suivie n'est en provenance de la requête.

```
var blog = context.Blogs
    .Select(b =>
    new
    {
        Id = b.BlogId,
        Url = b.Url
    });
}
```

EF Core prend en charge l'évaluation du client dans la projection de niveau supérieur. Si EF Core matérialise une instance d'entité pour l'évaluation du client, elle est suivie. Ici, puisque nous passons les entités `blog` à la méthode client `standardizeURL`, EF Core effectuera le suivi des instances de blog.

```

var blogs = context.Blogs
    .OrderByDescending(blog => blog.Rating)
    .Select(blog => new
    {
        Id = blog.BlogId,
        Url = StandardizeUrl(blog)
    })
    .ToList();

```

```

public static string StandardizeUrl(Blog blog)
{
    var url = blog.Url.ToLower();

    if (!url.StartsWith("http://"))
    {
        url = string.Concat("http://", url);
    }

    return url;
}

```

EF Core n'effectue pas le suivi des instances d'entité keymoins contenues dans le résultat. Mais EF Core effectue le suivi de toutes les autres instances de types d'entité avec la clé conformément aux règles ci-dessus.

Certaines des règles ci-dessus ont fonctionné différemment avant EF Core 3.0. Pour plus d'informations, consultez [versions précédentes](#).

## Versions antérieures

Avant la version 3.0, les EF Core présentaient certaines différences quant à la façon dont le suivi a été effectué. Les différences notables sont les suivantes :

- Comme expliqué dans la page [évaluation des clients vs Server](#), EF Core évaluation du client prise en charge dans n'importe quelle partie de la requête avant la version 3.0. L'évaluation du client a provoqué la matérialisation des entités, ce qui n'a pas fait partie du résultat. Par conséquent, EF Core analysé le résultat pour détecter les éléments à suivre. Cette conception présentait certaines différences, comme suit :
  - L'évaluation du client dans la projection, qui provoquait la matérialisation mais n'a pas retourné l'instance d'entité matérialisée n'a pas été suivie. L'exemple suivant n'a pas suivi les entités `blog`.

```

var blogs = context.Blogs
    .OrderByDescending(blog => blog.Rating)
    .Select(blog => new
    {
        Id = blog.BlogId,
        Url = StandardizeUrl(blog)
    })
    .ToList();

```

- Dans certains cas, EF Core n'avez pas suivi les objets en provenance de la composition LINQ. L'exemple suivant n'a pas suivi `Post`.

```
var blog = context.Blogs
    .Select(b =>
        new
        {
            Blog = b,
            Post = b.Posts.OrderBy(p => p.Rating).LastOrDefault()
        });

```

- Chaque fois que les résultats de la requête contiennent des types d'entité sans clé, l'ensemble de la requête a été rendu non suivi. Cela signifie que les types d'entité avec des clés, qui sont dans le résultat n'ont pas été suivis.
- EF Core a fait la résolution d'identité dans une requête de non-suivi. Elle utilisait des références faibles pour effectuer le suivi des entités qui avaient déjà été retournées. Par conséquent, si un jeu de résultats contenait la même entité plusieurs fois, vous obtiendriez la même instance pour chaque occurrence. Toutefois, si un résultat précédent avec la même identité est hors de portée et qu'il a été récupéré par le garbage collector, EF Core retourné une nouvelle instance.

# Opérateurs de requête complexes

23/11/2019 • 13 minutes to read • [Edit Online](#)

LINQ (Language Integrated Query) contient de nombreux opérateurs complexes, qui combinent plusieurs sources de données ou effectuent un traitement complexe. Les opérateurs LINQ n'ont pas tous des traductions appropriées côté serveur. Parfois, une requête dans un formulaire se traduit par le serveur, mais s'il est écrit dans un autre formulaire, même si le résultat est le même. Cette page décrit certains des opérateurs complexes et leurs variations prises en charge. Dans les versions ultérieures, nous pouvons reconnaître plus de modèles et ajouter les traductions correspondantes. Il est également important de garder à l'esprit que la prise en charge de la traduction varie d'un fournisseur à l'autre. Une requête particulière, qui est traduite dans SqlServer, peut ne pas fonctionner pour les bases de données SQLite.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Join

L'opérateur de jointure LINQ vous permet de connecter deux sources de données en fonction du sélecteur de clé pour chaque source, en générant un tuple de valeurs lorsque la clé correspond. Il se traduit naturellement par `INNER JOIN` sur les bases de données relationnelles. Alors que la jointure LINQ a des sélecteurs de clé externe et interne, la base de données requiert une seule condition de jointure. Ainsi EF Core génère une condition de jointure en comparant le sélecteur de clé externe au sélecteur de clé interne pour l'égalité. En outre, si les sélecteurs de clé sont des types anonymes, EF Core génère une condition de jointure pour comparer l'égalité des composants.

```
var query = from photo in context.Set<PersonPhoto>()
            join person in context.Set<Person>()
                on photo.PersonPhotoId equals person.PhotoId
            select new { person, photo };
```

```
SELECT [p].[PersonId], [p].[Name], [p].[PhotoId], [p0].[PersonPhotoId], [p0].[Caption], [p0].[Photo]
FROM [PersonPhoto] AS [p0]
INNER JOIN [Person] AS [p] ON [p0].[PersonPhotoId] = [p].[PhotoId]
```

## GroupJoin

L'opérateur LINQ GroupJoin vous permet de connecter deux sources de données similaires à Join, mais elle crée un groupe de valeurs internes pour les éléments externes correspondants. L'exécution d'une requête comme l'exemple suivant génère un résultat de `Blog & IEnumerable<Post>`. Étant donné que les bases de données (notamment les bases de données relationnelles) n'ont pas de moyen de représenter une collection d'objets côté client, GroupJoin ne se traduit pas par le serveur dans de nombreux cas. Elle nécessite que vous obteniez toutes les données du serveur pour effectuer la GroupJoin sans sélecteur spécial (première requête ci-dessous). Toutefois, si le sélecteur limite les données sélectionnées, l'extraction de toutes les données du serveur peut entraîner des problèmes de performances (deuxième requête ci-dessous). C'est la raison pour laquelle EF Core ne traduit pas GroupJoin.

```
var query = from b in context.Set<Blog>()
            join p in context.Set<Post>()
            on b.BlogId equals p.PostId into grouping
            select new { b, grouping };
```

```
var query = from b in context.Set<Blog>()
            join p in context.Set<Post>()
            on b.BlogId equals p.PostId into grouping
            select new { b, Posts = grouping.Where(p => p.Content.Contains("EF")).ToList() };
```

## SelectMany

L'opérateur SelectMany de LINQ vous permet d'énumérer un sélecteur de collection pour chaque élément externe et de générer des tuples de valeurs à partir de chaque source de données. En d'autres termes, il s'agit d'une jointure, mais sans aucune condition, chaque élément externe est connecté avec un élément de la source de la collection. Selon la façon dont le sélecteur de collection est lié à la source de données externe, SelectMany peut traduire en différentes requêtes différentes côté serveur.

### Le sélecteur de collection ne fait pas référence à l'extérieur

Lorsque le sélecteur de collection ne référence rien de la source externe, le résultat est un produit cartésien des deux sources de données. Il se traduit par `CROSS JOIN` dans les bases de données relationnelles.

```
var query = from b in context.Set<Blog>()
            from p in context.Set<Post>()
            select new { b, p };
```

```
SELECT [b].[BlogId], [b].[OwnerId], [b].[Rating], [b].[Url], [p].[PostId], [p].[AuthorId], [p].[BlogId], [p].[Content], [p].[Rating], [p].[Title]
FROM [Blogs] AS [b]
CROSS JOIN [Posts] AS [p]
```

### Le sélecteur de collection fait référence à Outer dans une clause WHERE

Lorsque le sélecteur de collection a une clause WHERE, qui fait référence à l'élément externe, EF Core le convertit en une jointure de base de données et utilise le prédictat comme condition de jointure. Ce cas se produit généralement lors de l'utilisation de la navigation de collection sur l'élément externe comme sélecteur de collection. Si la collection est vide pour un élément externe, aucun résultat ne sera généré pour cet élément externe. Toutefois, si `DefaultIfEmpty` est appliqué au sélecteur de collection, l'élément externe est connecté avec une valeur par défaut de l'élément interne. En raison de cette distinction, ce type de requêtes se traduit par `INNER JOIN` en l'absence de `DefaultIfEmpty` et de `LEFT JOIN` quand `DefaultIfEmpty` est appliqué.

```
var query = from b in context.Set<Blog>()
            from p in context.Set<Post>().Where(p => b.BlogId == p.BlogId)
            select new { b, p };

var query2 = from b in context.Set<Blog>()
            from p in context.Set<Post>().Where(p => b.BlogId == p.BlogId).DefaultIfEmpty()
            select new { b, p };
```

```

SELECT [b].[BlogId], [b].[OwnerId], [b].[Rating], [b].[Url], [p].[PostId], [p].[AuthorId], [p].[BlogId], [p].[Content],
       [p].[Rating], [p].[Title]
  FROM [Blogs] AS [b]
INNER JOIN [Posts] AS [p] ON [b].[BlogId] = [p].[BlogId]

SELECT [b].[BlogId], [b].[OwnerId], [b].[Rating], [b].[Url], [p].[PostId], [p].[AuthorId], [p].[BlogId], [p].[Content],
       [p].[Rating], [p].[Title]
  FROM [Blogs] AS [b]
LEFT JOIN [Posts] AS [p] ON [b].[BlogId] = [p].[BlogId]

```

### Le sélecteur de collection fait référence à l'extérieur dans un cas autre que Where

Lorsque le sélecteur de collection fait référence à l'élément externe, qui n'est pas dans une clause WHERE (comme le cas ci-dessus), il ne se convertit pas en jointure de base de données. C'est pourquoi nous devons évaluer le sélecteur de collection pour chaque élément externe. Il se traduit par `APPLY` opérations dans de nombreuses bases de données relationnelles. Si la collection est vide pour un élément externe, aucun résultat ne sera généré pour cet élément externe. Toutefois, si `DefaultIfEmpty` est appliqué au sélecteur de collection, l'élément externe est connecté avec une valeur par défaut de l'élément interne. En raison de cette distinction, ce type de requêtes se traduit par `CROSS APPLY` en l'absence de `DefaultIfEmpty` et de `OUTER APPLY` quand `DefaultIfEmpty` est appliqué. Certaines bases de données telles que SQLite ne prennent pas en charge les opérateurs `APPLY`, de sorte que ce type de requête peut ne pas être traduit.

```

var query = from b in context.Set<Blog>()
            from p in context.Set<Post>().Select(p => b.Url + "=" + p.Title)
            select new { b, p };

var query2 = from b in context.Set<Blog>()
            from p in context.Set<Post>().Select(p => b.Url + "=" + p.Title).DefaultIfEmpty()
            select new { b, p };

```

```

SELECT [b].[BlogId], [b].[OwnerId], [b].[Rating], [b].[Url], ([b].[Url] + N'=>') + [p].[Title] AS [p]
  FROM [Blogs] AS [b]
CROSS APPLY [Posts] AS [p]

SELECT [b].[BlogId], [b].[OwnerId], [b].[Rating], [b].[Url], ([b].[Url] + N'=>') + [p].[Title] AS [p]
  FROM [Blogs] AS [b]
OUTER APPLY [Posts] AS [p]

```

## GroupBy

Les opérateurs LINQ GroupBy créent un résultat de type `IGrouping<TKey, TElement>` où `TKey` et `TElement` peut être n'importe quel type arbitraire. En outre, `IGrouping` implémente `IEnumerable<TElement>`, ce qui signifie que vous pouvez le composer à l'aide d'un opérateur LINQ après le regroupement. Dans la mesure où aucune structure de base de données ne peut représenter un `IGrouping`, les opérateurs GroupBy n'ont pas de traduction dans la plupart des cas. Quand un opérateur d'agrégation est appliqué à chaque groupe, qui retourne une valeur scalaire, il peut être traduit en SQL `GROUP BY` dans des bases de données relationnelles. Le `GROUP BY` SQL est également restrictif. Elle vous oblige à regrouper uniquement par valeurs scalaires. La projection peut uniquement contenir des colonnes clés de regroupement ou n'importe quel agrégat appliqué à une colonne. EF Core identifie ce modèle et le convertit en serveur, comme dans l'exemple suivant :

```
var query = from p in context.Set<Post>()
            group p by p.AuthorId into g
            select new
            {
                g.Key,
                Count = g.Count()
            };

```

```
SELECT [p].[AuthorId] AS [Key], COUNT(*) AS [Count]
FROM [Posts] AS [p]
GROUP BY [p].[AuthorId]
```

EF Core convertit également les requêtes où un opérateur d'agrégation sur le regroupement s'affiche dans un opérateur LINQ WHERE ou OrderBy (ou autre classement). Elle utilise `HAVING` clause dans SQL pour la clause WHERE. La partie de la requête avant l'application de l'opérateur GroupBy peut être une requête complexe, à condition qu'elle puisse être traduite en serveur. En outre, une fois que vous appliquez des opérateurs d'agrégation sur une requête de regroupement pour supprimer des regroupements de la source résultante, vous pouvez composer en plus de celle-ci comme n'importe quelle autre requête.

```
var query = from p in context.Set<Post>()
            group p by p.AuthorId into g
            where g.Count() > 0
            orderby g.Key
            select new
            {
                g.Key,
                Count = g.Count()
            };

```

```
SELECT [p].[AuthorId] AS [Key], COUNT(*) AS [Count]
FROM [Posts] AS [p]
GROUP BY [p].[AuthorId]
HAVING COUNT(*) > 0
ORDER BY [p].[AuthorId]
```

Les opérateurs d'agrégation EF Core prend en charge sont les suivants :

- Moyenne
- Nombre
- LongCount
- Max
- Min
- Sum

## Jointure gauche

Alors que la jointure de gauche n'est pas un opérateur LINQ, les bases de données relationnelles ont le concept d'une jointure de gauche qui est fréquemment utilisée dans les requêtes. Un modèle particulier dans les requêtes LINQ donne le même résultat qu'une `LEFT JOIN` sur le serveur. EF Core identifie de tels modèles et génère l'équivalent `LEFT JOIN` côté serveur. Le modèle implique la création d'une GroupJoin entre les sources de données, puis l'aplatissement du regroupement à l'aide de l'opérateur SelectMany avec DefaultIfEmpty sur la source de regroupement pour correspondre à NULL lorsque le interne n'a pas d'élément associé. L'exemple suivant montre à quoi ressemble ce modèle et ce qu'il génère.

```
var query = from b in context.Set<Blog>()
    join p in context.Set<Post>()
        on b.BlogId equals p.BlogId into grouping
    from p in grouping.DefaultIfEmpty()
    select new { b, p };
```

```
SELECT [b].[BlogId], [b].[OwnerId], [b].[Rating], [b].[Url], [p].[PostId], [p].[AuthorId], [p].[BlogId], [p].[Content], [p].[Rating], [p].[Title]
FROM [Blogs] AS [b]
LEFT JOIN [Posts] AS [p] ON [b].[BlogId] = [p].[BlogId]
```

Le modèle ci-dessus crée une structure complexe dans l’arborescence de l’expression. Pour cette raison, EF Core vous oblige à aplatiser les résultats de regroupement de l’opérateur GroupJoin dans une étape qui suit immédiatement l’opérateur. Même si GroupJoin-DefaultIfEmpty-SelectMany est utilisé, mais dans un modèle différent, il est possible qu’il ne soit pas identifié comme une jointure de gauche.

# Chargement des données associées

07/11/2019 • 13 minutes to read • [Edit Online](#)

Entity Framework Core vous permet d'utiliser les propriétés de navigation dans votre modèle pour charger des entités associées. Il existe trois modèles O/RM communs utilisés pour charger les données associées.

- Le **Chargement hâtif** signifie que les données associées sont chargées à partir de la base de données dans le cadre de la requête initiale.
- Le **Chargement explicite** signifie que les données associées sont explicitement chargées à partir de la base de données à un moment ultérieur.
- Le **Chargement différé** signifie que les données associées sont chargées de façon transparente à partir de la base de données lors de l'accès à la propriété de navigation.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Chargement hâtif

Vous pouvez utiliser la méthode `Include` pour spécifier les données associées à inclure dans les résultats de la requête. Dans l'exemple suivant, les blogs retournés dans les résultats auront leurs propriétés `Posts` remplies avec les billets associés.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .ToList();
}
```

## TIP

Entity Framework Core corrige automatiquement les propriétés de navigation vers d'autres entités qui étaient précédemment chargées dans l'instance de contexte. Ainsi, même si vous n'incluez pas explicitement toutes les données pour une propriété de navigation, la propriété peut toujours être renseignée si toutes ou une partie des entités associées ont été précédemment chargées.

Vous pouvez inclure des données associées provenant de plusieurs relations dans une seule requête.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
        .Include(blog => blog.Owner)
        .ToList();
}
```

## Inclusion de plusieurs niveaux

Vous pouvez descendre dans la hiérarchie des relations pour inclure plusieurs niveaux de données associées à

l'aide de la méthode `ThenInclude`. L'exemple suivant charge tous les blogs, leurs messages associés et l'auteur de chaque publication.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
        .ToList();
}
```

Vous pouvez enchaîner plusieurs appels à `ThenInclude` pour continuer à inclure les niveaux de données associées suivants.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
                .ThenInclude(author => author.Photo)
        .ToList();
}
```

Vous pouvez combiner tout cela pour inclure les données associées de plusieurs niveaux et plusieurs racines dans la même requête.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
                .ThenInclude(author => author.Photo)
        .Include(blog => blog.Owner)
            .ThenInclude(owner => owner.Photo)
        .ToList();
}
```

Vous pourriez souhaiter inclure plusieurs entités associées pour une entité qui est incluse. Par exemple, quand vous interrogez des `Blogs`, vous incluez `Posts`, puis souhaitez inclure à la fois les `Author` et les `Tags` des `Posts`. Pour ce faire, vous devez spécifier chaque chemin d'accès à inclure à partir de la racine. Par exemple : `Blog -> Posts -> Author` et `Blog -> Posts -> Tags`. Cela ne signifie pas que vous obtiendrez des jointures redondantes. Dans la plupart des cas, EF consolide les jointures lors de la génération du SQL.

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Author)
        .Include(blog => blog.Posts)
            .ThenInclude(post => post.Tags)
        .ToList();
}
```

#### Caution

Depuis la version 3.0.0, chaque `Include` entraîne l'ajout d'une jointure supplémentaire aux requêtes SQL générées par les fournisseurs relationnels, tandis que les versions précédentes généraient des requêtes SQL supplémentaires. Cela peut considérablement modifier les performances de vos requêtes. En particulier, les requêtes LINQ avec un nombre excessif de `Include` opérateurs peuvent devoir être décomposées en plusieurs

requêtes LINQ distinctes afin d'éviter le problème d'éclatement cartésien.

## Inclure des types dérivés

Vous pouvez inclure des données associées provenant de navigations définies uniquement sur un type dérivé à l'aide de `Include` et `ThenInclude`.

En partant du modèle suivant :

```
public class SchoolContext : DbContext
{
    public DbSet<Person> People { get; set; }
    public DbSet<School> Schools { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<School>().HasMany(s => s.Students).WithOne(s => s.School);
    }
}

public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Student : Person
{
    public School School { get; set; }
}

public class School
{
    public int Id { get; set; }
    public string Name { get; set; }

    public List<Student> Students { get; set; }
}
```

Le contenu de la navigation `School` de toutes les personnes qui sont des étudiants peut être chargé dynamiquement à l'aide d'un certain nombre de modèles :

- utilisation du forçage de type

```
context.People.Include(person => ((Student)person).School).ToList()
```

- utilisation de l'opérateur `as`

```
context.People.Include(person => (person as Student).School).ToList()
```

- utilisation de la surcharge de `Include` qui accepte les paramètres de type `string`

```
context.People.Include("School").ToList()
```

## Chargement explicite

Vous pouvez charger explicitement une propriété de navigation via l'API `DbContext.Entry(...)`.

```

using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    context.Entry(blog)
        .Collection(b => b.Posts)
        .Load();

    context.Entry(blog)
        .Reference(b => b.Owner)
        .Load();
}

```

Vous pouvez également explicitement charger une propriété de navigation en exécutant une requête distincte qui retourne les entités associées. Si le suivi des modifications est activé, lors du chargement d'une entité, EF Core définit automatiquement les propriétés de navigation de l'entité qui vient d'être chargée pour faire référence à toutes les entités déjà chargées et définir les propriétés de navigation des entités déjà chargées pour faire référence à l'entité qui vient d'être chargée.

### Interrogation des entités associées

Vous pouvez également obtenir une requête LINQ qui représente le contenu d'une propriété de navigation.

Cela vous permet d'effectuer des opérations telles que l'exécution d'un opérateur d'agrégation sur les entités associées sans les charger dans la mémoire.

```

using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    var postCount = context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Count();
}

```

Vous pouvez également filtrer les entités associées qui sont chargées en mémoire.

```

using (var context = new BloggingContext())
{
    var blog = context.Blogs
        .Single(b => b.BlogId == 1);

    var goodPosts = context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Where(p => p.Rating > 3)
        .ToList();
}

```

## Chargement différé

La façon la plus simple d'utiliser le chargement différé est d'installer le package

[Microsoft.EntityFrameworkCore.Proxies](#) et de l'activer avec un appel à `UseLazyLoadingProxies`. Exemple :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder
        .UseLazyLoadingProxies()
        .UseSqlServer(myConnectionString);
```

Ou bien, lors de l'utilisation d'AddDbContext :

```
.AddDbContext<BloggingContext>(
    b => b.UseLazyLoadingProxies()
        .UseSqlServer(myConnectionString));
```

EF Core active ensuite le chargement différé pour n'importe quelle propriété de navigation qui peut être substituée, c'est-à-dire qui doit être `virtual` et sur une classe qui peut être héritée. Par exemple, dans les entités suivantes, les propriétés de navigation `Post.Blog` et `Blog.Posts` seront chargées en différé.

```
public class Blog
{
    public int Id { get; set; }
    public string Name { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public virtual Blog Blog { get; set; }
}
```

### Chargement différé sans proxy

Les proxys à chargement différé fonctionnent en injectant le service `ILazyLoader` dans une entité, comme décrit dans [Constructeurs de type d'entité](#) Exemple :

```

public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}

public class Post
{
    private Blog _blog;

    public Post()
    {
    }

    private Post(ILazyLoader lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private ILazyLoader LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}

```

Il n'est pas nécessaire que l'héritage des types d'entité à partir des propriétés de navigation soit virtuel, et cela permet aux instances d'entité créées avec `new` d'être chargées en différé une fois jointes à un contexte. Toutefois, il requiert une référence au service `ILazyLoader`, qui est défini dans le package [Microsoft.EntityFrameworkCore.Abstractions](#). Ce package contient un ensemble minimal de types, de sorte que dépendre de celui-ci n'a qu'un impact très limité. Toutefois, pour éviter complètement de dépendre des packages EF Core dans les types d'entité, il est possible d'injecter la méthode `ILazyLoader.Load` en tant que délégué. Exemple :

```

public class Blog
{
    private ICollection<Post> _posts;

    public Blog()
    {
    }

    private Blog(Action<object, string> lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private Action<object, string> LazyLoader { get; set; }

    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Post> Posts
    {
        get => LazyLoader.Load(this, ref _posts);
        set => _posts = value;
    }
}

public class Post
{
    private Blog _blog;

    public Post()
    {
    }

    private Post(Action<object, string> lazyLoader)
    {
        LazyLoader = lazyLoader;
    }

    private Action<object, string> LazyLoader { get; set; }

    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog
    {
        get => LazyLoader.Load(this, ref _blog);
        set => _blog = value;
    }
}

```

Le code ci-dessus utilise une méthode d'extension `Load` pour rendre l'utilisation du délégué un peu plus propre :

```
public static class PocoLoadingExtensions
{
    public static TRelated Load<TRelated>(
        this Action<object, string> loader,
        object entity,
        ref TRelated navigationField,
        [CallerMemberName] string navigationName = null)
        where TRelated : class
    {
        loader?.Invoke(entity, navigationName);

        return navigationField;
    }
}
```

#### NOTE

Le paramètre de constructeur pour le délégué à chargement différé doit être appelé « `lazyLoader` ». La configuration permettant d'utiliser un nom différent est prévue pour une version ultérieure.

## Données associées et sérialisation

Étant donné qu'EF Core corrigera automatiquement les propriétés de navigation, vous pouvez vous retrouver avec des cycles dans votre graphique d'objets. Par exemple, le chargement d'un blog et de ses billets associés aboutit à un objet de blog qui fait référence à une collection de billets. Chacun de ces billets aura une référence vers le blog.

Certaines infrastructures de sérialisation n'autorisent pas de tels cycles. Par exemple, Json.NET lève l'exception suivante si un cycle est rencontré.

```
Newtonsoft.Json.JsonSerializationException: Self referencing loop detected for property 'Blog' with type  
'MyApplication.Models.Blog'.
```

Si vous utilisez ASP.NET Core, vous pouvez configurer Json.NET pour ignorer les cycles qu'il trouve dans le graphique d'objets. Cette opération est effectuée dans la méthode `ConfigureServices(...)` de `Startup.cs`.

```
public void ConfigureServices(IServiceCollection services)
{
    ...

    services.AddMvc()
        .AddJsonOptions(
            options => options.SerializerSettings.ReferenceLoopHandling =
Newtonsoft.Json.ReferenceLoopHandling.Ignore
        );

    ...
}
```

Vous pouvez aussi décorer une des propriétés de navigation avec l'attribut `[JsonIgnore]`, ce qui indique à Json.NET de ne pas parcourir cette propriété de navigation lors de la sérialisation.

# Requêtes asynchrones

11/10/2019 • 2 minutes to read • [Edit Online](#)

Les requêtes asynchrones évitent de bloquer un thread pendant que la requête est exécutée dans la base de données. Les requêtes Async sont importantes pour conserver une interface utilisateur réactive dans des applications clientes lourdes. Ils peuvent également augmenter le débit dans les applications Web où ils libèrent le thread pour traiter d'autres requêtes dans des applications Web. Pour plus d'informations sur la programmation asynchrone, consultez [Programmation asynchrone en C#](#).

## WARNING

EF Core ne prend pas en charge l'exécution de plusieurs opérations parallèles sur la même instance de contexte. Vous devez toujours attendre qu'une opération se termine avant de commencer l'opération suivante. Cela s'effectue généralement en utilisant le mot clé `await` sur chaque opération asynchrone.

Entity Framework Core fournit un ensemble de méthodes d'extension Async similaires aux méthodes LINQ, qui exécutent une requête et retournent des résultats. Exemples : `ToListAsync()`, `ToDictionaryAsync()`, `SingleAsync()`. Il n'existe aucune version asynchrone de certains opérateurs LINQ, tels que `Where(...)` ou `OrderBy(...)`, car ces méthodes génèrent uniquement l'arborescence de l'expression LINQ et n'entraînent pas l'exécution de la requête dans la base de données.

## IMPORTANT

Les méthodes d'extension EF Core asynchrones sont définies dans l'espace de noms `Microsoft.EntityFrameworkCore`. Cet espace de noms doit être importé pour que les méthodes soient disponibles.

```
public async Task<List<Blog>> GetBlogsAsync()
{
    using (var context = new BloggingContext())
    {
        return await context.Blogs.ToListAsync();
    }
}
```

# Requêtes SQL brutes

17/10/2019 • 9 minutes to read • [Edit Online](#)

Entity Framework Core vous permet d'examiner les requêtes SQL brutes lorsque vous travaillez avec une base de données relationnelle. Les requêtes SQL brutes sont utiles si la requête que vous souhaitez ne peut pas être exprimée à l'aide de LINQ. Les requêtes SQL brutes sont également utilisées si une requête LINQ aboutit à une requête SQL inefficace. Les requêtes SQL brutes peuvent retourner des types d'entité standard ou des [types d'entité sans clé](#) qui font partie de votre modèle.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Requêtes SQL brutes de base

Vous pouvez utiliser la méthode d'extension `FromSqlRaw` pour commencer une requête LINQ basée sur une requête SQL brute. `FromSqlRaw` ne peut être utilisé que sur les racines de requête, qui se trouvent directement sur la `DbSet<>`.

```
var blogs = context.Blogs
    .FromSqlRaw("SELECT * FROM dbo.Blogs")
    .ToList();
```

Les requêtes SQL brutes peuvent servir à exécuter une procédure stockée.

```
var blogs = context.Blogs
    .FromSqlRaw("EXECUTE dbo.GetMostPopularBlogs")
    .ToList();
```

## Passage de paramètres

### WARNING

#### Toujours utiliser le paramétrage pour les requêtes SQL brutes

Lorsque vous introduisez des valeurs fournies par l'utilisateur dans une requête SQL brute, vous devez veiller à éviter les attaques par injection SQL. En plus de valider le fait que ces valeurs ne contiennent pas de caractères non valides, utilisez toujours le paramétrage qui envoie les valeurs séparées du texte SQL.

En particulier, ne transmettez jamais une chaîne concaténée ou interpolée (`""`) avec des valeurs non validées fournies par l'utilisateur dans `FromSqlRaw` ou `ExecuteSqlRaw`. Les méthodes `FromSqlInterpolated` et `ExecuteSqlInterpolated` permettent d'utiliser la syntaxe d'interpolation de chaîne d'une manière qui protège contre les attaques par injection SQL.

L'exemple suivant passe un paramètre unique à une procédure stockée en incluant un espace réservé de paramètre dans la chaîne de requête SQL et en fournissant un argument supplémentaire. Alors que cette syntaxe peut ressembler à la syntaxe `String.Format`, la valeur fournie est encapsulée dans une `SqlParameter` et le nom de paramètre généré est inséré à l'endroit où l'espace réservé `{0}` a été spécifié.

```
var user = "johndoe";

var blogs = context.Blogs
    .FromSqlRaw("EXECUTE dbo.GetMostPopularBlogsForUser {0}", user)
    .ToList();
```

`FromSqlInterpolated` est semblable à `FromSqlRaw`, mais vous permet d'utiliser la syntaxe d'interpolation de chaîne. Tout comme `FromSqlRaw`, `FromSqlInterpolated` ne peut être utilisé que sur les racines de requête. Comme dans l'exemple précédent, la valeur est convertie en `DbParameter` et n'est pas vulnérable à l'injection SQL.

#### NOTE

Avant la version 3.0, les `FromSqlRaw` et `FromSqlInterpolated` étaient deux surcharges nommées `FromSql`. Pour plus d'informations, consultez la [section versions précédentes](#).

```
var user = "johndoe";

var blogs = context.Blogs
    .FromSqlInterpolated($"EXECUTE dbo.GetMostPopularBlogsForUser {user}")
    .ToList();
```

Vous pouvez également construire un objet `SqlParameter` et le fournir en tant que valeur de paramètre. Étant donné qu'un espace réservé de paramètre SQL standard est utilisé, plutôt qu'un espace réservé de chaîne, `FromSqlRaw` peut être utilisé en toute sécurité :

```
var user = new SqlParameter("user", "johndoe");

var blogs = context.Blogs
    .FromSqlRaw("EXECUTE dbo.GetMostPopularBlogsForUser @user", user)
    .ToList();
```

`FromSqlRaw` vous permet d'utiliser des paramètres nommés dans la chaîne de requête SQL, ce qui est utile lorsqu'une procédure stockée a des paramètres facultatifs :

```
var user = new SqlParameter("user", "johndoe");

var blogs = context.Blogs
    .FromSqlRaw("EXECUTE dbo.GetMostPopularBlogsForUser @filterByUser=@user", user)
    .ToList();
```

## Composition avec LINQ

Vous pouvez composer en haut de la requête SQL brute initiale à l'aide des opérateurs LINQ. EF Core le traitera comme sous-requête et composera dessus dans la base de données. L'exemple suivant utilise une requête SQL brute qui effectue une sélection à partir d'une fonction table (TVF). Puis le compose à l'aide de LINQ pour effectuer un filtrage et un tri.

```
var searchTerm = ".NET";

var blogs = context.Blogs
    .FromSqlInterpolated($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
    .Where(b => b.Rating > 3)
    .OrderByDescending(b => b.Rating)
    .ToList();
```

La requête ci-dessus génère le code SQL suivant :

```
SELECT [b].[BlogId], [b].[OwnerId], [b].[Rating], [b].[Url]
FROM (
    SELECT * FROM dbo.SearchBlogs(@p0)
) AS [b]
WHERE [b].[Rating] > 3
ORDER BY [b].[Rating] DESC
```

## Inclusion de données associées

La méthode `Include` peut être utilisée pour inclure des données associées, comme avec toute autre requête LINQ :

```
var searchTerm = ".NET";

var blogs = context.Blogs
    .FromSqlInterpolated($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
    .Include(b => b.Posts)
    .ToList();
```

La composition avec LINQ nécessite que votre requête SQL brute soit composable, car EF Core traite le SQL fourni comme sous-requête. Les requêtes SQL qui peuvent être composées commencent par le mot clé `SELECT`. En outre, SQL passé ne doit pas contenir de caractères ou d'options qui ne sont pas valides dans une sous-requête, par exemple :

- Point-virgule de fin
- Sur le serveur SQL Server, une indication de niveau de requête en fin (par exemple, `OPTION (HASH JOIN)`)
- Sur SQL Server, une clause `ORDER BY` qui n'est pas utilisée avec `OFFSET 0` ou `TOP 100 PERCENT` dans la clause `SELECT`

SQL Server n'autorise pas la composition sur les appels de procédure stockée, toute tentative d'appliquer des opérateurs de requête supplémentaires à un tel appel aura pour résultat un SQL non valide. Utilisez la méthode `AsEnumerable` ou `AsAsyncEnumerable` juste après les méthodes `FromSqlRaw` ou `FromSqlInterpolated` pour vous assurer que EF Core n'essaie pas de composer une procédure stockée.

## Suivi des modifications

Les requêtes qui utilisent les méthodes `FromSqlRaw` ou `FromSqlInterpolated` suivent exactement les mêmes règles de suivi des modifications que toute autre requête LINQ dans EF Core. Par exemple, si la requête projette des types d'entités, les résultats sont suivis par défaut.

L'exemple suivant utilise une requête SQL brute qui effectue une sélection à partir d'une fonction table (TVF), puis désactive le suivi des modifications avec l'appel à `AsNoTracking` :

```
var searchTerm = ".NET";

var blogs = context.Blogs
    .FromSqlInterpolated($"SELECT * FROM dbo.SearchBlogs({searchTerm})")
    .AsNoTracking()
    .ToList();
```

## Limitations

Il existe quelques limitations à connaître lors de l'utilisation des requêtes SQL brutes :

- La requête SQL doit retourner des données pour toutes les propriétés du type d'entité.
- Les noms de colonne dans le jeu de résultats doivent correspondre aux noms de colonne mappés aux propriétés. Notez que ce comportement est différent de EF6. EF6 ignore la propriété pour le mappage de colonnes pour les requêtes SQL brutes et les noms de colonnes du jeu de résultats devaient correspondre aux noms de propriété.
- La requête SQL ne peut pas contenir de données associées. Toutefois, dans de nombreux cas, vous pouvez composer au-dessus de la requête à l'aide de l'opérateur `Include` pour retourner des données associées (consultez [Inclusion de données associées](#)).

## Versions antérieures

EF Core version 2.2 et les versions antérieures comportaient deux surcharges de méthode nommées `FromSql`, qui se présentaient de la même façon que la plus récente `FromSqlRaw` et `FromSqlInterpolated`. Il était facile d'appeler par erreur la méthode de chaîne brute lorsque l'intention était d'appeler la méthode de chaîne interpolée, et d'inverse. L'appel accidentel d'une surcharge incorrecte peut entraîner des requêtes qui ne sont pas paramétrables quand elles devraient l'être.

# Filtres de requête globale

06/09/2019 • 4 minutes to read • [Edit Online](#)

## NOTE

Cette fonctionnalité date de la publication d'EF Core 2.0.

Les filtres de requête globale permet aux prédictats de requête LINQ (expression booléenne généralement passée à l'opérateur de requête LINQ *Where*) d'être appliqués sur des types d'entité dans le modèle de métadonnées (généralement dans *OnModelCreating*). Ces filtres sont automatiquement appliqués à toutes les requêtes LINQ impliquant ces types d'entités, y compris ceux référencés indirectement, par exemple à l'aide d'*Include* ou de références de propriété de navigation directe. Voici deux applications courantes de cette fonctionnalité :

- **Suppression réversible** : un type d'entité définit une propriété *IsDeleted*.
- **Architecture multilocataire** : un type d'entité définit une propriété *TenantId*.

## Exemple

L'exemple suivant montre comment utiliser les filtres de requête globale pour implémenter des comportements de suppression réversible et d'architecture multilocataire dans un modèle de création de blogs simple.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

Tout d'abord définissez les entités :

```
public class Blog
{
    private string _tenantId;

    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public bool IsDeleted { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

Notez la déclaration d'un champ *tenantId* sur l'entité *Blog*. Celui-ci doit servir à associer chaque instance de blog à un client spécifique. Une propriété *IsDeleted* est également définie sur le type d'entité *Post*. Elle est utilisée pour déterminer si une instance *Post* a été « supprimée de façon réversible ». Autrement dit, l'instance est marquée comme supprimée sans que les données sous-jacentes soient réellement supprimées.

Ensuite, configurez les filtres de requête dans *OnModelCreating* à l'aide de l'API `HasQueryFilter`.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>().Property<string>("TenantId").HasField("_tenantId");

    // Configure entity filters
    modelBuilder.Entity<Blog>().HasQueryFilter(b => EF.Property<string>(b, "TenantId") == _tenantId);
    modelBuilder.Entity<Post>().HasQueryFilter(p => !p.IsDeleted);
}
```

Les expressions de prédictat passées aux appels `HasQueryFilter` seront désormais automatiquement appliquées à toutes les requêtes LINQ pour ces types.

#### TIP

Notez l'utilisation d'un champ de niveau d'instance `DbContext` : `_tenantId` permet de définir le client en cours. Les filtres au niveau du modèle utilisent la valeur de l'instance de contexte correcte (c'est-à-dire celle qui exécute la requête).

#### NOTE

Il n'est actuellement pas possible de définir plusieurs filtres de requête sur la même entité ; seul le dernier sera appliqué. Toutefois, vous pouvez définir un filtre unique avec plusieurs conditions à l'aide de l'opérateur `and` logique (`&&` dans C#).

## Désactivation des filtres

Les filtres peuvent être désactivés pour des requêtes LINQ individuelles à l'aide de l'opérateur

```
IgnoreQueryFilters()
```

```
blogs = db.Blogs
    .Include(b => b.Posts)
    .IgnoreQueryFilters()
    .ToList();
```

## Limitations

Les filtres de requête globale présentent les limitations suivantes :

- Les filtres ne peuvent pas contenir de références à des propriétés de navigation.
- Les filtres ne peuvent être définis que pour le type d'entité racine d'une hiérarchie d'héritage.

# Balises de requête

07/11/2019 • 2 minutes to read • [Edit Online](#)

## NOTE

Cette fonctionnalité est une nouveauté d'EF Core 2.2.

Cette fonctionnalité simplifie la corrélation des requêtes LINQ dans le code avec des requêtes SQL générées et capturées dans des journaux. Vous annotez une requête LINQ à l'aide de la nouvelle méthode `TagWith()` :

```
var nearestFriends =
    (from f in context.Friends.TagWith("This is my spatial query!")
     orderby f.Location.Distance(myLocation) descending
     select f).Take(5).ToList();
```

Cette requête LINQ est traduite dans l'instruction SQL suivante :

```
-- This is my spatial query!

SELECT TOP(@__p_1) [f].[Name], [f].[Location]
FROM [Friends] AS [f]
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

Il est possible d'appeler `TagWith()` plusieurs fois sur la même requête. Les balises de requête sont cumulatives. Par exemple, avec les méthodes suivantes :

```
IQueryable<Friend> GetNearestFriends(Point myLocation) =>
    from f in context.Friends.TagWith("GetNearestFriends")
    orderby f.Location.Distance(myLocation) descending
    select f;

IQueryable<T> Limit<T>(IQueryable<T> source, int limit) =>
    source.TagWith("Limit").Take(limit);
```

Cette requête :

```
var results = Limit(GetNearestFriends(myLocation), 25).ToList();
```

Se traduit par :

```
-- GetNearestFriends

-- Limit

SELECT TOP(@__p_1) [f].[Name], [f].[Location]
FROM [Friends] AS [f]
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

Il est également possible d'utiliser des chaînes multilignes comme balises de requête. Exemple :

```
var results = Limit(GetNearestFriends(myLocation), 25).TagWith(
    @"This is a multi-line
    string").ToList();
```

Génère l'instruction SQL suivante :

```
-- GetNearestFriends

-- Limit

-- This is a multi-line
-- string

SELECT TOP(@__p_1) [f].[Name], [f].[Location]
FROM [Friends] AS [f]
ORDER BY [f].[Location].STDistance(@__myLocation_0) DESC
```

## Limitations connues

**Les balises de requête ne sont pas paramétrables :** EF Core traite toujours les balises de requête dans la requête LINQ comme des opérateurs sur chaîne inclus dans l'instruction SQL générée. Les requêtes compilées qui utilisent des balises de requête comme paramètres ne sont pas autorisées.

# Fonctionnement des requêtes

07/11/2019 • 5 minutes to read • [Edit Online](#)

Entity Framework Core utilise LINQ (Language Integrated Query) pour interroger les données de la base de données. LINQ vous permet d'utiliser C# (ou le langage .NET de votre choix) pour écrire des requêtes fortement typées en fonction de votre contexte dérivé et de vos classes d'entité.

## La durée de vie d'une requête

Voici une vue d'ensemble de haut niveau du processus que chaque requête traverse.

1. La requête LINQ est traitée par Entity Framework Core pour générer une représentation qui est prête à être traitée par le fournisseur de base de données
  - a. Le résultat est mis en cache afin que ce traitement n'ait pas besoin d'être effectué chaque fois que la requête est exécutée
2. Le résultat est transmis au fournisseur de base de données
  - a. Le fournisseur de base de données identifie les parties de la requête qui peuvent être évaluées dans la base de données
  - b. Ces parties de la requête sont traduites en langage de requête spécifique de base de données (par exemple, SQL pour une base de données relationnelle)
  - c. Une ou plusieurs requêtes sont envoyées à la base de données et un jeu de résultats est retourné (les résultats sont des valeurs de la base de données, et non des instances d'entité)
3. Pour chaque élément du jeu de résultats
  - a. S'il s'agit d'une requête de suivi, EF vérifie si les données représentent une entité déjà présente dans le traceur de modifications pour l'instance de contexte
    - Dans ce cas, l'entité existante est retournée
    - Si ce n'est pas le cas, une nouvelle entité est créée, le suivi des modifications est configuré, et la nouvelle entité est retournée
  - b. S'il s'agit d'une requête sans suivi, EF vérifie si les données représentent une entité déjà présente dans le jeu de résultats pour cette requête
    - Dans ce cas, l'entité existante est retournée<sup>(1)</sup>
    - Dans le cas contraire, une nouvelle entité est créée et retournée

<sup>(1)</sup> les requêtes de suivi non utilisent des références faibles pour effectuer le suivi des entités qui ont déjà été retournées. Si un résultat antérieur avec la même identité est hors de portée, et que le nettoyage de la mémoire s'exécute, vous risquez d'obtenir une nouvelle instance de l'entité.

## Lorsque des requêtes sont exécutées

Lorsque vous appelez des opérateurs LINQ, vous créez simplement une représentation en mémoire de la requête. La requête est envoyée à la base de données uniquement lorsque les résultats sont consommés.

Les opérations qui génèrent les requêtes envoyées à la base de données les plus courantes sont :

- Itération des résultats dans une boucle `for`
- Utilisation d'un opérateur tel que `ToList`, `ToArray`, `Single`, `Count`
- Liaison des données des résultats d'une requête sur une interface utilisateur

#### **WARNING**

**Toujours valider l'entrée d'utilisateur :** bien qu'EF Core protège contre les attaques par injection SQL au moyen de paramètres et de l'échappement des littéraux dans les requêtes, il ne valide pas les entrées. Vous devez effectuer une validation adéquate, conformément aux exigences de l'application, avant que les valeurs provenant d'une source non fiable soient utilisées dans des requêtes LINQ, affectées à des propriétés d'entité ou transmises à d'autres API EF Core. Cela inclut toute entrée utilisateur utilisée pour construire des requêtes de façon dynamique. Même si vous utilisez LINQ, si vous acceptez les entrées d'utilisateur pour générer des expressions, vous devez vous assurer que seules les expressions prévues peuvent être construites.

# Enregistrement de données

28/08/2018 • 2 minutes to read • [Edit Online](#)

Chaque instance de contexte a un `ChangeTracker` qui est responsable du suivi des modifications à écrire dans la base de données. Quand vous apportez des modifications à des instances de vos classes d'entité, ces modifications sont enregistrées dans le `ChangeTracker`, puis écrites dans la base de données quand vousappelez `SaveChanges`. Le fournisseur de base de données est chargé de traduire les modifications en opérations de base de données (par exemple les commandes `INSERT`, `UPDATE` et `DELETE` pour une base de données relationnelle).

# Enregistrement de base

24/09/2019 • 3 minutes to read • [Edit Online](#)

Découvrez comment ajouter, modifier et supprimer des données à l'aide de vos classes de contexte et d'entité.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Ajout de données

Utilisez la méthode `DbSet.Add` pour ajouter de nouvelles instances de vos classes d'entité. Les données seront insérées dans la base de données lorsque vous appelez `SaveChanges`.

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Url = "http://sample.com" };
    context.Blogs.Add(blog);
    context.SaveChanges();
}
```

## TIP

Les méthodes `Add`, `Attach` et `Update` fonctionnent toutes sur le graphique complet des entités passées, comme décrit dans la section [Données associées](#). Vous pouvez aussi utiliser la propriété `EntityEntry.State` pour définir l'état d'une seule entité. Par exemple, `context.Entry(blog).State = EntityState.Modified`.

## Mise à jour des données

EF détecte automatiquement les modifications apportées à une entité existante qui est suivie par le contexte. Cela inclut les entités que vous chargez/demandez à partir de la base de données et des entités qui ont été précédemment ajoutées et enregistrées dans la base de données.

Modifiez simplement les valeurs affectées aux propriétés, puisappelez `SaveChanges`.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.First();
    blog.Url = "http://sample.com/blog";
    context.SaveChanges();
}
```

## Suppression de données

Utilisez la méthode `DbSet.Remove` pour supprimer les instances de vos classes d'entité.

Si l'entité existe déjà dans la base de données, elle sera supprimée pendant `SaveChanges`. Si l'entité n'a pas encore été enregistrée dans la base de données (autrement dit, si elle est suivie comme ajoutée), elle sera supprimée du contexte et ne pourra plus être insérée lorsque `SaveChanges` est appelé.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.First();
    context.Blogs.Remove(blog);
    context.SaveChanges();
}
```

## Plusieurs opérations dans un seul SaveChanges

Vous pouvez combiner plusieurs opérations d'ajout/mise à jour/suppression en un seul appel à *SaveChanges*.

### NOTE

Pour la plupart des fournisseurs de base de données, *SaveChanges* est transactionnel. Cela signifie que toutes les opérations réussissent ou échouent complètement et que les opérations ne sont jamais partiellement appliquées.

```
using (var context = new BloggingContext())
{
    // seeding database
    context.Blogs.Add(new Blog { Url = "http://sample.com/blog" });
    context.Blogs.Add(new Blog { Url = "http://sample.com/another_blog" });
    context.SaveChanges();
}

using (var context = new BloggingContext())
{
    // add
    context.Blogs.Add(new Blog { Url = "http://sample.com/blog_one" });
    context.Blogs.Add(new Blog { Url = "http://sample.com/blog_two" });

    // update
    var firstBlog = context.Blogs.First();
    firstBlog.Url = "";

    // remove
    var lastBlog = context.Blogs.Last();
    context.Blogs.Remove(lastBlog);

    context.SaveChanges();
}
```

# Enregistrement des données associées

24/09/2019 • 4 minutes to read • [Edit Online](#)

En plus des entités isolées, vous pouvez également utiliser les relations définies dans votre modèle.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Ajout d'un graphique de nouvelles entités

Si vous créez plusieurs nouvelles entités associées, l'ajout d'une d'elles au contexte provoquera l'ajout des autres.

Dans l'exemple suivant, le blog et trois billets associés sont tous insérés dans la base de données. Les billets sont trouvés et ajoutés, car ils sont accessibles via la propriété de navigation `Blog.Posts`.

```
using (var context = new BloggingContext())
{
    var blog = new Blog
    {
        Url = "http://blogs.msdn.com/dotnet",
        Posts = new List<Post>
        {
            new Post { Title = "Intro to C#" },
            new Post { Title = "Intro to VB.NET" },
            new Post { Title = "Intro to F#" }
        }
    };
    context.Blogs.Add(blog);
    context.SaveChanges();
}
```

## TIP

Utilisez la propriété `EntityEntry.State` pour définir l'état d'une seule entité. Par exemple,

```
context.Entry(blog).State = EntityState.Modified.
```

## Ajout d'une entité associée

Si vous référez une nouvelle entité à partir de la propriété de navigation d'une entité qui est déjà suivie par le contexte, l'entité est découverte et insérée dans la base de données.

Dans l'exemple suivant, l'entité `post` est insérée car elle est ajoutée à la propriété `Posts` de l'entité `blog` qui a été récupérée à partir de la base de données.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Include(b => b.Posts).First();
    var post = new Post { Title = "Intro to EF Core" };

    blog.Posts.Add(post);
    context.SaveChanges();
}
```

## Modification de relations

Si vous modifiez la propriété de navigation d'une entité, les modifications correspondantes porteront sur la colonne de clé étrangère dans la base de données.

Dans l'exemple suivant, l'entité `post` est mise à jour pour appartenir à la nouvelle entité `blog`, car sa propriété de navigation `Blog` est définie pour pointer vers `blog`. Notez que `blog` est également inséré dans la base de données, car il s'agit d'une nouvelle entité qui est référencée par la propriété de navigation d'une entité déjà suivie par le contexte (`post`).

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Url = "http://blogs.msdn.com/visualstudio" };
    var post = context.Posts.First();

    post.Blog = blog;
    context.SaveChanges();
}
```

## Suppression de relations

Vous pouvez supprimer une relation en définissant une navigation de référence sur `null`, ou en supprimant de l'entité associée à partir d'une navigation de collection.

Supprimer une relation peut avoir des effets secondaires sur l'entité dépendante, en fonction du comportement de suppression en cascade configuré dans la relation.

Par défaut, pour les relations requises, un comportement de suppression en cascade est configuré et l'entité dépendante/enfant est supprimée de la base de données. La suppression en cascade n'est pas configurée pour les relations facultatives par défaut, mais la propriété de clé étrangère est définie avec la valeur null.

Consultez [Relations obligatoires et facultatives](#) pour en savoir plus sur la configuration de la nécessité des relations.

Consultez [Suppression en cascade](#) pour plus d'informations sur la façon dont les comportements de suppression en cascade fonctionnent, dont ils peuvent être configurés explicitement ou encore être sélectionnés par convention.

Dans l'exemple suivant, une suppression en cascade est configurée sur la relation entre `Blog` et `Post`, donc l'entité `post` est supprimée de la base de données.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Include(b => b.Posts).First();
    var post = blog.Posts.First();

    blog.Posts.Remove(post);
    context.SaveChanges();
}
```

# Suppression en cascade

07/11/2019 • 23 minutes to read • [Edit Online](#)

La suppression en cascade est couramment utilisée dans la terminologie de base de données pour décrire une caractéristique qui permet à la suppression d'une ligne de déclencher automatiquement la suppression de lignes associées. Un concept étroitement lié également couvert par les comportements de suppression d'EF cœur est la suppression automatique d'une entité enfant lorsque sa relation à un parent a été interrompue. Cela est communément appelé « suppression des orphelins ».

EF Core implémente plusieurs comportements de suppression différents et permet la configuration de comportements de suppression de relations individuelles. EF Core implémente également des conventions qui configurent automatiquement les comportements de suppression par défaut utiles pour chaque relation en fonction de la [nécessité de la relation](#).

## Comportements de suppression

Les comportements de suppression sont définis dans le type énumérateur `deleteBehavior()` et peuvent être passés à l'API Fluent `OnDelete` pour contrôler si la suppression d'une entité principale/parente ou l'interruption de la relation avec les entités dépendantes/enfant doit avoir un effet secondaire sur les entités dépendantes/enfant.

Il existe trois mesures qu'EF peut prendre lorsqu'une entité principale/parent est supprimée ou que la relation avec l'enfant est interrompue :

- L'entité dépendante/enfant peut être supprimée
- Les valeurs de clé étrangère de l'enfant peuvent être définies avec la valeur null
- L'enfant reste inchangé

### NOTE

Le comportement de suppression configuré dans le modèle EF Core est uniquement appliqué lorsque l'entité principale est supprimée à l'aide d'EF Core et que les entités dépendantes sont chargées en mémoire (par exemple, pour les suivis dépendants). Un comportement de cascade correspondant doit être configuré dans la base de données pour s'assurer que les données qui ne sont pas suivies par le contexte disposent de la bonne action appliquée. Si vous utilisez EF Core pour créer la base de données, ce comportement en cascade sera configuré pour vous.

Pour la deuxième action ci-dessus, la définition d'une clé étrangère sur la valeur null n'est pas valide si cette clé étrangère n'accepte pas la valeur null. (Une clé étrangère qui n'accepte pas les valeurs NULL est équivalente à une relation obligatoire.) Dans ce cas, EF Core suit que la propriété de clé étrangère a été marquée comme Null jusqu'à ce que `SaveChanges` soit appelé, auquel cas une exception est levée, car la modification ne peut pas être rendue persistante dans la base de données. Cela est similaire à une violation de contrainte de la base de données.

Il existe quatre comportements de suppression, répertoriés dans les tableaux ci-dessous.

### Relations facultatives

Pour les relations facultatives (clé étrangère acceptant la valeur null) il est possible d'enregistrer une valeur null de clé étrangère, ce qui entraîne les conséquences suivantes :

| NOM DU COMPORTEMENT               | EFFET SUR LES ENTITÉS DÉPENDANTES/ENFANT EN MÉMOIRE               | EFFET SUR LES ENTITÉS DÉPENDANTES/ENFANT DANS LA BASE DE DONNÉES  |
|-----------------------------------|---|---|
| <b>Cascade</b>                    | Les entités sont supprimées                                       | Les entités sont supprimées                                       |
| <b>ClientSetNull</b> (par défaut) | Les propriétés de clé étrangère sont définies avec la valeur null | aucune.   |
| <b>SetNull</b>                    | Les propriétés de clé étrangère sont définies avec la valeur null | Les propriétés de clé étrangère sont définies avec la valeur null |
| <b>Restrict</b>                   | aucune.   | aucune.   |

### Relations requises

Pour les relations requises (clé étrangère n'acceptant pas la valeur null) il *n'est pas* possible d'enregistrer une valeur null de clé étrangère, ce qui entraîne les conséquences suivantes :

| NOM DU COMPORTEMENT         | EFFET SUR LES ENTITÉS DÉPENDANTES/ENFANT EN MÉMOIRE | EFFET SUR LES ENTITÉS DÉPENDANTES/ENFANT DANS LA BASE DE DONNÉES |
|-----------------------------|---|--|
| <b>Cascade</b> (par défaut) | Les entités sont supprimées                         | Les entités sont supprimées                                      |
| <b>ClientSetNull</b>        | Lève une exception SaveChanges                      | aucune.  |
| <b>SetNull</b>              | Lève une exception SaveChanges                      | Lève une exception SaveChanges                                   |
| <b>Restrict</b>             | aucune.   | aucune.  |

Dans les tableaux ci-dessus, *Aucun* peut entraîner une violation de contrainte. Par exemple, si une entité principale/enfant est supprimée, mais qu'aucune action n'est effectuée pour modifier la clé étrangère d'une entité dépendante/enfant, la base de données lèvera probablement une exception sur SaveChanges en raison d'une violation de contrainte étrangère.

À un niveau élevé :

- Si vous avez des entités qui ne peuvent pas exister sans un parent, et que vous souhaitez qu'EF se charge de la suppression des enfants automatiquement, utilisez *Cascade*.
  - Les entités qui ne peuvent pas exister sans un parent utilisent généralement les relations requises, dont *Cascade* est la valeur par défaut.
- Si vous avez des entités qui peuvent ou ne peuvent pas avoir un parent, et que vous souhaitez qu'EF prenne en charge la définition de la clé étrangère sur null pour vous, utilisez *ClientSetNull*
  - Les entités qui peuvent exister sans un parent utilisent généralement les relations optionnelles, dont *ClientSetNull* est la valeur par défaut.
  - Si vous souhaitez que la base de données essaye également de propager les valeurs null aux clés étrangères enfant même lorsque l'entité enfant n'est pas chargée, utilisez *SetNull*. Toutefois, notez que la base de données doit prendre en charge cela, et que la configuration de la base de données de cette façon peut entraîner d'autres restrictions, qui dans la pratique rendent souvent cette option inappropriée. C'est pourquoi *SetNull* n'est pas la valeur par défaut.
- Si vous souhaitez qu'EF Core ne supprime jamais une entité automatiquement ou règle automatiquement la clé étrangère sur null, utilisez *Restrict*. Notez que dans ce cas votre code doit synchroniser les entités enfant et leurs valeurs de clé étrangère manuellement. Dans le cas contraire des exceptions de contrainte seront levées.

#### NOTE

Dans EF Core, contrairement à EF6, les effets en cascade ne se produisent pas immédiatement, mais à la place uniquement lorsque `SaveChanges` est appelé.

#### NOTE

**Changements dans EF Core 2.0 :** dans les versions précédentes, `Restrict` provoquerait la définition des propriétés de clé étrangère facultatives dans des entités dépendantes suivies sur null. C'était le comportement de suppression par défaut pour les relations optionnelles. Dans EF Core 2.0, l'option `ClientSetNull` a été introduite pour représenter ce comportement et est devenue la valeur par défaut pour les relations facultatives. Le comportement de `Restrict` a été ajusté pour ne jamais avoir d'effets sur les entités dépendantes.

## Exemples de suppression d'entité

Le code suivant fait partie d'un [exemple](#) qui peut être téléchargé et exécuté. L'exemple montre ce qui se passe pour chaque comportement de suppression pour les relations facultatives et requises lorsqu'une entité parente est supprimée.

```
var blog = context.Blogs.Include(b => b.Posts).First();
var posts = blog.Posts.ToList();

DumpEntities(" After loading entities:", context, blog, posts);

context.Remove(blog);

DumpEntities($" After deleting blog '{blog.BlogId}':", context, blog, posts);

try
{
    Console.WriteLine();
    Console.WriteLine(" Saving changes:");

    context.SaveChanges();

    DumpSql();

    DumpEntities(" After SaveChanges:", context, blog, posts);
}
catch (Exception e)
{
    DumpSql();

    Console.WriteLine();
    Console.WriteLine($" SaveChanges threw {e.GetType().Name}: {(e is DbUpdateException ? e.InnerException.Message : e.Message)}");
}
```

Nous allons étudier chaque variation de comprendre ce qui se passe.

### DeleteBehavior.Cascade avec une relation obligatoire ou facultative

```
After loading entities:  
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
After deleting blog '1':  
Blog '1' is in state Deleted with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
Saving changes:  
DELETE FROM [Posts] WHERE [PostId] = 1  
DELETE FROM [Posts] WHERE [PostId] = 2  
DELETE FROM [Blogs] WHERE [BlogId] = 1
```

```
After SaveChanges:  
Blog '1' is in state Detached with 2 posts referenced.  
Post '1' is in state Detached with FK '1' and no reference to a blog.  
Post '2' is in state Detached with FK '1' and no reference to a blog.
```

- Le blog est marqué comme supprimé
- Les billets restent initialement inchangés étant donné que les cascades ne se produisent pas avant SaveChanges
- SaveChanges envoie des suppressions pour les entités dépendantes/enfant (les billets), puis pour l'entité principale/parent (le blog)
- Après l'enregistrement, toutes les entités sont détachées, car elles ont été supprimées de la base de données

#### DeleteBehavior.ClientSetNull ou DeleteBehavior.SetNull avec une relation requise

```
After loading entities:  
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
After deleting blog '1':  
Blog '1' is in state Deleted with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
Saving changes:  
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1
```

```
SaveChanges threw DbUpdateException: Cannot insert the value NULL into column 'BlogId', table  
'EFSaving.CascadeDelete.dbo.Posts'; column does not allow nulls. UPDATE fails. The statement has been  
terminated.
```

- Le blog est marqué comme supprimé
- Les billets restent initialement inchangés étant donné que les cascades ne se produisent pas avant SaveChanges
- SaveChanges tente de définir la clé étrangère du billet sur null, mais cette opération échoue car la clé étrangère ne peut pas avoir la valeur null

#### DeleteBehavior.ClientSetNull ou DeleteBehavior.SetNull avec une relation facultative

```

After loading entities:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

After deleting blog '1':
Blog '1' is in state Deleted with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

Saving changes:
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 2
DELETE FROM [Blogs] WHERE [BlogId] = 1

After SaveChanges:
Blog '1' is in state Detached with 2 posts referenced.
Post '1' is in state Unchanged with FK 'null' and no reference to a blog.
Post '2' is in state Unchanged with FK 'null' and no reference to a blog.

```

- Le blog est marqué comme supprimé
- Les billets restent initialement inchangés étant donné que les cascades ne se produisent pas avant SaveChanges
- SaveChanges tente de définir clé étrangère des entités dépendantes/enfant (les billets) sur null avant de supprimer l'entité principale/parent (le blog)
- Après l'enregistrement, l'entité principale/parent (le blog) est supprimée, mais les entités dépendantes/enfant (les billets) sont toujours suivies
- Les entités dépendantes/enfant suivies (les billets) ont maintenant des valeurs de clé étrangère null et leur référence à l'entité principale/parent (le blog) a été supprimée

### DeleteBehavior.Restrict avec une relation obligatoire ou facultative

```

After loading entities:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

After deleting blog '1':
Blog '1' is in state Deleted with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

Saving changes:
SaveChanges threw InvalidOperationException: The association between entity types 'Blog' and 'Post' has
been severed but the foreign key for this relationship cannot be set to null. If the dependent entity should
be deleted, then setup the relationship to use cascade deletes.

```

- Le blog est marqué comme supprimé
- Les billets restent initialement inchangés étant donné que les cascades ne se produisent pas avant SaveChanges
- Étant donné que *Restrict* indique à EF de ne pas automatiquement définir la clé étrangère sur null, elle reste non null et SaveChanges lève une exception sans enregistrer

## Exemples de suppression d'orphelins

Le code suivant fait partie d'un [exemple](#) qui peut être téléchargé et exécuté. L'exemple montre ce qui se passe pour chaque comportement de suppression pour les relations facultatives et requises lorsque la relation entre une entité principale/parent et ses entités dépendantes/enfant est interrompue. Dans cet exemple, la relation est

rompue en supprimant les entités dépendantes/enfant (les billets) de la propriété de navigation de collection sur l'entité principale/parent (le blog). Toutefois, le comportement est le même si la référence de l'entité dépendante/enfant vers l'entité principale/parent est annulée à la place.

```
var blog = context.Blogs.Include(b => b.Posts).First();
var posts = blog.Posts.ToList();

DumpEntities(" After loading entities:", context, blog, posts);

blog.Posts.Clear();

DumpEntities(" After making posts orphans:", context, blog, posts);

try
{
    Console.WriteLine();
    Console.WriteLine(" Saving changes:");

    context.SaveChanges();

    DumpSql();

    DumpEntities(" After SaveChanges:", context, blog, posts);
}
catch (Exception e)
{
    DumpSql();

    Console.WriteLine();
    Console.WriteLine($" SaveChanges threw {e.GetType().Name}: {(e is DbUpdateException ? e.InnerException.Message : e.Message)}");
}
```

Nous allons étudier chaque variation de comprendre ce qui se passe.

### DeleteBehavior.Cascade avec une relation obligatoire ou facultative

```
After loading entities:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.

After making posts orphans:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Modified with FK '1' and no reference to a blog.
Post '2' is in state Modified with FK '1' and no reference to a blog.

Saving changes:
DELETE FROM [Posts] WHERE [PostId] = 1
DELETE FROM [Posts] WHERE [PostId] = 2

After SaveChanges:
Blog '1' is in state Unchanged with 2 posts referenced.
Post '1' is in state Detached with FK '1' and no reference to a blog.
Post '2' is in state Detached with FK '1' and no reference to a blog.
```

- Les billets sont marqués comme modifiés, car l'interruption de la relation a provoqué la définition de la clé étrangère sur null
  - Si la clé étrangère ne peut pas être null, la valeur réelle ne changera pas même si elle est marquée en tant que valeur null
- SaveChanges envoie les suppressions pour les entités dépendantes/enfant (les billets)
- Après l'enregistrement, les entités dépendantes/enfant (les billets) sont détachées, car elles ont été supprimées

de la base de données

### DeleteBehavior.ClientSetNull ou DeleteBehavior.SetNull avec une relation requise

After loading entities:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

After making posts orphans:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Modified with FK 'null' and no reference to a blog.  
Post '2' is in state Modified with FK 'null' and no reference to a blog.
```

Saving changes:

```
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1
```

```
SaveChanges threw DbUpdateException: Cannot insert the value NULL into column 'BlogId', table  
'EFSaving.CascadeDelete.dbo.Posts'; column does not allow nulls. UPDATE fails. The statement has been  
terminated.
```

- Les billets sont marqués comme modifiés, car l'interruption de la relation a provoqué la définition de la clé étrangère sur null
  - Si la clé étrangère ne peut pas être null, la valeur réelle ne changera pas même si elle est marquée en tant que valeur null
- SaveChanges tente de définir la clé étrangère du billet sur null, mais cette opération échoue car la clé étrangère ne peut pas avoir la valeur null

### DeleteBehavior.ClientSetNull ou DeleteBehavior.SetNull avec une relation facultative

After loading entities:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

After making posts orphans:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Modified with FK 'null' and no reference to a blog.  
Post '2' is in state Modified with FK 'null' and no reference to a blog.
```

Saving changes:

```
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 1  
UPDATE [Posts] SET [BlogId] = NULL WHERE [PostId] = 2
```

After SaveChanges:

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK 'null' and no reference to a blog.  
Post '2' is in state Unchanged with FK 'null' and no reference to a blog.
```

- Les billets sont marqués comme modifiés, car l'interruption de la relation a provoqué la définition de la clé étrangère sur null
  - Si la clé étrangère ne peut pas être null, la valeur réelle ne changera pas même si elle est marquée en tant que valeur null
- SaveChanges définit la clé étrangère des entités dépendantes/enfant (billets) sur null
- Après enregistrement, les entités dépendantes/enfant suivies (les billets) ont maintenant des valeurs de clé étrangère null et leur référence à l'entité principale/parent (le blog) a été supprimée

### DeleteBehavior.Restrict avec une relation obligatoire ou facultative

```
After loading entities:
```

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Unchanged with FK '1' and reference to blog '1'.  
Post '2' is in state Unchanged with FK '1' and reference to blog '1'.
```

```
After making posts orphans:
```

```
Blog '1' is in state Unchanged with 2 posts referenced.  
Post '1' is in state Modified with FK '1' and no reference to a blog.  
Post '2' is in state Modified with FK '1' and no reference to a blog.
```

```
Saving changes:
```

```
SaveChanges threw InvalidOperationException: The association between entity types 'Blog' and 'Post' has  
been severed but the foreign key for this relationship cannot be set to null. If the dependent entity should  
be deleted, then setup the relationship to use cascade deletes.
```

- Les billets sont marqués comme modifiés, car l'interruption de la relation a provoqué la définition de la clé étrangère sur null
  - Si la clé étrangère ne peut pas être null, la valeur réelle ne changera pas même si elle est marquée en tant que valeur null
- Étant donné que *Restrict* indique à EF de ne pas automatiquement définir la clé étrangère sur null, elle reste non null et SaveChanges lève une exception sans enregistrer

## Cascade pour les entités non suivies

Lorsque vous appelez *SaveChanges*, les règles de suppression en cascade s'appliqueront à toutes les entités qui sont suivies par le contexte. C'est le cas dans tous les exemples ci-dessus, c'est pourquoi le SQL généré pour supprimer l'entité principale/parent (le blog) et toutes les entités dépendantes/enfant (les billets) :

```
DELETE FROM [Posts] WHERE [PostId] = 1  
DELETE FROM [Posts] WHERE [PostId] = 2  
DELETE FROM [Blogs] WHERE [BlogId] = 1
```

Si seule l'entité principale est chargée, par exemple lorsqu'une requête est faite sur un blog sans `Include(b => b.Posts)` pour inclure également les billets, alors *SaveChanges* génère uniquement le SQL pour supprimer l'entité principale/parent :

```
DELETE FROM [Blogs] WHERE [BlogId] = 1
```

Les entités dépendantes/enfant (les billets) seront supprimées uniquement si la base de données a un comportement de cascade correspondant configuré. Si vous utilisez EF pour créer la base de données, ce comportement en cascade sera configuré pour vous.

# Gestion de conflits d'accès concurrentiel

07/11/2019 • 7 minutes to read • [Edit Online](#)

## NOTE

Cette page décrit le fonctionnement de l'accès concurrentiel dans EF Core et comment gérer les conflits d'accès concurrentiel dans votre application. Consultez [Jetons d'accès concurrentiel](#) pour plus d'informations sur la configuration des jetons d'accès concurrentiel dans votre modèle.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

*L'accès concurrentiel à la base de données* fait référence aux situations dans lesquelles plusieurs processus ou utilisateurs accèdent à ou modifient les mêmes données dans une base de données en même temps. Le *Contrôle d'accès concurrentiel* fait référence à des mécanismes spécifiques permettant de garantir la cohérence des données en présence de modifications simultanées.

EF Core implémente un *contrôle d'accès concurrentiel optimiste*, ce qui signifie qu'il permet à plusieurs processus ou utilisateurs d'apporter des modifications indépendamment sans surcharge de synchronisation ou de verrouillage. Dans l'idéal, ces modifications n'interfèrent pas entre elles et sont donc en mesure de réussir. Dans le pire des cas, deux ou plusieurs processus ou plus tentent d'apporter des modifications en conflit, et seulement un d'eux doit réussir.

## Fonctionnement du contrôle d'accès concurrentiel dans EF Core

Les propriétés configurées en tant que jetons d'accès concurrentiel sont utilisées pour implémenter un contrôle d'accès concurrentiel optimiste : chaque fois qu'une opération de mise à jour ou de suppression est effectuée pendant `SaveChanges`, la valeur du jeton d'accès concurrentiel sur la base de données est comparée à la valeur d'origine lue par EF Core.

- Si les valeurs correspondent, l'opération peut s'effectuer.
- Si les valeurs ne correspondent pas, EF Core suppose qu'un autre utilisateur a effectué une opération conflictuelle et abandonne la transaction en cours.

Le cas où un autre utilisateur a effectué une opération en conflit avec l'opération en cours est appelé *Conflit d'accès concurrentiel*.

Les fournisseurs de base de données sont responsables de l'implémentation de la comparaison des valeurs de jeton d'accès concurrentiel.

Sur les bases de données relationnelles, EF Core inclut une vérification de la valeur du jeton d'accès concurrentiel dans la clause `WHERE` de toute instruction `UPDATE` ou `DELETE`. Après l'exécution des instructions, EF Core lit le nombre de lignes affectées.

Si aucune ligne n'est affectée, un conflit d'accès concurrentiel est détecté, et EF Core lève `DbUpdateConcurrencyException`.

Par exemple, nous pouvons choisir de configurer `LastName` sur `Person` comme jeton d'accès concurrentiel. Toute opération de mise à jour sur la personne inclut alors le contrôle d'accès concurrentiel dans la clause `WHERE` :

```
UPDATE [Person] SET [FirstName] = @p1  
WHERE [PersonId] = @p0 AND [LastName] = @p2;
```

## Résolution des conflits d'accès concurrentiel

Dans la suite de l'exemple précédent, si un utilisateur tente d'enregistrer certaines modifications apportées à une `Person`, mais qu'un autre utilisateur a déjà modifié le `LastName`, alors une exception sera levée.

À ce stade, l'application peut simplement indiquer à l'utilisateur que la mise à jour a échoué en raison de modifications en conflit et passer à la suite. Mais il peut être souhaitable d'inviter l'utilisateur à s'assurer que cet enregistrement représente toujours la même personne réelle et à recommencer l'opération.

Ce processus est un exemple de *résolution d'un conflit d'accès concurrentiel*.

La résolution d'un conflit d'accès concurrentiel consiste à fusionner les modifications en attente à partir du `DbContext` actuel avec les valeurs dans la base de données. Les valeurs fusionnées varient en fonction de l'application et peuvent être contrôlées par saisie de l'utilisateur.

### Il existe trois ensembles de valeurs disponibles pour résoudre un conflit d'accès concurrentiel :

- Les **Valeurs actuelles** sont les valeurs que l'application a tenté d'écrire dans la base de données.
- Les **Valeurs d'origine** sont les valeurs qui ont été récupérées à l'origine à partir de la base de données, avant que les modifications soient apportées.
- Les **Valeurs de base de données** sont les valeurs actuellement stockées dans la base de données.

L'approche générale pour gérer les conflits d'accès concurrentiel est la suivante :

1. Intercevez `DbUpdateConcurrencyException` pendant `SaveChanges`.
2. Utilisez `DbUpdateConcurrencyException.Entries` pour préparer un nouvel ensemble de modifications pour les entités concernées.
3. Actualisez les valeurs d'origine du jeton d'accès concurrentiel pour refléter les valeurs actuelles dans la base de données.
4. Recommencez le processus jusqu'à ce qu'aucun conflit ne se produise.

Dans l'exemple suivant, `Person.FirstName` et `Person.LastName` sont configurés en tant que jetons d'accès concurrentiel. Il existe un commentaire `// TODO:` à l'emplacement où vous incluez la logique spécifique de l'application pour choisir la valeur à enregistrer.

```

using (var context = new PersonContext())
{
    // Fetch a person from database and change phone number
    var person = context.People.Single(p => p.PersonId == 1);
    person.PhoneNumber = "555-555-5555";

    // Change the person's name in the database to simulate a concurrency conflict
    context.Database.ExecuteSqlRaw(
        "UPDATE dbo.People SET FirstName = 'Jane' WHERE PersonId = 1");

    var saved = false;
    while (!saved)
    {
        try
        {
            // Attempt to save changes to the database
            context.SaveChanges();
            saved = true;
        }
        catch (DbUpdateConcurrencyException ex)
        {
            foreach (var entry in ex.Entries)
            {
                if (entry.Entity is Person)
                {
                    var proposedValues = entry.CurrentValues;
                    var databaseValues = entry.GetDatabaseValues();

                    foreach (var property in proposedValues.Properties)
                    {
                        var proposedValue = proposedValues[property];
                        var databaseValue = databaseValues[property];

                        // TODO: decide which value should be written to database
                        // proposedValues[property] = <value to be saved>;
                    }

                    // Refresh original values to bypass next concurrency check
                    entry.OriginalValues.SetValues(databaseValues);
                }
                else
                {
                    throw new NotSupportedException(
                        "Don't know how to handle concurrency conflicts for "
                        + entry.Metadata.Name);
                }
            }
        }
    }
}

```

# Utilisation de transactions

07/11/2019 • 9 minutes to read • [Edit Online](#)

Les transactions permettent à plusieurs opérations de base de données d'être traitées de manière atomique. Si la transaction est validée, toutes les opérations sont appliquées avec succès à la base de données. Si la transaction est restaurée, aucune des opérations n'est appliquée à la base de données.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Comportement de transaction par défaut

Par défaut, si le fournisseur de base de données prend en charge les transactions, toutes les modifications dans un seul appel à `SaveChanges()` sont appliquées à une transaction. Si certaines des modifications échouent, la transaction est annulée et aucune des modifications n'est appliquée à la base de données. Cela signifie que `SaveChanges()` réussit complètement ou laisse la base de données non modifiée si une erreur se produit.

Pour la plupart des applications, ce comportement par défaut est suffisant. Vous devez uniquement contrôler manuellement les transactions si les exigences de votre application le jugent nécessaire.

## Contrôle des transactions

Vous pouvez utiliser l'API `dbContext.Database` pour commencer, valider et annuler les transactions. L'exemple suivant montre deux opérations `SaveChanges()` et une requête LINQ en cours d'exécution dans une transaction unique.

Tous les fournisseurs de base de données ne prennent pas en charge les transactions. Certains fournisseurs peuvent lever une exception ou ne pas exécuter d'opération lorsque des API de transaction sont appelées.

```

using (var context = new BloggingContext())
{
    using (var transaction = context.Database.BeginTransaction())
    {
        try
        {
            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
            context.SaveChanges();

            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/visualstudio" });
            context.SaveChanges();

            var blogs = context.Blogs
                .OrderBy(b => b.Url)
                .ToList();

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            transaction.Commit();
        }
        catch (Exception)
        {
            // TODO: Handle failure
        }
    }
}

```

## Transaction à contexte croisé (bases de données relationnelles uniquement)

Vous pouvez également partager une transaction sur plusieurs instances de contexte. Cette fonctionnalité est disponible uniquement lorsque vous utilisez un fournisseur de base de données relationnelle, car elle requiert l'utilisation de `DbTransaction` et `DbConnection`, qui sont propres aux bases de données relationnelles.

Pour partager une transaction, les contextes doivent partager une `DbConnection` et une `DbTransaction`.

### Autoriser la fourniture externe de la connexion

Le partage d'une `DbConnection` nécessite la possibilité de passer une connexion dans un contexte lors de la construction.

Le moyen le plus simple pour autoriser la `DbConnection` à être fournie en externe, arrêtez d'utiliser la méthode `DbContext.OnConfiguring` pour configurer le contexte et créez les `DbContextOptions` en externe avant de les passer au constructeur de contexte.

#### TIP

`DbContextOptionsBuilder` est l'API que vous avez utilisée dans `DbContext.OnConfiguring` pour configurer le contexte. Vous allez maintenant l'utiliser en externe pour créer `DbContextOptions`.

```

public class BloggingContext : DbContext
{
    public BloggingContext(DbContextOptions<BloggingContext> options)
        : base(options)
    { }

    public DbSet<Blog> Blogs { get; set; }
}

```

Une alternative consiste à continuer à utiliser `DbContext.OnConfiguring`, mais accepte une `DbConnection` qui est enregistrée et ensuite utilisée dans `DbContext.OnConfiguring`.

```
public class BloggingContext : DbContext
{
    private DbConnection _connection;

    public BloggingContext(DbConnection connection)
    {
        _connection = connection;
    }

    public DbSet<Blog> Blogs { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connection);
    }
}
```

### Partage de connexions et transactions

Vous pouvez désormais créer plusieurs instances de contexte qui partagent la même connexion. Utilisez ensuite l'API `DbContext.Database.UseTransaction(DbTransaction)` pour inscrire les deux contextes dans la même transaction.

```
var options = new DbContextOptionsBuilder<BloggingContext>()
    .UseSqlServer(new SqlConnection(connectionString))
    .Options;

using (var context1 = new BloggingContext(options))
{
    using (var transaction = context1.Database.BeginTransaction())
    {
        try
        {
            context1.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
            context1.SaveChanges();

            using (var context2 = new BloggingContext(options))
            {
                context2.Database.UseTransaction(transaction.GetDbTransaction());

                var blogs = context2.Blogs
                    .OrderBy(b => b.Url)
                    .ToList();
            }
        }

        // Commit transaction if all commands succeed, transaction will auto-rollback
        // when disposed if either commands fails
        transaction.Commit();
    }
    catch (Exception)
    {
        // TODO: Handle failure
    }
}
```

## Utilisation de DbTransactions externes (bases de données relationnelles uniquement)

Si vous utilisez plusieurs technologies d'accès aux données pour accéder à une base de données relationnelle, vous

souhaiterez partager une transaction entre les opérations effectuées par ces différentes technologies.

L'exemple suivant montre comment effectuer une opération ADO.NET SqlConnection et une opération Entity Framework Core dans la même transaction.

```
using (var connection = new SqlConnection(connectionString))
{
    connection.Open();

    using (var transaction = connection.BeginTransaction())
    {
        try
        {
            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.Transaction = transaction;
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseSqlServer(connection)
                .Options;

            using (var context = new BloggingContext(options))
            {
                context.Database.UseTransaction(transaction);
                context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
                context.SaveChanges();
            }

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            transaction.Commit();
        }
        catch (System.Exception)
        {
            // TODO: Handle failure
        }
    }
}
```

## Utilisation de System.Transactions

### NOTE

Cette fonctionnalité est une nouveauté d'EF Core 2.1.

Il est possible d'utiliser les transactions ambiantes si vous avez besoin de coordonner sur une plus grande portée.

```
using (var scope = new TransactionScope(
    TransactionScopeOption.Required,
    new TransactionOptions { IsolationLevel = IsolationLevel.ReadCommitted }))
{
    using (var connection = new SqlConnection(connectionString))
    {
        connection.Open();

        try
        {
            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            var options = new DbContextOptionsBuilder<BloggingContext>()
                .UseSqlServer(connection)
                .Options;

            using (var context = new BloggingContext(options))
            {
                context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
                context.SaveChanges();
            }

            // Commit transaction if all commands succeed, transaction will auto-rollback
            // when disposed if either commands fails
            scope.Complete();
        }
        catch (System.Exception)
        {
            // TODO: Handle failure
        }
    }
}
```

Il est également possible de s'inscrire dans une transaction explicite.

```

using (var transaction = new CommittableTransaction(
    new TransactionOptions { IsolationLevel = IsolationLevel.ReadCommitted }))
{
    var connection = new SqlConnection(connectionString);

    try
    {
        var options = new DbContextOptionsBuilder<BloggingContext>()
            .UseSqlServer(connection)
            .Options;

        using (var context = new BloggingContext(options))
        {
            context.Database.OpenConnection();
            context.Database.EnlistTransaction(transaction);

            // Run raw ADO.NET command in the transaction
            var command = connection.CreateCommand();
            command.CommandText = "DELETE FROM dbo.Blogs";
            command.ExecuteNonQuery();

            // Run an EF Core command in the transaction
            context.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/dotnet" });
            context.SaveChanges();
            context.Database.CloseConnection();
        }

        // Commit transaction if all commands succeed, transaction will auto-rollback
        // when disposed if either commands fails
        transaction.Commit();
    }
    catch (System.Exception)
    {
        // TODO: Handle failure
    }
}

```

## Limitations de System.Transactions

- EF Core s'appuie sur les fournisseurs de base de données pour implémenter la prise en charge de System.Transactions. Bien que la prise en charge est assez courante parmi les fournisseurs ADO.NET de .NET Framework, l'API n'a été que récemment ajoutée à .NET Core et n'est par conséquent pas aussi répandue. Si un fournisseur n'implémente pas la prise en charge de System.Transactions, il est possible que les appels à ces API soient complètement ignorés. SqlClient pour .NET Core le prend en charge à compter de la version 2.1 et versions ultérieures. SqlClient pour .NET Core 2.0 lève une exception en cas de tentative d'utilisation de la fonctionnalité.

### IMPORTANT

Il est recommandé de vérifier que l'API se comporte correctement avec votre fournisseur avant de l'utiliser pour la gestion des transactions. Nous vous invitons à contacter le chargé de maintenance du fournisseur de base de données si ce n'est pas le cas.

- À compter de la version 2.1, l'implémentation de System.Transactions dans .NET Core n'inclut pas de prise en charge des transactions distribuées. Par conséquent, vous ne pouvez pas utiliser `TransactionScope` ou `CommittableTransaction` pour coordonner des transactions sur plusieurs gestionnaires de ressources.

# Enregistrement asynchrone

24/09/2019 • 2 minutes to read • [Edit Online](#)

L'enregistrement asynchrone évite de bloquer un thread lorsque les modifications sont écrites dans la base de données. Cela peut être utile pour éviter le gel de l'interface utilisateur d'une application client lourde. Les opérations asynchrones peuvent également augmenter le débit dans une application web, où le thread peut être libéré pour d'autres demandes de service lors de la fin de l'opération de base de données. Pour plus d'informations sur la programmation asynchrone, consultez [Programmation asynchrone en C#](#).

## WARNING

EF Core ne prend pas en charge les opérations parallèles multiples en cours d'exécution sur la même instance de contexte. Vous devez toujours attendre qu'une opération se termine avant de commencer l'opération suivante. Cela est généralement effectué à l'aide du mot-clé `await` sur chaque opération asynchrone.

Entity Framework Core fournit `DbContext.SaveChangesAsync()` comme alternative asynchrone à `DbContext.SaveChanges()`.

```
public static async Task AddBlogAsync(string url)
{
    using (var context = new BloggingContext())
    {
        var blog = new Blog { Url = url };
        context.Blogs.Add(blog);
        await context.SaveChangesAsync();
    }
}
```

# Entités déconnectées

07/11/2019 • 14 minutes to read • [Edit Online](#)

Une instance de `DbContext` suit automatiquement les entités retournées à partir de la base de données. Les modifications apportées à ces entités seront ensuite détectées lorsque `SaveChanges` est appelé, et la base de données sera mise à jour en fonction des besoins. Consultez [Enregistrement de base](#) et [Données associées](#) pour plus d'informations.

Toutefois, parfois, les entités sont interrogées par une instance de contexte, puis enregistrées à l'aide d'une autre instance. Cela se produit souvent dans les scénarios « déconnectés », par exemple une application web où les entités sont interrogées, envoyées au client, modifiées, envoyées sur le serveur dans une demande et puis enregistrées. Dans ce cas, la deuxième instance de contexte doit savoir si les entités sont nouvelles (doivent être insérées) ou existantes (doivent être mises à jour).

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## TIP

EF Core peut suivre une seule instance d'une entité avec une valeur de clé primaire donnée. La meilleure façon d'éviter ce problème consiste à utiliser un contexte de courte durée de vie pour chaque unité de travail, de sorte que le contexte commence vide, ait des entités associées, enregistre ces entités, puis soit supprimé.

## Identification des nouvelles entités

### Le client identifie de nouvelles entités

Le cas le plus simple à gérer est lorsque le client indique au serveur si l'entité est nouvelle ou existante. Par exemple, la requête d'insertion d'une nouvelle entité diffère souvent de la requête pour mettre à jour une entité existante.

Le reste de cette section couvre les cas où il est nécessaire de déterminer, d'une autre manière, s'il faut insérer ou mettre à jour.

### Avec des clés générées automatiquement

La valeur d'une clé générée automatiquement peut souvent être utilisée pour déterminer si une entité doit être insérée ou mise à jour. Si la clé n'a pas été définie (autrement dit si elle a toujours la valeur CLR par défaut de null, zéro, etc.), alors l'entité doit être nouvelle et a besoin d'être insérée. En revanche, si la valeur de clé a été définie, l'entité doit avoir déjà été précédemment enregistrée et doit être mise à jour. En d'autres termes, si la clé a une valeur, cette entité a été interrogée, envoyée au client et est maintenant revenue pour être mise à jour.

Il est facile de rechercher une clé non définie lorsque le type d'entité est connu :

```
public static bool IsItNew(Blog blog)
    => blog.BlogId == 0;
```

Toutefois, EF a également un moyen intégré de faire cela pour n'importe quel type d'entité et type de clé :

```
public static bool IsItNew(DbContext context, object entity)
=> !context.Entry(entity).IsKeySet;
```

#### TIP

Les clés sont définies dès que les entités sont suivies par le contexte, même si l'entité est dans l'état Ajouté. Cela est utile lors du parcours d'un graphique d'entités pour décider quelles actions effectuer avec chacune, par exemple lors de l'utilisation de l'API TrackGraph. La valeur de clé doit uniquement être utilisée de la manière illustrée ici *avant* qu'un appel soit envoyé pour effectuer le suivi de l'entité.

### Avec d'autres clés

Un autre mécanisme est nécessaire pour identifier les nouvelles entités lorsque les valeurs des clés ne sont pas générées automatiquement. Il existe deux approches générales pour cela :

- Requête pour l'entité
- Passez un indicateur à partir du client

Pour rechercher l'entité, utilisez simplement la méthode Find :

```
public static bool IsItNew(BloggingContext context, Blog blog)
=> context.Blogs.Find(blog.BlogId) == null;
```

L'affichage du code complet pour le passage d'un indicateur depuis un client n'entre pas dans la portée de ce document. Dans une application web, cela consiste généralement à effectuer des demandes différentes pour différentes actions, ou à passer un état dans la demande, puis l'extraire dans le contrôleur.

## Enregistrement d'entités uniques

Si on sait si une insertion ou une mise à jour est nécessaire, vous pouvez ajouter Add ou Update de manière appropriée :

```
public static void Insert(DbContext context, object entity)
{
    context.Add(entity);
    context.SaveChanges();
}

public static void Update(DbContext context, object entity)
{
    context.Update(entity);
    context.SaveChanges();
}
```

Toutefois, si l'entité utilise les valeurs de clé générées automatiquement, la méthode Update peut être utilisée pour les deux cas :

```
public static void InsertOrUpdate(DbContext context, object entity)
{
    context.Update(entity);
    context.SaveChanges();
}
```

Normalement, la méthode Update marque l'entité pour la mise à jour, et non l'insertion. Toutefois, si l'entité a une clé générée automatiquement, et qu'aucune valeur de clé n'a été définie, l'entité est automatiquement marquée

pour insertion.

#### TIP

Ce comportement a été introduit dans EF Core 2.0. Pour les versions antérieures, il est toujours nécessaire de choisir explicitement Add ou Update.

Si l'entité n'utilise pas les clés générées automatiquement, l'application doit décider si l'entité doit être insérée ou mise à jour. Par exemple :

```
public static void InsertOrUpdate(BloggingContext context, Blog blog)
{
    var existingBlog = context.Blogs.Find(blog.BlogId);
    if (existingBlog == null)
    {
        context.Add(blog);
    }
    else
    {
        context.Entry(existingBlog).CurrentValues.SetValues(blog);
    }

    context.SaveChanges();
}
```

Les étapes ici sont :

- Si Find retourne null, la base de données ne contient pas encore le blog avec cet ID, nous appelons donc Add pour le marquer pour insertion.
- Si la recherche retourne une entité, il existe dans la base de données et le contexte suit maintenant l'entité existante
  - Ensuite, nous utilisons SetValues pour définir les valeurs de toutes les propriétés de cette entité sur celles envoyées par le client.
  - L'appel à SetValues marque l'entité à mettre à jour en fonction des besoins.

#### TIP

SetValues marque uniquement comme modifiées les propriétés qui ont des valeurs différentes de celles de l'entité suivie. Cela signifie que lorsque la mise à jour est envoyée, seules les colonnes qui ont été modifiées seront mises à jour. (Et si rien n'a changé, alors aucune mise à jour ne sera envoyée du tout).

## Travail avec les graphiques

### Résolution de l'identité

Comme indiqué ci-dessus, EF Core peut suivre une seule instance d'une entité avec une valeur de clé primaire donnée. Lorsque vous travaillez avec des graphiques, le graphique doit idéalement être créé de sorte que cet invariant est géré, et le contexte doit être utilisé pour une seule unité de travail. Si le graphique contient des doublons, il sera nécessaire de traiter le graphique avant de l'envoyer à EF pour consolider les instances multiples en une seule. Cela peut être difficile lorsque les instances ont des valeurs et relations en conflit. La consolidation des doublons doit donc être effectuée dès que possible dans le pipeline de votre application afin d'éviter la résolution des conflits.

### Toutes les entités nouvelles/toutes les entités existantes

Un exemple d'utilisation des graphiques est l'insertion ou la mise à jour d'un blog ainsi que de sa collection de

billets associés. Si toutes les entités dans le graphique doivent être insérées, ou toutes doivent être mises à jour, le processus est identique à celui décrit ci-dessus pour les entités uniques. Par exemple, un graphique de blogs et publications créé comme suit :

```
var blog = new Blog
{
    Url = "http://sample.com",
    Posts = new List<Post>
    {
        new Post {Title = "Post 1"},
        new Post {Title = "Post 2"},
    }
};
```

peut être inséré comme suit :

```
public static void InsertGraph(DbContext context, object rootEntity)
{
    context.Add(rootEntity);
    context.SaveChanges();
}
```

L'appel à Add marque le blog et tous les billets à insérer.

De même, si toutes les entités dans un graphique doivent être mises à jour, alors Update peut être utilisé :

```
public static void UpdateGraph(DbContext context, object rootEntity)
{
    context.Update(rootEntity);
    context.SaveChanges();
}
```

Le blog et tous ses billets seront marqués pour être mis à jour.

### Combinaison d'entités nouvelles et existantes

Avec les clés générées automatiquement, Update est de nouveau utilisable à la fois pour les insertions et pour les mises à jour, même si le graphique contient un mélange d'entités qui nécessitent l'insertion et d'autres qui nécessitent la mise à jour :

```
public static void InsertOrUpdateGraph(DbContext context, object rootEntity)
{
    context.Update(rootEntity);
    context.SaveChanges();
}
```

Update marque une entité dans le graphique, le blog ou un billet, pour insertion si elle ne dispose pas d'un ensemble clé-valeur, tandis que toutes les autres entités sont marquées pour mise à jour.

Comme précédemment, lorsque vous n'utilisez pas les clés générées automatiquement, vous pouvez utiliser une requête et un traitement :

```

public static void InsertOrUpdateGraph(BloggingContext context, Blog blog)
{
    var existingBlog = context.Blogs
        .Include(b => b.Posts)
        .FirstOrDefault(b => b.BlogId == blog.BlogId);

    if (existingBlog == null)
    {
        context.Add(blog);
    }
    else
    {
        context.Entry(existingBlog).CurrentValues.SetValues(blog);
        foreach (var post in blog.Posts)
        {
            var existingPost = existingBlog.Posts
                .FirstOrDefault(p => p.PostId == post.PostId);

            if (existingPost == null)
            {
                existingBlog.Posts.Add(post);
            }
            else
            {
                context.Entry(existingPost).CurrentValues.SetValues(post);
            }
        }
    }

    context.SaveChanges();
}

```

## Gestion des suppressions

La suppression peut être compliquée à gérer, car souvent l'absence d'une entité signifie qu'elle doit être supprimée. Une façon de gérer cela consiste à utiliser des « suppressions récupérables » par exemple en marquant l'entité comme supprimée plutôt que la supprimer réellement. Les suppressions s'apparentent alors à des mises à jour. Les suppressions récupérables peuvent être implémentées à l'aide de [filtres de requête](#).

Pour les vraies suppressions, il est courant d'utiliser une extension du modèle de requête pour effectuer ce qui est essentiellement une comparaison de graphique. Exemple :

```

public static void InsertUpdateOrDeleteGraph(BloggingContext context, Blog blog)
{
    var existingBlog = context.Blogs
        .Include(b => b.Posts)
        .FirstOrDefault(b => b.BlogId == blog.BlogId);

    if (existingBlog == null)
    {
        context.Add(blog);
    }
    else
    {
        context.Entry(existingBlog).CurrentValues.SetValues(blog);
        foreach (var post in blog.Posts)
        {
            var existingPost = existingBlog.Posts
                .FirstOrDefault(p => p.PostId == post.PostId);

            if (existingPost == null)
            {
                existingBlog.Posts.Add(post);
            }
            else
            {
                context.Entry(existingPost).CurrentValues.SetValues(post);
            }
        }

        foreach (var post in existingBlog.Posts)
        {
            if (!blog.Posts.Any(p => p.PostId == post.PostId))
            {
                context.Remove(post);
            }
        }
    }

    context.SaveChanges();
}

```

## TrackGraph

En interne, Add, Attach et Update utilisent la traversée de graphique en déterminant pour chaque entité si elle doit être marquée comme Added (à insérer), Modified (à mettre à jour), Unchanged (ne rien faire), ou Deleted (à supprimer). Ce mécanisme est exposé via l'API TrackGraph. Par exemple, supposons que, lorsque le client envoie un graphique d'entités, il définit certains indicateurs sur chaque entité indiquant comment elle doit être gérée.

TrackGraph peut ensuite être utilisé pour traiter cet indicateur :

```
public static void SaveAnnotatedGraph(DbContext context, object rootEntity)
{
    context.ChangeTracker.TrackGraph(
        rootEntity,
        n =>
    {
        var entity = (EntityBase)n.Entry.Entity;
        n.Entry.State = entity isNew
            ? EntityState.Added
            : entity.IsChanged
                ? EntityState.Modified
                : entity.IsDeleted
                    ? EntityState.Deleted
                    : EntityState.Unchanged;
    });
    context.SaveChanges();
}
```

Les indicateurs sont uniquement affichés dans le cadre de l'entité pour simplifier l'exemple. En général, les indicateurs feraient partie d'un DTO ou d'un autre état inclus dans la demande.

# Définition de valeurs explicites pour les propriétés générées

07/11/2019 • 6 minutes to read • [Edit Online](#)

Une propriété générée est une propriété dont la valeur est générée (soit par la base de données soit par EF) lorsque l'entité est ajoutée ou mise à jour. Pour plus d'informations, consultez [Propriétés générées](#).

Il peut arriver que vous souhaitiez définir une valeur explicite pour une propriété générée, au lieu d'en générer une.

## TIP

Vous pouvez afficher cet [exemple](#) sur GitHub.

## Le modèle

Le modèle utilisé dans cet article contient une seule entité `Employee`.

```
public class Employee
{
    public int EmployeeId { get; set; }
    public string Name { get; set; }
    public DateTime EmploymentStarted { get; set; }
    public int Salary { get; set; }
    public DateTime? LastPayRaise { get; set; }
}
```

## Enregistrement d'une valeur explicite lors de l'ajout

La propriété `Employee.EmploymentStarted` est configurée pour avoir des valeurs générées par la base de données pour les nouvelles entités (avec une valeur par défaut).

```
modelBuilder.Entity<Employee>()
    .Property(b => b.EmploymentStarted)
    .HasDefaultValueSql("CONVERT(date, GETDATE())");
```

Le code suivant insère deux employés dans la base de données.

- Pour le premier, aucune valeur n'est attribuée à la propriété `Employee.EmploymentStarted`, elle reste donc définie sur la valeur par défaut de CLR pour `DateTime`.
- Pour le deuxième, nous avons défini une valeur explicite de `1-Jan-2000`.

```

using (var context = new EmployeeContext())
{
    context.Employees.Add(new Employee { Name = "John Doe" });
    context.Employees.Add(new Employee { Name = "Jane Doe", EmploymentStarted = new DateTime(2000, 1, 1) });
    context.SaveChanges();

    foreach (var employee in context.Employees)
    {
        Console.WriteLine(employee.EmployeeId + ": " + employee.Name + ", " + employee.EmploymentStarted);
    }
}

```

La sortie indique que la base de données a généré une valeur pour le premier employé et utilisé notre valeur explicite pour le deuxième.

```

1: John Doe, 1/26/2017 12:00:00 AM
2: Jane Doe, 1/1/2000 12:00:00 AM

```

### Valeurs explicites dans les colonnes IDENTITY de SQL Server

Par convention la propriété `Employee.EmployeeId` est une colonne `IDENTITY` générée par le magasin.

Pour la plupart des cas, l'approche illustrée ci-dessus fonctionne pour les propriétés de clé. Toutefois, pour insérer des valeurs explicites dans une colonne `IDENTITY` SQL Server, vous devez activer manuellement `IDENTITY_INSERT` avant d'appeler `SaveChanges()`.

#### NOTE

Nous avons une [demande de fonctionnalité](#) dans notre backlog pour faire cela automatiquement dans le fournisseur SQL Server.

```

using (var context = new EmployeeContext())
{
    context.Employees.Add(new Employee { EmployeeId = 100, Name = "John Doe" });
    context.Employees.Add(new Employee { EmployeeId = 101, Name = "Jane Doe" });

    context.Database.OpenConnection();
    try
    {
        context.Database.ExecuteSqlRaw("SET IDENTITY_INSERT dbo.Employees ON");
        context.SaveChanges();
        context.Database.ExecuteSqlRaw("SET IDENTITY_INSERT dbo.Employees OFF");
    }
    finally
    {
        context.Database.CloseConnection();
    }

    foreach (var employee in context.Employees)
    {
        Console.WriteLine(employee.EmployeeId + ": " + employee.Name);
    }
}

```

La sortie indique que les ID fournis ont été enregistrés dans la base de données.

```
100: John Doe  
101: Jane Doe
```

## Définition d'une valeur explicite pendant une mise à jour

La propriété `Employee.LastPayRaise` est configurée pour avoir des valeurs générées par la base de données lors des mises à jour.

```
modelBuilder.Entity<Employee>()  
    .Property(b => b.LastPayRaise)  
    .ValueGeneratedOnAddOrUpdate();  
  
modelBuilder.Entity<Employee>()  
    .Property(b => b.LastPayRaise)  
    .Metadata.SetAfterSaveBehavior(PropertySaveBehavior.Ignore);
```

### NOTE

Par défaut, EF Core lève une exception si vous essayez d'enregistrer une valeur explicite pour une propriété qui est configurée pour être générée pendant la mise à jour. Pour éviter ce problème, vous devez descendre au niveau de l'API de bas niveau et définir `AfterSaveBehavior` (comme indiqué ci-dessus).

### NOTE

**Modifications dans EF Core 2.0 :** dans les versions précédentes, le comportement après enregistrement était contrôlé par le biais de l'indicateur `IsReadOnlyAfterSave`. Cet indicateur est obsolète et a été remplacé par `AfterSaveBehavior`.

Il existe également un déclencheur dans la base de données pour générer des valeurs pour la colonne `LastPayRaise` pendant les opérations `UPDATE`.

```

CREATE TRIGGER [dbo].[Employees_UPDATE] ON [dbo].[Employees]
    AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF ((SELECT TRIGGER_NESTLEVEL()) > 1) RETURN;

    IF UPDATE(Salary) AND NOT Update(LastPayRaise)
    BEGIN
        DECLARE @Id INT
        DECLARE @OldSalary INT
        DECLARE @NewSalary INT

        SELECT @Id = INSERTED.EmployeeId, @NewSalary = Salary
        FROM INSERTED

        SELECT @OldSalary = Salary
        FROM deleted

        IF @NewSalary > @OldSalary
        BEGIN
            UPDATE dbo.Employees
            SET LastPayRaise = CONVERT(date, GETDATE())
            WHERE EmployeeId = @Id
        END
    END
END

```

Le code suivant augmente le salaire des deux employés dans la base de données.

- Pour le premier, aucune valeur n'est attribuée à la propriété `Employee.LastPayRaise`, elle reste donc définie sur null.
- Pour le second, nous avons défini une valeur explicite d'une semaine plus tôt (réactivation de l'augmentation de salaire).

```

using (var context = new EmployeeContext())
{
    var john = context.Employees.Single(e => e.Name == "John Doe");
    john.Salary = 200;

    var jane = context.Employees.Single(e => e.Name == "Jane Doe");
    jane.Salary = 200;
    jane.LastPayRaise = DateTime.Today.AddDays(-7);

    context.SaveChanges();

    foreach (var employee in context.Employees)
    {
        Console.WriteLine(employee.EmployeeId + ": " + employee.Name + ", " + employee.LastPayRaise);
    }
}

```

La sortie indique que la base de données a généré une valeur pour le premier employé et utilisé notre valeur explicite pour le deuxième.

```

1: John Doe, 1/26/2017 12:00:00 AM
2: Jane Doe, 1/19/2017 12:00:00 AM

```

# Implémentations de .NET prises en charge par EF Core

10/01/2020 • 4 minutes to read • [Edit Online](#)

Nous souhaitons qu'EF Core soit disponible pour les développeurs sur toutes les implémentations de .NET modernes et nous travaillons toujours à cet objectif. Si la prise en charge d'EF Core sur .NET Core fait l'objet de tests automatisés et fonctionne correctement dans de nombreuses applications, Mono, Xamarin et UWP posent des problèmes.

## Vue d'ensemble

Le tableau suivant fournit des conseils pour chaque implémentation de .NET :

| EF CORE                        | 2.1        | 3.0                  | 3.1        |
|--------------------------------|------------|----------------------|------------|
| .NET Standard                  | 2.0        | 2.1                  | 2.0        |
| .NET Core                      | 2.0        | 3.0                  | 2.0        |
| .NET Framework <sup>(1)</sup>  | 4.7.2      | (non pris en charge) | 4.7.2      |
| Mono                           | 5,4        | 6.4                  | 5,4        |
| Xamarin.iOS <sup>(2)</sup>     | 10.14      | 12.16                | 10.14      |
| Xamarin.Android <sup>(2)</sup> | 8.0        | 10.0                 | 8.0        |
| UWP <sup>(3)</sup>             | 10.0.16299 | TBD                  | 10.0.16299 |
| Unity <sup>(4)</sup>           | 2018.1     | TBD                  | 2018.1     |

<sup>(1)</sup> Consultez la section [.NET Framework](#) ci-dessous.

<sup>(2)</sup> La présence de problèmes et de limitations connues avec Xamarin peut entraîner un dysfonctionnement de certaines applications développées à l'aide d'EF Core. Consultez la liste des [problèmes actifs](#) pour connaître les solutions de contournement.

<sup>(3)</sup> EF Core 2.0.1 et versions ultérieures recommandées. Installez le [package .NET Core UWP 6.x](#). Consultez la section [Plateforme Windows universelle](#) de cet article.

<sup>4</sup> Il existe des problèmes et des limitations connus avec Unity. Consultez la liste des [problèmes actifs](#).

## .NET Framework

Vous devez peut-être modifier les applications qui ciblent le .NET Framework pour qu'elles fonctionnent avec les bibliothèques .NET Standard :

Modifiez le fichier projet et vérifiez que l'entrée suivante apparaît dans le groupe de propriétés initiales :

```
<AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
```

Pour les projets de test, vérifiez également que l'entrée suivante est présente :

```
<GenerateBindingRedirectsOutputType>true</GenerateBindingRedirectsOutputType>
```

Si vous voulez utiliser une ancienne version de Visual Studio, vous devez [mettre à niveau le client NuGet vers la version 3.6.0](#) pour utiliser les bibliothèques .NET Standard 2.0.

Nous vous recommandons également d'effectuer une migration depuis NuGet packages.config vers PackageReference, si possible. Ajoutez la propriété suivante à votre fichier projet :

```
<RestoreProjectStyle>PackageReference</RestoreProjectStyle>
```

## Plateforme Windows universelle

Les versions antérieures d'EF Core et de .NET UWP présentaient de nombreux problèmes de compatibilité, notamment avec les applications compilées avec la chaîne d'outils .NET Native. La nouvelle version de .NET UWP prend en charge .NET Standard 2.0 et contient .NET Native 2.0 qui résout la plupart des problèmes de compatibilité signalés précédemment. EF Core 2.0.1 a été testé de manière plus approfondie avec UWP, mais les tests ne sont pas automatisés.

Avec EF Core sur UWP :

- Pour optimiser les performances des requêtes, évitez les types anonymes dans les requêtes LINQ. Une application UWP doit être générée avec .NET Native pour pouvoir être déployée dans le magasin d'applications. Les requêtes avec des types anonymes ont des performances inférieures sur .NET Native.
- Pour optimiser les performances `SaveChanges()`, utilisez `ChangeTrackingStrategy.ChangingAndChangedNotifications` et implémentez `INotifyPropertyChanged`, `INotifyPropertyChanging` et `INotifyCollectionChanged` dans vos types d'entité.

## Signaler des problèmes

Pour toute combinaison ne fonctionnant pas comme prévu, nous vous invitons à signaler les nouveaux problèmes dans le [système de suivi des problèmes EF Core](#). Pour les problèmes spécifiques à Xamarin, utilisez le système de suivi des problèmes pour [Xamarin.Android](#) ou [Xamarin.iOS](#).

# Fournisseurs de bases de données

10/01/2020 • 6 minutes to read

Entity Framework Core peut accéder à différentes bases de données par le biais de bibliothèques plug-in appelées fournisseurs de base de données.

## Fournisseurs actuels

### IMPORTANT

Les fournisseurs EF Core sont créés à partir de différentes sources. Les fournisseurs ne sont pas tous gérés dans le cadre du [projet Entity Framework Core](#). Quand vous envisagez un fournisseur, veillez à évaluer la qualité, la gestion des licences, la prise en charge, etc., pour vérifier qu'il répond à vos besoins. Veillez également à passer en revue la documentation de chaque fournisseur pour obtenir des informations détaillées sur la compatibilité des versions.

### IMPORTANT

Les fournisseurs EF Core fonctionnent généralement avec des versions mineures, mais pas avec des versions majeures. Par exemple, un fournisseur publié pour EF Core 2.1 doit fonctionner avec EF Core 2.2, mais il ne fonctionnera pas avec EF Core 3.0.

| PACKAGE NUGET   | MOTEURS DE BASE DE DONNÉES PRIS EN CHARGE | CHARGÉ DE MAINTENANCE / FOURNISSEUR        | REMARQUES / EXIGENCES | CRÉÉ POUR LA VERSION | LIENS UTILES           |
|---|---|--|-----------------------|----------------------|------------------------|
| <a href="#">Microsoft.EntityFrameworkCore.SqlServer</a> | SQL Server 2012 et ultérieur              | <a href="#">Projet EF Core (Microsoft)</a> |                       | 3.1                  | <a href="#">docs</a>   |
| <a href="#">Microsoft.EntityFrameworkCore.SQLite</a>    | SQLite 3.7 et ultérieur                   | <a href="#">Projet EF Core (Microsoft)</a> |                       | 3.1                  | <a href="#">docs</a>   |
| <a href="#">Microsoft.EntityFrameworkCore.InMemory</a>  | Fournisseur en mémoire EF Core            | <a href="#">Projet EF Core (Microsoft)</a> | Limitations           | 3.1                  | <a href="#">docs</a>   |
| <a href="#">Microsoft.EntityFrameworkCore.Cosmos</a>    | API SQL Azure Cosmos DB                   | <a href="#">Projet EF Core (Microsoft)</a> |                       | 3.1                  | <a href="#">docs</a>   |
| <a href="#">Npgsql.EntityFrameworkCore.PostgreSQL</a>   | PostgreSQL                                | Équipe de développement Npgsql             |                       | 3.1                  | <a href="#">docs</a>   |
| <a href="#">Pomelo.EntityFrameworkCore.MySql</a>        | MySQL, MariaDB                            | <a href="#">Projet Pomelo Foundation</a>   |                       | 3.1                  | <a href="#">readme</a> |

| PACKAGE NUGET                              | MOTEURS DE BASE DE DONNÉES PRIS EN CHARGE       | CHARGÉ DE MAINTENANCE / FOURNISSEUR | REMARQUES / EXIGENCES       | CRÉÉ POUR LA VERSION | LIENS UTILES             |
|--|---|-------------------------------------|-----------------------------|----------------------|--------------------------|
| Devart.Data.MySql.EntityFrameworkCore      | MySQL 5 et ultérieur                            | DevArt                              | Payé                        | 3.0                  | <a href="#">docs</a>     |
| Devart.Data.Oracle.EntityFrameworkCore     | Oracle DB 9.2.0.4 et versions ultérieures       | DevArt                              | Payé                        | 3.0                  | <a href="#">docs</a>     |
| Devart.Data.PostgreSql.EntityFrameworkCore | PostgreSQL 8.0 et ultérieur                     | DevArt                              | Payé                        | 3.0                  | <a href="#">docs</a>     |
| Devart.Data.SQLite.EntityFrameworkCore     | SQLite 3 et ultérieur                           | DevArt                              | Payé                        | 3.0                  | <a href="#">docs</a>     |
| FileContextCore                            | Stocke les données dans des fichiers            | Morris Janatzek                     | À des fins de développement | 3.0                  | <a href="#">readme</a>   |
| EntityFrameworkCore.Jet                    | Fichiers Microsoft Access                       | Bubi                                | .NET Framework              | 2.2                  | <a href="#">readme</a>   |
| EntityFrameworkCore.SqlServerCompact35     | SQL Server Compact 3.5                          | Erik Ejlskov Jensen                 | .NET Framework              | 2.2                  | <a href="#">wiki</a>     |
| EntityFrameworkCore.SqlServerCompact40     | SQL Server Compact 4,0                          | Erik Ejlskov Jensen                 | .NET Framework              | 2.2                  | <a href="#">wiki</a>     |
| FirebirdSql.EntityFrameworkCore.Firebird   | Firebird 2.5 et 3.x                             | Jiří Činčura                        |                             | 2.2                  | <a href="#">docs</a>     |
| Teradata.EntityFrameworkCore               | Teradata Database 16.10 et versions ultérieures | Teradata                            | Version préliminaire        | 2.2                  | <a href="#">site web</a> |
| EntityFrameworkCore.FirebirdSql            | Firebird 2.5 et 3.x                             | Rafael Almeida                      |                             | 2.1                  | <a href="#">wiki</a>     |
| EntityFrameworkCore.OpenEdge               | Progress OpenEdge                               | Alex Wiese                          |                             | 2.1                  | <a href="#">readme</a>   |
| MySQL.Data.EntityFrameworkCore             | MySQL   | Projet MySQL (Oracle)               |                             | 2.1                  | <a href="#">docs</a>     |
| Oracle.EntityFrameworkCore                 | Oracle DB 11.2 et versions ultérieures          | Oracle                              |                             | 2.1                  | <a href="#">site web</a> |
| IBM.EntityFrameworkCore                    | Db2, Informix                                   | IBM                                 | Version Windows             | 2.0                  | <a href="#">blog</a>     |

| PACKAGE NUGET                    | MOTEURS DE BASE DE DONNÉES PRIS EN CHARGE | CHARGÉ DE MAINTENANCE / FOURNISSEUR | REMARQUES / EXIGENCES           | CRÉÉ POUR LA VERSION | LIENS UTILES           |
|----------------------------------|---|-------------------------------------|---------------------------------|----------------------|------------------------|
| IBM.EntityFrameworkCore-Inx      | Db2, Informix                             | IBM                                 | Version Linux                   | 2.0                  | <a href="#">blog</a>   |
| IBM.EntityFrameworkCore-osx      | Db2, Informix                             | IBM                                 | Version macOS                   | 2.0                  | <a href="#">blog</a>   |
| Pomelo.EntityFrameworkCore.MyCat | Serveur MyCAT                             | Projet Pomelo Foundation            | Version préliminaire uniquement | 1.1                  | <a href="#">readme</a> |

## Ajout d'un fournisseur de base de données à votre application

La plupart des fournisseurs de base de données pour EF Core sont distribués sous la forme de packages NuGet et peuvent être installés comme suit :

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet add package provider_package_name
```

Une fois les packages installés, configurez le fournisseur dans votre `DbContext` : soit dans la méthode `OnConfiguring`, soit dans la méthode `AddDbContext` si vous utilisez un conteneur d'injection de dépendance. Par exemple, la ligne suivante configure le fournisseur SQL Server avec la chaîne de connexion passée :

```
optionsBuilder.UseSqlServer(
    "Server=(localdb)\mssqllocaldb;Database=MyDatabase;Trusted_Connection=True;");
```

Les fournisseurs de base de données peuvent étendre Core EF pour activer des fonctionnalités propres à des bases de données spécifiques. Certains concepts, communs à la plupart des bases de données, sont inclus dans les principaux composants d'EF Core. Ces concepts comprennent l'expression des requêtes dans LINQ, les transactions et le suivi des changements apportés aux objets une fois qu'ils sont chargés à partir de la base de données. Certains concepts sont propres à un fournisseur. Par exemple, le fournisseur SQL Server vous permet de [configurer des tables à mémoire optimisée](#) (fonctionnalité spécifique à SQL Server). D'autres concepts sont propres à une classe de fournisseurs. Par exemple, les fournisseurs EF Core pour les bases de données relationnelles reposent sur la bibliothèque `Microsoft.EntityFrameworkCore.Relational` commune, qui fournit des API pour configurer les mappages de tables et de colonnes, les contraintes de clé étrangère, etc. Les fournisseurs sont généralement distribués sous la forme de packages NuGet.

### IMPORTANT

Quand une nouvelle version corrective d'EF Core est publiée, des mises à jour au package `Microsoft.EntityFrameworkCore.Relational` sont souvent incluses. Quand vous ajoutez un fournisseur de base de données relationnelle, ce package devient une dépendance transitive de votre application. Mais de nombreux fournisseurs sont publiés indépendamment d'EF et peuvent ne pas être mis à jour pour dépendre de la nouvelle version corrective de ce package. Pour que vous receviez tous les correctifs, nous vous recommandons d'ajouter la version corrective de `Microsoft.EntityFrameworkCore.Relational` comme dépendance directe de votre application.

# Fournisseur de base de données EF Core Microsoft SQL Server

08/01/2020 • 2 minutes to read

Ce fournisseur de base de données permet d'utiliser Entity Framework Core avec Microsoft SQL Server (notamment Azure SQL Database). Il est géré dans le cadre du [projet Entity Framework Core](#).

## Installez

Installez le [package NuGet Microsoft.EntityFrameworkCore.SqlServer](#).

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

### NOTE

Depuis la version 3.0.0, le fournisseur référence Microsoft.Data.SqlClient (les versions précédentes dépendaient de System.Data.SqlClient). Si votre projet dépend directement de SqlConnection, assurez-vous qu'il fait référence au package Microsoft.Data.SqlClient.

## Moteurs de base de données pris en charge

- Microsoft SQL Server (versions 2012 et suivantes)

# Prise en charge des tables optimisées en mémoire dans SQL Server fournisseur de base de données EF Core

05/12/2019 • 2 minutes to read

Les [tables mémoire optimisées](#) sont une fonctionnalité de SQL Server où la table entière réside en mémoire. Une deuxième copie des données de la table est conservée sur le disque, mais uniquement pour la durabilité. Les données des tables mémoire optimisées sont uniquement lues à partir du disque lors de la récupération d'une base de données. Par exemple, après le redémarrage d'un serveur.

## Configuration d'une table optimisée en mémoire

Vous pouvez spécifier que la table à laquelle est mappée une entité a une mémoire optimisée. Lorsque vous utilisez EF Core pour créer et gérer une base de données basée sur votre modèle (avec [migrations](#) ou [EnsureCreated](#)), une table optimisée en mémoire est créée pour ces entités.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Blog>().IsMemoryOptimized();
}
```

# Spécification des options de Azure SQL Database

05/12/2019 • 2 minutes to read

## NOTE

Cette API est nouvelle dans EF Core 3,1.

Azure SQL Database propose [diverses options de tarification](#) qui sont généralement configurées via le portail Azure. Toutefois, si vous gérez le schéma à l'aide de [EF Core migrations](#), vous pouvez spécifier les options souhaitées dans le modèle lui-même.

Vous pouvez spécifier le niveau de service de la base de données (édition) à l'aide de [HasServiceTier](#):

```
modelBuilder.HasServiceTier("BusinessCritical");
```

Vous pouvez spécifier la taille maximale de la base de données à l'aide de [HasDatabaseMaxSize](#):

```
modelBuilder.HasDatabaseMaxSize("2 GB");
```

Vous pouvez spécifier le niveau de performance de la base de données (SERVICE\_OBJECTIVE) à l'aide de [HasPerformanceLevel](#):

```
modelBuilder.HasPerformanceLevel("BC_Gen4_1");
```

Utilisez [HasPerformanceLevelSql](#) pour configurer le pool élastique, car la valeur n'est pas un littéral de chaîne :

```
modelBuilder.HasPerformanceLevelSql("ELASTIC_POOL ( name = myelasticpool )");
```

## TIP

Vous pouvez trouver toutes les valeurs prises en charge dans la [documentation ALTER DATABASE](#).

# Fournisseur de base de données EF Core SQLite

08/01/2020 • 2 minutes to read

Ce fournisseur de base de données permet d'utiliser Entity Framework Core avec SQLite. Il est géré dans le cadre du [projet Entity Framework Core](#).

## Installez

Installez le [package NuGet Microsoft.EntityFrameworkCore.Sqlite](#).

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet add package Microsoft.EntityFrameworkCore.Sqlite
```

## Moteurs de base de données pris en charge

- SQLite (versions 3.7 et suivantes)

## Limitations

Consultez [Limitations de SQLite](#) pour connaître certaines limitations importantes du fournisseur SQLite.

# Limitations du fournisseur de base de données SQLite EF Core

11/10/2019 • 4 minutes to read

Le fournisseur SQLite a un certain nombre de limitations de migrations. La plupart de ces limitations résultent de limitations dans le moteur de base de données SQLite sous-jacent et ne sont pas spécifiques à EF.

## Limitations de modélisation

La bibliothèque relationnelle commune (partagée par Entity Framework fournisseurs de bases de données relationnelles) définit des API pour les concepts de modélisation communs à la plupart des moteurs de base de données relationnelles. Quelques-uns de ces concepts ne sont pas pris en charge par le fournisseur SQLite.

- Schémas
- Séquences
- Colonnes calculées

## Limitations des requêtes

SQLite ne prend pas en charge en mode natif les types de données suivants. EF Core pouvez lire et écrire des valeurs de ces types, et l'interrogation de l'égalité (`where e.Property == value`) est également prise en charge. Toutefois, d'autres opérations, comme la comparaison et le tri, nécessitent une évaluation sur le client.

- DateTimeOffset
- Decimal
- TimeSpan
- UInt64

Au lieu de `DateTimeOffset`, nous vous recommandons d'utiliser des valeurs `DateTime`. Lors de la gestion de plusieurs fuseaux horaires, nous vous recommandons de convertir les valeurs en temps UTC avant de les enregistrer, puis de les reconvertir dans le fuseau horaire approprié.

Le type `Decimal` offre un niveau élevé de précision. Toutefois, si vous n'avez pas besoin de ce niveau de précision, nous vous recommandons d'utiliser à la place un double. Vous pouvez utiliser un [convertisseur de valeur](#) pour continuer à utiliser `Decimal` dans vos classes.

```
modelBuilder.Entity<MyEntity>()
    .Property(e => e.DecimalProperty)
    .HasConversion<double>();
```

## Limitations des migrations

Le moteur de base de données SQLite ne prend pas en charge un certain nombre d'opérations de schéma prises en charge par la plupart des autres bases de données relationnelles. Si vous tentez d'appliquer l'une des opérations non prises en charge à une base de données SQLite, une `NotSupportedException` sera levée.

| OPÉRATION            | GÉRÉ?                | VERSION REQUISE |
|----------------------|----------------------|-----------------|
| AddColumn            | ✓                    | 1.0             |
| AddForeignKey        | ✗                    |                 |
| AddPrimaryKey        | ✗                    |                 |
| AddUniqueConstraint  | ✗                    |                 |
| AlterColumn          | ✗                    |                 |
| CreateIndex          | ✓                    | 1.0             |
| CreateTable          | ✓                    | 1.0             |
| DropColumn           | ✗                    |                 |
| DropForeignKey       | ✗                    |                 |
| DroplIndex           | ✓                    | 1.0             |
| DropPrimaryKey       | ✗                    |                 |
| DropTable            | ✓                    | 1.0             |
| DropUniqueConstraint | ✗                    |                 |
| RenameColumn         | ✓                    | 2.2.2           |
| RenameIndex          | ✓                    | 2.1             |
| RenameTable          | ✓                    | 1.0             |
| EnsureSchema         | ✓ (aucune opération) | 2.0             |
| DropSchema           | ✓ (aucune opération) | 2.0             |
| Insert               | ✓                    | 2.0             |
| Mise à jour          | ✓                    | 2.0             |
| Supprimer            | ✓                    | 2.0             |

## Solution de contournement des limitations des migrations

Vous pouvez contourner certaines de ces limitations en écrivant manuellement du code dans vos migrations pour effectuer une reconstruction de table. Une reconstruction de la table implique la modification du nom de la table existante, la création d'une nouvelle table, la copie des données vers la nouvelle table et la suppression de l'ancienne table. Vous devrez utiliser la méthode `Sql(string)` pour effectuer certaines de ces étapes.

Pour plus d'informations, consultez [création d'autres types de modifications de schéma de table](#) dans la documentation sqlite.

À l'avenir, EF peut prendre en charge certaines de ces opérations à l'aide de l'approche de reconstruction de table en coulisses. Vous pouvez [suivre cette fonctionnalité sur notre projet GitHub](#).

# Fournisseur Azure Cosmos DB EF Core

10/01/2020 • 8 minutes to read

## NOTE

Ce fournisseur est nouveau dans EF Core 3.0.

Ce fournisseur de base de données permet d'utiliser Entity Framework Core avec Azure Cosmos DB. Il est géré dans le cadre du [projet Entity Framework Core](#).

Avant de lire cette section, il est fortement recommandé de vous familiariser avec la documentation d'[Azure Cosmos DB](#).

## NOTE

Ce fournisseur fonctionne uniquement avec l'API SQL d'Azure Cosmos DB.

## Installez

Installez le [package NuGet Microsoft.EntityFrameworkCore.Cosmos](#).

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet add package Microsoft.EntityFrameworkCore.Cosmos
```

## Prise en main

### TIP

Vous pouvez afficher cet [exemple sur GitHub](#).

Comme pour les autres fournisseurs, la première étape consiste à appeler [UseCosmos](#) :

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder.UseCosmos(
        "https://localhost:8081",
        "C2y6yDjf5/R+ob0N8A7Cgv30VRDJWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGgyPMbIZnqyMsEcaGQy67XIw/Jw==",
        databaseName: "OrdersDB");
```

### WARNING

Le point de terminaison et la clé sont codés en dur ici par souci de simplicité, mais dans une application de production, ils doivent être stockés de manière sécurisée

Dans cet exemple, `Order` est une entité simple avec une référence au [type détenu](#) `StreetAddress`.

```
public class Order
{
    public int Id { get; set; }
    public int? TrackingNumber { get; set; }
    public string PartitionKey { get; set; }
    public StreetAddress ShippingAddress { get; set; }
}
```

```
public class StreetAddress
{
    public string Street { get; set; }
    public string City { get; set; }
}
```

L'enregistrement et l'interrogation des données suivent le modèle EF normal :

```
using (var context = new OrderContext())
{
    await context.Database.EnsureDeletedAsync();
    await context.Database.EnsureCreatedAsync();

    context.Add(new Order
    {
        Id = 1,
        ShippingAddress = new StreetAddress { City = "London", Street = "221 B Baker St" },
        PartitionKey = "1"
    });

    await context.SaveChangesAsync();
}

using (var context = new OrderContext())
{
    var order = await context.Orders.FirstAsync();
    Console.WriteLine($"First order will ship to: {order.ShippingAddress.Street},
{order.ShippingAddress.City}");
    Console.WriteLine();
}
```

### IMPORTANT

L'appel de `EnsureCreatedAsync` est nécessaire pour créer les conteneurs requis et insérer les [données initiales](#) si elles sont présentes dans le modèle. Toutefois, `EnsureCreatedAsync` ne doit être appelé qu'au cours du déploiement, pas pendant le fonctionnement normal, car cela peut entraîner des problèmes de performances.

## Personnalisation du modèle propre à Cosmos

Par défaut, tous les types d'entité sont mappés au même conteneur, nommé d'après le contexte dérivé (`"OrderContext"` dans le cas présent). Pour changer le nom de conteneur par défaut, utilisez [HasDefaultContainer](#) :

```
modelBuilder.HasDefaultContainer("Store");
```

Pour mapper un type d'entité à un autre conteneur, utilisez [ToContainer](#) :

```
modelBuilder.Entity<Order>()
    .ToContainer("Orders");
```

Pour identifier le type d'entité représenté par un élément donné, EF Core ajoute une valeur de discriminateur même en l'absence de type d'entité dérivé. Le nom et la valeur du discriminateur [peuvent être changés](#).

Si aucun autre type d'entité ne sera jamais stocké dans le même conteneur, le discriminant peut être supprimé en appelant [HasNoDiscriminator](#) :

```
modelBuilder.Entity<Order>()
    .HasNoDiscriminator();
```

## Clés de partition

Par défaut EF Core crée des conteneurs avec la clé de partition définie sur `"__partitionKey"` sans fournir de valeur pour celle-ci lors de l'insertion d'éléments. Mais pour tirer pleinement parti des fonctionnalités de performances d'Azure Cosmos, une [clé de partition soigneusement sélectionnée](#) doit être utilisée. Elle peut être configurée en appelant [HasPartitionKey](#) :

```
modelBuilder.Entity<Order>()
    .HasPartitionKey(o => o.PartitionKey);
```

### NOTE

La propriété de clé de partition peut être de n'importe quel type tant qu'elle est [convertie en chaîne](#).

Une fois configurée, la propriété de clé de partition doit toujours avoir une valeur non Null. Lors de l'émission d'une requête, une condition peut être ajoutée pour qu'elle soit à partition unique.

```
using (var context = new OrderContext())
{
    context.Add(new Order
    {
        Id = 2,
        ShippingAddress = new StreetAddress { City = "New York", Street = "11 Wall Street" },
        PartitionKey = "2"
    });

    await context.SaveChangesAsync();
}

using (var context = new OrderContext())
{
    var order = await context.Orders.Where(p => p.PartitionKey == "2").LastAsync();
    Console.WriteLine("Last order will ship to: ");
    Console.WriteLine($"{order.ShippingAddress.Street}, {order.ShippingAddress.City}");
    Console.WriteLine();
}
```

## Entités incorporées

Pour Cosmos, les entités détenues sont incorporées dans le même élément que le propriétaire. Pour changer un nom de propriété, utilisez [ToJsonProperty](#) :

```

modelBuilder.Entity<Order>().OwnsOne(
    o => o.ShippingAddress,
    sa =>
    {
        sa.ToJsonProperty("Address");
        sa.Property(p => p.Street).ToJsonProperty("ShipsToStreet");
        sa.Property(p => p.City).ToJsonProperty("ShipsToCity");
    });
}

```

Avec cette configuration, la commande dans l'exemple ci-dessus est stockée comme suit :

```

{
    "Id": 1,
    "PartitionKey": "1",
    "TrackingNumber": null,
    "id": "1",
    "Address": {
        "ShipsToCity": "London",
        "ShipsToStreet": "221 B Baker St"
    },
    "_rid": "6QEKAM+BOOABAAAAAAA==",
    "_self": "dbs/6QEKA=/colls/6QEKAM+BOOA=/docs/6QEKAM+BOOABAAAAAAA==/",
    "_etag": "\"00000000-0000-0000-683c-692e763901d5\"",
    "_attachments": "attachments/",
    "_ts": 1568163674
}

```

Les collections d'entités détenues sont également incorporées. Pour l'exemple suivant, nous allons utiliser la classe

`Distributor` avec la collection `StreetAddress` :

```

public class Distributor
{
    public int Id { get; set; }
    public ICollection<StreetAddress> ShippingCenters { get; set; }
}

```

Les entités détenues n'ont pas besoin de fournir des valeurs de clés explicites pour être stockées :

```

var distributor = new Distributor
{
    Id = 1,
    ShippingCenters = new HashSet<StreetAddress> {
        new StreetAddress { City = "Phoenix", Street = "500 S 48th Street" },
        new StreetAddress { City = "Anaheim", Street = "5650 Dolly Ave" }
    }
};

using (var context = new OrderContext())
{
    context.Add(distributor);

    await context.SaveChangesAsync();
}

```

Elles sont conservées de cette façon :

```
{
    "Id": 1,
    "Discriminator": "Distributor",
    "id": "Distributor|1",
    "ShippingCenters": [
        {
            "City": "Phoenix",
            "Street": "500 S 48th Street"
        },
        {
            "City": "Anaheim",
            "Street": "5650 Dolly Ave"
        }
    ],
    "_rid": "6QEKANzISj0BAAAAAAA==",
    "_self": "dbs/6QEKA==/colls/6QEKANzISj0=/docs/6QEKANzISj0BAAAAAAA==/",
    "_etag": "\"00000000-0000-0000-683c-7b2b439701d5\"",
    "_attachments": "attachments/",
    "_ts": 1568163705
}
```

En interne, EF Core doit toujours avoir des valeurs de clé uniques pour toutes les entités suivies. La clé primaire créée par défaut pour les collections de types détenus se compose des propriétés de clé étrangère qui pointent vers le propriétaire et d'une propriété `int` correspondant à l'index dans le tableau JSON. Pour récupérer ces valeurs, vous pouvez utiliser l'API d'entrée :

```
using (var context = new OrderContext())
{
    var firstDistributor = await context.Distributors.FirstAsync();
    Console.WriteLine($"Number of shipping centers: {firstDistributor.ShippingCenters.Count}");

    var addressEntry = context.Entry(firstDistributor.ShippingCenters.First());
    var addressPKProperties = addressEntry.Metadata.FindPrimaryKey().Properties;

    Console.WriteLine($"First shipping center PK:
({addressEntry.Property(addressPKProperties[0].Name).CurrentValue},
{addressEntry.Property(addressPKProperties[1].Name).CurrentValue})");
    Console.WriteLine();
}
```

#### TIP

Quand cela est nécessaire, la clé primaire par défaut pour les types d'entités détenus peut être changée, mais les valeurs de clé doivent être fournies explicitement.

## Utilisation d'entités déconnectées

Chaque élément doit avoir une valeur `id` unique pour la clé de partition donnée. Par défaut, EF Core génère la valeur en concaténant les valeurs du discriminateur et de la clé primaire, au moyen du caractère « | » en guise de séparateur. Les valeurs de clé sont générées uniquement quand une entité passe à l'état `Added`. Cela peut poser un problème lors de l'[attachement d'entités](#) si elles n'ont pas de propriété `id` sur le type .NET pour stocker la valeur.

Pour contourner cette limitation, il est possible de créer et de définir la valeur `id` manuellement, ou bien de marquer l'entité comme étant ajoutée, puis de lui affecter l'état souhaité :

```

using (var context = new OrderContext())
{
    var distributorEntry = context.Add(distributor);
    distributorEntry.State = EntityState.Unchanged;

    distributor.ShippingCenters.Remove(distributor.ShippingCenters.Last());

    await context.SaveChangesAsync();
}

using (var context = new OrderContext())
{
    var firstDistributor = await context.Distributors.FirstAsync();
    Console.WriteLine($"Number of shipping centers is now: {firstDistributor.ShippingCenters.Count}");

    var distributorEntry = context.Entry(firstDistributor);
    var idProperty = distributorEntry.Property<string>("id");
    Console.WriteLine($"The distributor 'id' is: {idProperty.CurrentValue}");
}

```

Voici le code JSON résultant :

```
{
    "Id": 1,
    "Discriminator": "Distributor",
    "id": "Distributor|1",
    "ShippingCenters": [
        {
            "City": "Phoenix",
            "Street": "500 S 48th Street"
        }
    ],
    "_rid": "JBwtAN8oNYEBAAAAAAAA==",
    "_self": "dbs/JBwtAA==/colls/JBwtAN8oNYE=/docs/JBwtAN8oNYEBAAAAAAAA==/",
    "_etag": "\"00000000-0000-0000-9377-d7a1ae7c01d5\"",
    "_attachments": "attachments/",
    "_ts": 1572917100
}
```

# Utilisation de données non structurées dans EF Core fournisseur Azure Cosmos DB

07/11/2019 • 3 minutes to read

EF Core a été conçu pour faciliter l'utilisation des données qui suivent un schéma défini dans le modèle. Toutefois, l'une des forces de Azure Cosmos DB est la flexibilité dans la forme des données stockées.

## Accès au JSON brut

Il est possible d'accéder aux propriétés qui ne sont pas suivies par EF Core via une propriété spéciale dans l'**État Shadow** nommé `"__j0bject"` qui contient un `J0bject` représentant les données reçues du magasin et les données qui seront stockées :

```
using (var context = new OrderContext())
{
    await context.Database.EnsureDeletedAsync();
    await context.Database.EnsureCreatedAsync();

    var order = new Order
    {
        Id = 1,
        ShippingAddress = new StreetAddress { City = "London", Street = "221 B Baker St" },
        PartitionKey = "1"
    };

    context.Add(order);

    await context.SaveChangesAsync();
}

using (var context = new OrderContext())
{
    var order = await context.Orders.FirstAsync();
    var orderEntry = context.Entry(order);

    var jsonProperty = orderEntry.Property<J0bject>("__j0bject");
    jsonProperty.CurrentValue["BillingAddress"] = "Clarence House";

    orderEntry.State = EntityState.Modified;

    await context.SaveChangesAsync();
}

using (var context = new OrderContext())
{
    var order = await context.Orders.FirstAsync();
    var orderEntry = context.Entry(order);
    var jsonProperty = orderEntry.Property<J0bject>("__j0bject");

    Console.WriteLine($"First order will be billed to: {jsonProperty.CurrentValue["BillingAddress"]}");
}
```

```
{  
    "Id": 1,  
    "PartitionKey": "1",  
    "TrackingNumber": null,  
    "id": "1",  
    "Address": {  
        "ShipsToCity": "London",  
        "ShipsToStreet": "221 B Baker St"  
    },  
    "_rid": "eLMaAK8TzkIBAAAAAAA==",  
    "_self": "dbs/eLMaAA==/colls/eLMaAK8TzkI=/docs/eLMaAK8TzkIBAAAAAAA==/",  
    "_etag": "\"00000000-0000-0000-683e-0a12bf8d01d5\"",  
    "_attachments": "attachments/",  
    "BillingAddress": "Clarence House",  
    "_ts": 1568164374  
}
```

#### WARNING

La propriété `"__jObject"` fait partie de l'infrastructure EF Core et ne doit être utilisée qu'en dernier recours, car elle est susceptible d'avoir un comportement différent dans les futures versions.

#### NOTE

Les modifications apportées à l'entité remplacent les valeurs stockées dans `"__jObject"` lors de la `SaveChanges`.

## Utilisation de CosmosClient

Pour découpler complètement de EF Core récupérez l'objet [CosmosClient](#) qui fait [partie du kit de développement logiciel \(SDK\) Azure Cosmos DB](#) à partir de `DbContext` :

```
using (var context = new OrderContext())  
{  
    var cosmosClient = context.Database.GetCosmosClient();  
    var database = cosmosClient.GetDatabase("OrdersDB");  
    var container = database.GetContainer("Orders");  
  
    var resultSet = container.GetItemQueryIterator<JObject>(new QueryDefinition("select * from o"));  
    var order = (await resultSet.ReadNextAsync()).First();  
  
    Console.WriteLine($"First order JSON: {order}");  
  
    order.Remove("TrackingNumber");  
  
    await container.ReplaceItemAsync(order, order["id"].ToString());  
}
```

## Valeurs de propriété manquantes

Dans l'exemple précédent, nous avons supprimé la propriété `"TrackingNumber"` de la commande. En raison du fonctionnement de l'indexation dans Cosmos DB, les requêtes qui font référence à la propriété manquante ailleurs que dans la projection peuvent retourner des résultats inattendus. Exemple :

```
using (var context = new OrderContext())
{
    var orders = await context.Orders.ToListAsync();
    var sortedOrders = await context.Orders.OrderBy(o => o.TrackingNumber).ToListAsync();

    Console.WriteLine($"Number of orders: {orders.Count}");
    Console.WriteLine($"Number of sorted orders: {sortedOrders.Count}");
}
```

En fait, la requête triée ne retourne aucun résultat. Cela signifie qu'il faut veiller à toujours remplir les propriétés mappées par EF Core quand vous travaillez directement avec le magasin.

#### NOTE

Ce comportement peut changer dans les versions ultérieures de Cosmos. Par exemple, si la stratégie d'indexation définit l'index composite {ID/ ? ASC, TrackingNumber/ ? ASC}, une requête dont le a'ORDER BY c.Id ASC, c. discriminateur

**ASC**'retourne des éléments pour lesquels la propriété "TrackingNumber" est manquante.

# Limitations des fournisseurs de EF Core Azure Cosmos DB

07/11/2019 • 2 minutes to read

Le fournisseur Cosmos présente un certain nombre de limitations. La plupart de ces limitations résultent de limitations dans le moteur de base de données Cosmos sous-jacent et ne sont pas spécifiques à EF. Mais la plupart n'ont pas encore été implémentées.

## Limitations temporaires

- Même s'il existe un seul type d'entité sans héritage mappé à un conteneur, il possède toujours une propriété de discriminateur.
- Les types d'entités avec des clés de partition ne fonctionnent pas correctement dans certains scénarios
- les appels de `Include` ne sont pas pris en charge
- les appels de `Join` ne sont pas pris en charge

## Limitations du kit de développement logiciel Azure Cosmos DB

- Seules les méthodes Async sont fournies

### WARNING

Étant donné qu'il n'existe aucune version de synchronisation des méthodes de bas niveau EF Core s'appuie sur, la fonctionnalité correspondante est actuellement implémentée en appelant `.Wait()` sur le `Task` retourné. Cela signifie que l'utilisation de méthodes telles que `SaveChanges` ou `ToList` au lieu de leurs équivalents asynchrones peut entraîner un interblocage dans votre application

## Limitations de Azure Cosmos DB

Vous pouvez voir la vue d'ensemble complète des [fonctionnalités prises en charge par Azure Cosmos DB](#). il s'agit des différences les plus notables par rapport à une base de données relationnelle :

- Les transactions initiées par le client ne sont pas prises en charge
- Certaines requêtes de plusieurs partitions ne sont pas prises en charge ou sont beaucoup plus lentes en fonction des opérateurs impliqués

# Fournisseur de base de données InMemory EF Core

08/01/2020 • 2 minutes to read

Ce fournisseur de base de données permet d'utiliser Entity Framework Core avec une base de données en mémoire. Ceci peut s'avérer utile pour les tests, bien que le fournisseur SQLite en mode en mémoire constitue peut-être un remplacement de test plus approprié pour les bases de données relationnelles. Il est géré dans le cadre du [projet Entity Framework Core](#).

## Installez

Installez le [package NuGet Microsoft.EntityFrameworkCore.InMemory](#).

- [CLI .NET Core](#)
- [Visual Studio](#)

```
dotnet add package Microsoft.EntityFrameworkCore.InMemory
```

## Bien démarrer

Les ressources suivantes vous permettent de commencer à utiliser ce fournisseur.

- [Test avec InMemory](#)
- [Exemple de tests d'application UnicornStore](#)

## Moteurs de base de données pris en charge

Base de données en mémoire in-process (conçue uniquement à des fins de test)

# Écriture d'un fournisseur de base de données

07/11/2019 • 3 minutes to read

Pour plus d'informations sur l'écriture d'un fournisseur de base de données Entity Framework Core, consultez pour [ce faire, vous devez écrire un fournisseur EF Core](#) par [Arthur Vickers](#).

## NOTE

Ces publications n'ont pas été mises à jour depuis EF Core 1.1 et des modifications significatives ont été apportées depuis le moment où le [problème 681](#) est le suivi des mises à jour de cette documentation.

La base de code EF Core est open source et contient plusieurs fournisseurs de bases de données qui peuvent être utilisés comme référence. Vous pouvez trouver le code source à <https://github.com/aspnet/EntityFrameworkCore>. Il peut également être utile d'examiner le code pour les fournisseurs tiers couramment utilisés, tels que [npgsql](#), [Pomelo MySQL](#) et [SQL Server Compact](#). En particulier, ces projets sont configurés pour étendre et exécuter des tests fonctionnels que nous publions sur NuGet. Ce type d'installation est fortement recommandé.

## Tenir à jour les modifications apportées au fournisseur

À compter du travail après la version 2.1, nous avons créé un [Journal des modifications](#) qui peut nécessiter des modifications correspondantes dans le code du fournisseur. Cela est destiné à faciliter la mise à jour d'un fournisseur existant pour qu'il fonctionne avec une nouvelle version de EF Core.

Avant le 2.1, nous avons utilisé les étiquettes `providers-beware` et `providers-fyi` sur nos problèmes GitHub et les demandes de tirage (pull requests) à des fins similaires. Nous continuons à utiliser ces étiquettes sur les problèmes pour donner une indication que les éléments de travail d'une version donnée peuvent également nécessiter un travail dans les fournisseurs. Une étiquette `providers-beware` signifie généralement que l'implémentation d'un élément de travail peut arrêter des fournisseurs, tandis qu'une `providers-fyi` étiquette signifie généralement que les fournisseurs ne seront pas rompus, mais le code devra peut-être être modifié de toute manière, par exemple, pour activer les nouvelles fonctionnalités.

## Attribution d'un nom suggéré aux fournisseurs tiers

Nous vous suggérons d'utiliser le nom suivant pour les packages NuGet. Cela est cohérent avec les noms des packages fournis par l'équipe EF Core.

```
<Optional project/company name>.EntityFrameworkCore.<Database engine name>
```

Exemple :

- `Microsoft.EntityFrameworkCore.SqlServer`
- `Npgsql.EntityFrameworkCore.PostgreSQL`
- `EntityFrameworkCore.SqlServerCompact40`

# Modifications ayant un impact sur le fournisseur

07/11/2019 • 10 minutes to read

Cette page contient des liens vers les requêtes de tirage effectuées sur le EF Core référentiel qui peuvent nécessiter la réaction des auteurs d'autres fournisseurs de bases de données. L'objectif est de fournir un point de départ pour les auteurs de fournisseurs de bases de données tiers existants lors de la mise à jour de leur fournisseur vers une nouvelle version.

Nous commençons ce journal avec des modifications de 2,1 à 2,2. Avant le 2,1, nous avons utilisé les étiquettes `providers-beware` et `providers-fyi` sur nos problèmes et les demandes de tirage (pull requests).

## 2,2---> 3,0

Notez que la plupart des [modifications avec rupture au niveau](#) de l'application auront également un impact sur les fournisseurs.

- <https://github.com/aspnet/EntityFrameworkCore/pull/14022>
  - Suppression des API obsolètes et des surcharges de paramètre facultatives réduites
  - Suppression de DatabaseColumn. GetUnderlyingStoreType ()
- <https://github.com/aspnet/EntityFrameworkCore/pull/14589>
  - Suppression des API obsolètes
- <https://github.com/aspnet/EntityFrameworkCore/pull/15044>
  - Les sous-classes de CharTypeMapping peuvent avoir été rompues en raison des changements de comportement requis pour la résolution de quelques bogues dans l'implémentation de base.
- <https://github.com/aspnet/EntityFrameworkCore/pull/15090>
  - Ajout d'une classe de base pour IDatabaseModelFactory et mise à jour de celle-ci pour utiliser un objet paramètre afin d'atténuer les ruptures ultérieures.
- <https://github.com/aspnet/EntityFrameworkCore/pull/15123>
  - Objets de paramètres utilisés dans MigrationsSqlGenerator pour atténuer les ruptures ultérieures.
- <https://github.com/aspnet/EntityFrameworkCore/pull/14972>
  - La configuration explicite des niveaux de journal nécessitait des modifications des API que les fournisseurs peuvent utiliser. Plus précisément, si les fournisseurs utilisent l'infrastructure de journalisation directement, cette modification peut arrêter cette utilisation. En outre, les fournisseurs qui utilisent l'infrastructure (qui sera publique) devront dériver de `LoggingDefinitions` ou `RelationalLoggingDefinitions`. Pour obtenir des exemples, consultez les fournisseurs SQL Server et en mémoire.
- <https://github.com/aspnet/EntityFrameworkCore/pull/15091>
  - Les chaînes de ressources de base, relationnelles et d'abstractions sont désormais publiques.
  - `CoreLoggerExtensions` et `RelationalLoggerExtensions` sont désormais publics. Les fournisseurs doivent utiliser ces API lors de l'enregistrement des événements définis au niveau principal ou relationnel. N'accédez pas directement aux ressources de journalisation ; ils sont toujours internes.
  - `IRawSqlCommandBuilder` est passé d'un service Singleton à un service étendu
  - `IMigrationsSqlGenerator` est passé d'un service Singleton à un service étendu
- <https://github.com/aspnet/EntityFrameworkCore/pull/14706>
  - L'infrastructure de création de commandes relationnelles a été rendue publique afin de pouvoir être utilisée en toute sécurité par les fournisseurs et refactoriser légèrement.
- <https://github.com/aspnet/EntityFrameworkCore/pull/14733>

- `ILazyLoader` est passé d'un service étendu à un service temporaire
- <https://github.com/aspnet/EntityFrameworkCore/pull/14610>
  - `IUpdateSqlGenerator` est passé d'un service délimité à un service Singleton
  - En outre, `ISingletonUpdateSqlGenerator` a été supprimé
- <https://github.com/aspnet/EntityFrameworkCore/pull/15067>
  - Un grand nombre de code interne utilisé par les fournisseurs est devenu public
  - Il ne doit plus être nécessaire pour référencer `IndentedStringBuilder`, car il a été pris en compte dans les emplacements qui l'ont exposée
  - Les utilisations de `NonCapturingLazyInitializer` doivent être remplacées par `LazyInitializer` à partir de la bibliothèque de classes de bibliothèque
- <https://github.com/aspnet/EntityFrameworkCore/pull/14608>
  - Cette modification est entièrement traitée dans le document sur les modifications avec rupture d'application. Pour les fournisseurs, cela peut avoir un impact plus important dans la mesure où le test d'EF Core peut souvent entraîner ce problème, de sorte que l'infrastructure de test a été modifiée pour réduire le risque.
- <https://github.com/aspnet/EntityFrameworkCore/issues/13961>
  - `EntityMaterializerSource` a été simplifié
- <https://github.com/aspnet/EntityFrameworkCore/pull/14895>
  - La traduction StartsWith a été modifiée de telle sorte que les fournisseurs peuvent souhaiter réagir
- <https://github.com/aspnet/EntityFrameworkCore/pull/15168>
  - Les services d'ensemble de conventions ont changé. Les fournisseurs doivent maintenant hériter de « ProviderConventionSet » ou « RelationalConventionSet ».
  - Les personnalisations peuvent être ajoutées par le biais de `IConventionSetCustomizer` services, mais elles sont destinées à être utilisées par d'autres extensions, et non par des fournisseurs.
  - Les conventions utilisées au moment de l'exécution doivent être résolues à partir de `IConventionSetBuilder`.
- <https://github.com/aspnet/EntityFrameworkCore/pull/15288>
  - L'amorçage des données a été refactorisé dans une API publique pour éviter d'avoir à utiliser des types internes. Cela ne doit avoir d'impact que sur les fournisseurs non relationnels, car l'amorçage est géré par la classe relationnelle de base pour tous les fournisseurs relationnels.

2,1---> 2,2

### **Modifications de test uniquement**

- <https://github.com/aspnet/EntityFrameworkCore/pull/12057>-autoriser le séparateurs SQL personnalisable dans les tests
  - Test des modifications qui autorisent les comparaisons de virgule flottante non strictes dans `BuiltInDataTypesTestBase`
  - Testez les modifications qui permettent de réutiliser les tests de requête avec différents séparateurs SQL
- <https://github.com/aspnet/EntityFrameworkCore/pull/12072>-ajouter des tests DbFunction aux tests de spécification relationnelle
  - Ces tests peuvent être exécutés sur tous les fournisseurs de base de données
- <https://github.com/aspnet/EntityFrameworkCore/pull/12362>-nettoyage du test asynchrone
  - Supprimer les appels `Wait`, asynchrones inutiles et renommer certaines méthodes de test
- <https://github.com/aspnet/EntityFrameworkCore/pull/12666>-unification de l'infrastructure de test d'enregistrement
  - Ajout de `CreateListLoggerFactory` et suppression de l'infrastructure de journalisation précédente, qui nécessitera des fournisseurs utilisant ces tests pour réagir

- <https://github.com/aspnet/EntityFrameworkCore/pull/12500>-exécuter d'autres tests de requête à la fois de façon synchrone et asynchrone
  - Les noms et la factorisation des tests ont changé, ce qui nécessite que les fournisseurs utilisant ces tests réagissent
- <https://github.com/aspnet/EntityFrameworkCore/pull/12766>-attribution d'un nouveau nom aux navigations dans le modèle ComplexNavigations
  - Les fournisseurs utilisant ces tests peuvent être amenés à réagir
- <https://github.com/aspnet/EntityFrameworkCore/pull/12141>-retourne le contexte au pool au lieu de le supprimer dans les tests fonctionnels
  - Cette modification comprend une certaine refactorisation de test qui peut nécessiter des fournisseurs pour réagir

## **Modifications du code du produit et du test**

- <https://github.com/aspnet/EntityFrameworkCore/pull/12109>-consolider les méthodes RelationalTypeMapping. Clone
  - Les modifications apportées à 2,1 dans le RelationalTypeMapping sont autorisées pour une simplification dans les classes dérivées. Nous ne pensons pas que cela était une rupture pour les fournisseurs, mais les fournisseurs peuvent tirer parti de cette modification dans leurs classes de mappage de type dérivé.
- requêtes avec balises <https://github.com/aspnet/EntityFrameworkCore/pull/12069> ou nommées
  - Ajoute l'infrastructure pour le balisage des requêtes LINQ et l'affichage de ces balises sous forme de commentaires dans le SQL. Cela peut nécessiter que les fournisseurs réagissent dans la génération SQL.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13115>-prendre en charge les données spatiales via NTS
  - Autorise l'inscription des mappages de types et des traducteurs de membres en dehors du fournisseur
  - Les fournisseurs doivent appeler base. FindMapping () dans son implémentation ITypeMappingSource pour qu'elle fonctionne
  - Suivez ce modèle pour ajouter la prise en charge spatiale à votre fournisseur qui est cohérente entre les fournisseurs.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13199>-ajouter un débogage amélioré pour la création du fournisseur de services
  - Permet à DbContextOptionsExtensions d'implémenter une nouvelle interface qui peut aider les utilisateurs à comprendre pourquoi le fournisseur de services internes est recréé.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13289>-ajoute l'API CanConnect pour une utilisation par les contrôles d'intégrité
  - Ce PR ajoute le concept de `CanConnect` qui sera utilisé par les contrôles d'intégrité ASP.NET Core pour déterminer si la base de données est disponible. Par défaut, l'implémentation relationnelle appelle simplement `Exist`, mais les fournisseurs peuvent implémenter une autre valeur si nécessaire. Les fournisseurs non relationnels devront implémenter la nouvelle API pour que le contrôle d'intégrité soit utilisable.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13306>-mettre à jour les RelationalTypeMapping de base pour ne pas définir la taille DbParameter
  - Arrête la définition de la taille par défaut, car elle peut provoquer une troncation. Les fournisseurs doivent peut-être ajouter leur propre logique si la taille doit être définie.
- <https://github.com/aspnet/EntityFrameworkCore/pull/13372>-RevEng : toujours spécifier le type de colonne pour les colonnes décimales
  - Configurez toujours le type de colonne pour les colonnes décimales dans le code de génération de modèles automatique plutôt que par Convention.
  - Les fournisseurs ne doivent pas exiger de modifications à leur extrémité.

- <https://github.com/aspnet/EntityFrameworkCore/pull/13469>-ajoute CaseExpression pour la génération d'expressions de cas SQL
- <https://github.com/aspnet/EntityFrameworkCore/pull/13648> : ajoute la possibilité de spécifier des mappages de type sur SqlFunctionExpression pour améliorer l'inférence de type de magasin des arguments et des résultats.

# Outils et extensions EF Core

10/01/2020 • 9 minutes to read • [Edit Online](#)

Ces outils et extensions fournissent des fonctionnalités supplémentaires pour Entity Framework Core 2.1 et versions ultérieures.

## IMPORTANT

Les extensions sont générées par des sources diverses et ne sont pas gérées dans le cadre du projet Entity Framework Core. Quand vous envisagez une extension tierce, veillez à évaluer ses caractéristiques, notamment en termes de qualité, de gestion des licences, de compatibilité et de prise en charge, pour vérifier qu'elle répond à vos besoins. En particulier, une extension créée pour une version antérieure d'EF Core peut nécessiter une mise à jour avant de fonctionner avec les versions les plus récentes.

# Outils

## **LLBLGen Pro**

LLBLGen Pro est une solution de modélisation d'entités qui prend en charge Entity Framework et Entity Framework Core. Il vous permet de définir facilement votre modèle d'entités et de le mapper à votre base de données, en utilisant Database First ou Model First : vous pouvez donc commencer à écrire des requêtes immédiatement. Pour EF Core : 2.

[Site web](#)

## **Devart Entity Developer**

Entity Developer est un concepteur ORM pour ADO.NET Entity Framework, NHibernate, LinqConnect, Telerik Data Access et LINQ to SQL. Il prend en charge la conception visuelle de modèles EF Core, selon une approche Model First ou Database First, et la génération de code C# ou Visual Basic. Pour EF Core : 2.

[Site web](#)

## **EF Core Power Tools**

EF Core Power Tools est une extension Visual Studio qui expose différentes tâches EF Core au moment du design dans une interface utilisateur simple. Elle inclut l'ingénierie à rebours des classes DbContext et d'entité à partir de bases de données existantes et des [packages DAC SQL Server](#), la gestion des migrations de base de données et les visualisations de modèles. Pour EF Core : 2, 3.

[Wiki GitHub](#)

## **Entity Framework Visual Editor**

Entity Framework Visual Editor est une extension de Visual Studio qui ajoute un concepteur ORM permettant de concevoir visuellement des classes EF 6 et EF Core. Le code étant généré à l'aide de modèles T4, il peut être personnalisé pour répondre à tous les besoins. Il prend en charge les associations d'héritage, unidirectionnelles et bidirectionnelles, les énumérations ainsi que la possibilité de colorer le code de vos classes et d'ajouter des blocs de texte pour expliquer les parties potentiellement obscures de votre conception. Pour EF Core : 2.

[Marketplace](#)

## **CatFactory**

CatFactory est un moteur de génération de modèles automatique pour .NET Core qui peut automatiser la génération de classes DbContext, d'entités, de configurations de mappage et de classes de dépôts à partir d'une

base de données SQL Server. Pour EF Core : 2.

[Dépôt GitHub](#)

### Générateur Entity Framework Core de LoreSoft

Entity Framework Core Generator (efg) est un outil CLI .NET Core qui peut générer des modèles EF Core à partir d'une base de données existante, comme `dotnet ef dbcontext scaffold`, mais qui prend également en charge la [regénération](#) de code safe à travers le remplacement de région ou l'analyse des fichiers de mappage. Cet outil prend en charge la génération de code de mappeur d'objet, de validation et de modèles de vue. Pour EF Core : 2.

[Tutoriel Documentation](#)

## Extensions

### Microsoft.EntityFrameworkCore.AutoHistory

Bibliothèque de plug-ins qui permet l'enregistrement automatique des changements de données effectués par EF Core dans une table d'historique. Pour EF Core : 2.

[Dépôt GitHub](#)

### EFSecondLevelCache.Core

Extension qui permet de stocker les résultats de requêtes EF Core dans un cache de second niveau afin que les exécutions ultérieures des mêmes requêtes puissent récupérer les données directement à partir du cache sans avoir à accéder à la base de données. Pour EF Core : 2.

[Dépôt GitHub](#)

### Geco

Geco (Generator Console) est un générateur de code simple basé sur un projet de console qui s'exécute sur .NET Core et qui utilise des chaînes interpolées C# pour la génération de code. Geco inclut un générateur de modèle inverse pour EF Core avec prise en charge de la pluralisation, de la singularisation et des modèles modifiables. Il fournit également un générateur de script de données de départ, un exécuteur de scripts et un nettoyeur de base de données. Pour EF Core : 2.

[Dépôt GitHub](#)

### EntityFrameworkCore.Scaffolding.Handlebars

Permet de personnaliser des classes ayant fait l'objet d'une rétroconception à partir d'une base de données existante à l'aide de la chaîne d'outils Entity Framework Core avec des modèles Handlebars. Pour EF Core : 2, 3.

[Dépôt GitHub](#)

### NeinLinq.EntityFrameworkCore

NeinLinq étend les fonctionnalités des fournisseurs LINQ comme Entity Framework pour permettre la réutilisation de fonctions, la réécriture des requêtes et la génération de requêtes dynamiques à l'aide de sélecteurs et de prédictifs traduisibles. Pour EF Core : 2, 3.

[Dépôt GitHub](#)

### Microsoft.EntityFrameworkCore.UnitOfWork

Plug-in pour Microsoft.EntityFrameworkCore prenant en charge un dépôt, les modèles d'unités de travail et les bases de données multiples avec des transactions distribuées. Pour EF Core : 2.

[Dépôt GitHub](#)

### EFCore.BulkExtensions

Extensions EF Core pour les opérations en bloc (Insert, Update, Delete). Pour EF Core : 2, 3.

[Dépôt GitHub](#)

### **Bricelam.EntityFrameworkCore.Pluralizer**

Ajoute la pluralisation au moment du design. Pour EF Core : 2.

[Dépôt GitHub](#)

### **Toolbelt.EntityFrameworkCore.IndexAttribute**

Reprise de l'attribut [Index] (avec extension pour la création de modèles). Pour EF Core : 2, 3.

[Dépôt GitHub](#)

### **EfCore.InMemoryHelpers**

Fournit un wrapper autour du fournisseur de base de données In-Memory EF Core. Il fonctionne alors plus comme un fournisseur relationnel. Pour EF Core : 2.

[Dépôt GitHub](#)

### **EFCore.TemporalSupport**

Implémentation de la prise en charge temporelle. Pour EF Core : 2.

[Dépôt GitHub](#)

### **EfCoreTemporalTable**

Exécutez facilement des requêtes temporelles sur votre base de données favorite à l'aide de méthodes d'extension introduites : `AsTemporalAll()`, `AsTemporalAsOf(date)`, `AsTemporalFrom(startDate, endDate)`,  
`AsTemporalBetween(startDate, endDate)`, `AsTemporalContained(startDate, endDate)`. Pour EF Core : 3.

[Dépôt GitHub](#)

### **EFCore.TimeTraveler**

Autorisez des requêtes Entity Framework Core complètes sur [la table d'historique temporelle SQL Server](#) à l'aide du code, des entités et des mappages EF Core que vous avez déjà définis. Voyagez dans le temps en encapsulant votre code dans `using (TemporalQuery.AsOf(targetDateTime)) { ... }`. Pour EF Core : 3.

[Dépôt GitHub](#)

### **EntityFrameworkCore.TemporalTables**

Bibliothèque d'extensions pour Entity Framework Core qui permet aux développeurs qui utilisent SQL Server de se servir facilement des tables temporelles. Pour EF Core : 2.

[Dépôt GitHub](#)

### **EntityFrameworkCore.Cacheable**

Cache des requêtes de second niveau hautes performances. Pour EF Core : 2.

[Dépôt GitHub](#)

### **Entity Framework Plus**

Étend votre DbContext avec des fonctionnalités telles que : Include Filter (Inclure le filtre), Auditing (Audit), Caching (Mise en cache), Query Future (Requête ultérieure), Batch Delete (Suppression par lot), Batch Update (Mise à jour par lot), et bien plus encore. Pour EF Core : 2, 3.

[Site web Référentiel GitHub](#)

### **Extensions d'Entity Framework**

Étend votre DbContext avec des opérations en bloc hautes performances : BulkSaveChanges, BulkInsert, BulkUpdate, BulkDelete, BulkMerge, et bien plus encore. Pour EF Core : 2, 3.

[Site web](#)

# Informations de référence sur les outils Entity Framework Core

20/09/2019 • 2 minutes to read

Les outils Entity Framework Core aident à effectuer les tâches de développement au moment du design. Ils servent principalement à gérer les migrations et à structurer un `DbContext` et des types d'entité en reconstituant la logique du schéma d'une base de données.

- Les [outils de la Console du Gestionnaire de package EF Core](#) s'exécutent dans la [Console du Gestionnaire de package](#) dans Visual Studio.
- Les [outils de l'interface de ligne de commande \(CLI\) EF Core .NET](#) sont une extension multiplateforme des [outils CLI .NET Core](#). Ces outils nécessitent un projet de SDK .NET Core (dont le fichier projet contient `Sdk="Microsoft.NET.Sdk"` ou une ligne similaire).

Les deux outils exposent les mêmes fonctionnalités. Si vous développez dans Visual Studio, nous vous recommandons d'utiliser les outils de la **Console du Gestionnaire de package**, car ils procurent une expérience plus intégrée.

## Étapes suivantes

- [Informations de référence sur les outils de la Console du Gestionnaire de package EF Core](#)
- [Informations de référence sur les outils CLI .NET EF Core](#)

# Référence des outils de Entity Framework Core-console du gestionnaire de package dans Visual Studio

26/11/2019 • 17 minutes to read

Les outils de la console du gestionnaire de package (PMC) pour Entity Framework Core effectuent des tâches de développement au moment du Design. Par exemple, ils créent des [migrations](#), appliquent des migrations et génèrent du code pour un modèle basé sur une base de données existante. Les commandes s'exécutent dans Visual Studio à l'aide de la [console du gestionnaire de package](#). Ces outils fonctionnent avec les projets .NET Framework et .NET Core.

Si vous n'utilisez pas Visual Studio, nous vous recommandons d'utiliser les [outils en ligne de commande EF Core](#) à la place. Les outils CLI sont inter-plateformes et s'exécutent à l'intérieur d'une invite de commandes.

## Installation des outils

Les procédures d'installation et de mise à jour des outils diffèrent entre ASP.NET Core 2.1+ et les versions antérieures ou d'autres types de projets.

### ASP.NET Core version 2.1 et versions ultérieures

Les outils sont inclus automatiquement dans un projet ASP.NET Core 2.1+, car le package `Microsoft.EntityFrameworkCore.Tools` est inclus dans le [métapackage Microsoft.AspNetCore.App](#).

Par conséquent, vous n'avez rien à faire pour installer les outils, mais vous devez effectuer les opérations suivantes :

- Restaurez les packages avant d'utiliser les outils d'un nouveau projet.
- Installez un package pour mettre à jour les outils vers une version plus récente.

Pour vous assurer que vous obtenez la version la plus récente des outils, nous vous recommandons également d'effectuer les étapes suivantes :

- Modifiez votre fichier `.csproj` et ajoutez une ligne spécifiant la dernière version du package `Microsoft.EntityFrameworkCore.Tools`. Par exemple, le fichier `.csproj` peut inclure une `ItemGroup` qui ressemble à ceci :

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.App" />
  <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="2.1.3" />
  <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="2.1.1" />
</ItemGroup>
```

Mettez à jour les outils lorsque vous recevez un message comme dans l'exemple suivant :

La version des outils de EF Core « 2.1.1-RTM-30846 » est plus ancienne que celle du runtime « 2.1.3-RTM-32065 ». Mettez à jour les outils des dernières fonctionnalités et correctifs de bogues.

Pour mettre à jour les outils :

- Installez la dernière kit SDK .NET Core.

- Mettez à jour Visual Studio vers la dernière version.
- Modifiez le fichier `.csproj` afin qu'il inclue une référence de package au package d'outils le plus récent, comme indiqué plus haut.

### Autres types de versions et de projets

Installez les outils de la console du gestionnaire de package en exécutant la commande suivante dans la **console du gestionnaire de package**:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

Mettez à jour les outils en exécutant la commande suivante dans la **console du gestionnaire de package**.

```
Update-Package Microsoft.EntityFrameworkCore.Tools
```

### Vérifier l'installation

Vérifiez que les outils sont installés en exécutant la commande suivante :

```
Get-Help about_EntityFrameworkCore
```

La sortie ressemble à ceci (il ne vous indique pas la version des outils que vous utilisez) :

```

 _/\_
---=/   \\
  __|_.  \|\
 |__||_|_|_)  \\\
 |__||_|_|\_/_| //|\\
 |__||_|_|    /  \\\\
TOPIC
about_EntityFrameworkCore

SHORT DESCRIPTION
Provides information about the Entity Framework Core Package Manager Console Tools.

<A list of available commands follows, omitted here.>
```

## Utilisation des outils

Avant d'utiliser les outils :

- Comprenez la différence entre le projet cible et le projet de démarrage.
- Découvrez comment utiliser les outils avec des bibliothèques de classes .NET Standard.
- Pour les projets ASP.NET Core, définissez l'environnement.

### Projet de démarrage et cible

Les commandes font référence à un *projet* et à un *projet de démarrage*.

- Le *projet* est également appelé *projet cible*, car il s'agit de l'emplacement où les commandes ajoutent ou suppriment des fichiers. Par défaut, le **projet par défaut** sélectionné dans la **console du gestionnaire de package** est le projet cible. Vous pouvez spécifier un projet différent comme projet cible à l'aide de l'option `--project`.
- Le *projet de démarrage* est celui que les outils génèrent et exécutent. Les outils doivent exécuter le code

de l'application au moment de la conception pour obtenir des informations sur le projet, telles que la chaîne de connexion à la base de données et la configuration du modèle. Par défaut, le **projet de démarrage** dans **Explorateur de solutions** est le projet de démarrage. Vous pouvez spécifier un projet différent comme projet de démarrage à l'aide de l'option `--startup-project`.

Le projet de démarrage et le projet cible sont souvent le même projet. Un scénario classique dans lequel il s'agit de projets distincts est le suivant :

- Le contexte de EF Core et les classes d'entité se trouvent dans une bibliothèque de classes .NET Core.
- Une application console .NET Core ou une application Web fait référence à la bibliothèque de classes.

Il est également possible de [Placer le code de migrations dans une bibliothèque de classes distincte du contexte de EF Core](#).

### Autres frameworks cibles

Les outils de la console du gestionnaire de package fonctionnent avec .NET Core ou des projets .NET Framework. Les applications qui ont le modèle EF Core dans une bibliothèque de classes .NET Standard peuvent ne pas avoir de projet .NET Core ou .NET Framework. C'est le cas, par exemple, des applications Xamarin et plateforme Windows universelle. Dans ce cas, vous pouvez créer un projet d'application console .NET Core ou .NET Framework dont l'objectif est d'agir comme projet de démarrage pour les outils. Le projet peut être un projet factice sans code réel — il n'est nécessaire que pour fournir une cible pour les outils.

Pourquoi un projet factice est-il nécessaire ? Comme mentionné précédemment, les outils doivent exécuter le code de l'application au moment de la conception. Pour ce faire, ils doivent utiliser le Runtime .NET Core ou .NET Framework. Lorsque le modèle de EF Core se trouve dans un projet qui cible .NET Core ou .NET Framework, les outils de EF Core empruntent le runtime du projet. Ils ne peuvent pas le faire si le modèle de EF Core se trouve dans une bibliothèque de classes .NET Standard. Le .NET Standard n'est pas une implémentation .NET réelle. Il s'agit d'une spécification d'un ensemble d'API que les implémentations .NET doivent prendre en charge. Par conséquent, .NET Standard n'est pas suffisant pour que les outils de EF Core exécutent le code d'application. Le projet factice que vous créez à utiliser comme projet de démarrage fournit une plateforme cible concrète dans laquelle les outils peuvent charger la bibliothèque de classes .NET Standard.

### Environnement de ASP.NET Core

Pour spécifier l'environnement pour les projets ASP.NET Core, définissez `env` :

`ASPNETCORE_ENVIRONMENT` avant d'exécuter les commandes.

## Paramètres communs

Le tableau suivant montre les paramètres qui sont communs à toutes les commandes EF Core :

| PARAMÈTRE                                   | DESCRIPTION  |
|---|--|
| <code>-Context &lt;chaîne&gt;</code>        | Classe <code>DbContext</code> à utiliser. Nom de classe uniquement ou qualifié complet avec des espaces de noms. Si ce paramètre est omis, EF Core recherche la classe de contexte. S'il existe plusieurs classes de contexte, ce paramètre est obligatoire. |
| <code>-Project &lt;chaîne&gt;</code>        | Projet cible. Si ce paramètre est omis, le <b>projet par défaut</b> pour la <b>console du gestionnaire de package</b> est utilisé comme projet cible.  |
| <code>-StartupProject &lt;chaîne&gt;</code> | Projet de démarrage. Si ce paramètre est omis, le <b>projet de démarrage</b> dans les propriétés de la <b>solution</b> est utilisé comme projet cible.   |

| PARAMÈTRE | DESCRIPTION                   |
|-----------|-------------------------------|
| -Verbose  | Affichez la sortie détaillée. |

Pour afficher des informations d'aide sur une commande, utilisez la commande `Get-Help` de PowerShell.

#### TIP

Les paramètres Context, Project et StartupProject prennent en charge l'expansion de tabulation.

## Add-Migration

Ajoute une nouvelle migration.

Paramètres :

| PARAMÈTRE           | DESCRIPTION   |
|---------------------|---|
| Nom de <chaîne>     | Nom de la migration. Il s'agit d'un paramètre positionnel qui est obligatoire.  |
| -OutputDir <chaîne> | Répertoire (et sous-espace de noms) à utiliser. Les chemins d'accès sont relatifs au répertoire du projet cible. La valeur par défaut est « migrations ». |

## Drop-Database

Supprime la base de données.

Paramètres :

| PARAMÈTRE | DESCRIPTION   |
|-----------|---|
| -WhatIf   | Affichez la base de données qui sera supprimée, mais ne la supprimez pas. |

## Get-DbContext

Obtient des informations sur un type de `DbContext`.

## Remove-Migration

Supprime la dernière migration (restaure les modifications de code qui ont été effectuées pour la migration).

Paramètres :

| PARAMÈTRE | DESCRIPTION  |
|-----------|--|
| -Force    | Rétablissement la migration (annulez les modifications qui ont été appliquées à la base de données). |

## Scaffold-DbContext

Génère du code pour une `DbContext` et des types d'entités pour une base de données. Pour que

`Scaffold-DbContext` génère un type d'entité, la table de base de données doit avoir une clé primaire.

Paramètres :

| PARAMÈTRE            | DESCRIPTION  |
|----------------------|--|
| -Connection <chaîne> | Chaîne de connexion à la base de données. Pour les projets ASP.NET Core 2.x, la valeur peut être <i>Name =&lt;nom de la chaîne de connexion&gt;</i> . Dans ce cas, le nom provient des sources de configuration qui sont configurées pour le projet. Il s'agit d'un paramètre positionnel qui est obligatoire. |
| -Provider <chaîne>   | Fournisseur à utiliser. En général, il s'agit du nom du package NuGet, par exemple : <code>Microsoft.EntityFrameworkCore.SqlServer</code> . Il s'agit d'un paramètre positionnel qui est obligatoire.  |
| -OutputDir <chaîne>  | Répertoire dans lequel placer les fichiers. Les chemins d'accès sont relatifs au répertoire du projet.   |
| -ContextDir <chaîne> | Répertoire dans lequel placer le fichier <code>DbContext</code> . Les chemins d'accès sont relatifs au répertoire du projet.   |
| -Context <chaîne>    | Nom de la classe <code>DbContext</code> à générer.   |
| -Schemas <String []> | Schémas des tables pour lesquelles générer des types d'entité. Si ce paramètre est omis, tous les schémas sont inclus.   |
| -Tables <String []>  | Tables pour lesquelles générer des types d'entité. Si ce paramètre est omis, toutes les tables sont incluses.  |
| -DataAnnotations     | Utilisez des attributs pour configurer le modèle (dans la mesure du possible). Si ce paramètre est omis, seule l'API Fluent est utilisée.  |
| -UseDatabaseNames    | Utilisez les noms de table et de colonne exactement tels qu'ils apparaissent dans la base de données. Si ce paramètre est omis, les noms de base de données sont modifiés pour être C# plus conformes aux conventions de style de nom.   |
| -Force               | Remplacer les fichiers existants.  |

Exemple :

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Exemple qui génère uniquement les tables sélectionnées et crée le contexte dans un dossier distinct avec un nom spécifié :

```
Scaffold-DbContext "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Tables "Blog","Post" -ContextDir Context -  
Context BlogContext
```

# Script-Migration

Génère un script SQL qui applique toutes les modifications d'une migration sélectionnée à une autre migration sélectionnée.

Paramètres :

| PARAMÈTRE         | DESCRIPTION  |
|-------------------|--|
| -From <chaîne>    | Début de la migration. Les migrations peuvent être identifiées par leur nom ou par leur ID. Le nombre 0 est un cas spécial qui signifie <i>avant la première migration</i> . La valeur par défaut est 0.   |
| -To <chaîne>      | Fin de la migration. La valeur par défaut est la dernière migration.   |
| -Idempotent       | Générez un script qui peut être utilisé sur une base de données lors d'une migration.  |
| -Output <String > | Fichier dans lequel écrire le résultat. Si ce paramètre est omis, le fichier est créé avec un nom généré dans le même dossier que celui dans lequel les fichiers d'exécution de l'application sont créés, par exemple :<br><i>/obj/Debug/netcoreapp2.1/ghbkztfz.SQL/</i> . |

## TIP

Les paramètres to, from et Output prennent en charge l'expansion de tabulation.

L'exemple suivant crée un script pour la migration InitialCreate, en utilisant le nom de la migration.

```
Script-Migration -To InitialCreate
```

L'exemple suivant crée un script pour toutes les migrations après la migration de InitialCreate, à l'aide de l'ID de migration.

```
Script-Migration -From 20180904195021_InitialCreate
```

# Update-Database

Met à jour la base de données jusqu'à la dernière migration ou à une migration spécifiée.

| PARAMÈTRE           | DESCRIPTION   |
|---------------------|---|
| -Migration <chaîne> | Migration cible. Les migrations peuvent être identifiées par leur nom ou par leur ID. Le nombre 0 est un cas spécial qui signifie <i>avant la première migration</i> et entraîne la restauration de toutes les migrations. Si aucune migration n'est spécifiée, la commande prend par défaut la dernière migration. |

**TIP**

Le paramètre de migration prend en charge l'expansion de tabulation.

L'exemple suivant rétablit toutes les migrations.

```
Update-Database -Migration 0
```

Les exemples suivants mettent à jour la base de données vers une migration spécifiée. Le premier utilise le nom de la migration et le second utilise l'ID de migration :

```
Update-Database -Migration InitialCreate  
Update-Database -Migration 20180904195021_InitialCreate
```

## Ressources supplémentaires

- [Migrations](#)
- [Reconstitution de la logique des produits](#)

# Informations de référence sur les outils Entity Framework Core-CLI .NET

05/12/2019 • 18 minutes to read

Les outils de l'interface de ligne de commande (CLI) pour Entity Framework Core effectuent des tâches de développement au moment du Design. Par exemple, ils créent des [migrations](#), appliquent des migrations et génèrent du code pour un modèle basé sur une base de données existante. Les commandes sont une extension de la commande `dotnet` multiplateforme, qui fait partie du [Kit SDK .NET Core](#). Ces outils fonctionnent avec les projets .NET Core.

Si vous utilisez Visual Studio, nous vous recommandons d'utiliser les outils de la [console du gestionnaire de package](#) à la place :

- Ils fonctionnent automatiquement avec le projet actif sélectionné dans la **console du gestionnaire de package** sans que vous ayez besoin de basculer manuellement les répertoires.
- Ils ouvrent automatiquement les fichiers générés par une commande une fois la commande terminée.

## Installation des outils

La procédure d'installation dépend du type et de la version du projet :

- EF Core 3.x
- ASP.NET Core version 2.1 et versions ultérieures
- EF Core 2.x
- EF Core 1.x

### EF Core 3.x

- `dotnet ef` doit être installé en tant qu'outil Global ou local. La plupart des développeurs installent `dotnet ef` comme un outil global avec la commande suivante :

```
dotnet tool install --global dotnet-ef
```

Vous pouvez également utiliser `dotnet ef` en tant qu'outil local. Pour l'utiliser en tant qu'outil local, restaurez les dépendances d'un projet qui le déclare en tant que dépendance de l'outil à l'aide d'un [fichier manifeste d'outil](#).

- Installez le [Kit SDK .NET Core 3.0](#). Le kit de développement logiciel (SDK) doit être installé même si vous disposez de la dernière version de Visual Studio.
- Installez le dernier package de [Microsoft.EntityFrameworkCore.Design](#).

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

### ASP.NET Core 2.1 +

- Installez le [Kit SDK .NET Core](#) actuel. Le kit SDK doit être installé même si vous disposez de la dernière version de Visual Studio 2017.

C'est tout ce qui est nécessaire pour ASP.NET Core 2.1 +, car le package [Microsoft.EntityFrameworkCore.Design](#) est inclus dans le [AspNetCore](#).

## EF Core 2.x (non ASP.NET Core)

Les commandes `dotnet ef` sont incluses dans le kit SDK .NET Core, mais pour activer les commandes dont vous avez besoin pour installer le package `Microsoft.EntityFrameworkCore.Design`.

- Installez le [Kit SDK .NET Core](#) actuel. Le kit de développement logiciel (SDK) doit être installé même si vous disposez de la dernière version de Visual Studio.
- Installez le dernier package `Microsoft.EntityFrameworkCore.Design` stable.

```
dotnet add package Microsoft.EntityFrameworkCore.Design
```

## EF Core 1.x

- Installez la version de kit SDK .NET Core 2.1.200. Les versions ultérieures ne sont pas compatibles avec les outils CLI pour EF Core 1.0 et 1.1.
- Configurez l'application pour utiliser la version du kit de développement logiciel (SDK) 2.1.200 en modifiant son fichier [global.json](#). Ce fichier est normalement inclus dans le répertoire de la solution (l'un au-dessus du projet).
- Modifiez le fichier projet et ajoutez `Microsoft.EntityFrameworkCore.Tools.DotNet` en tant qu'élément `DotNetCliToolReference`. Spécifiez la version 1.x la plus récente, par exemple : 1.1.6. Consultez l'exemple de fichier projet à la fin de cette section.
- Installez la dernière version 1.x du package `Microsoft.EntityFrameworkCore.Design`, par exemple :

```
dotnet add package Microsoft.EntityFrameworkCore.Design -v 1.1.6
```

Une fois les deux références de package ajoutées, le fichier projet ressemble à ceci :

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design"
      Version="1.1.6"
      PrivateAssets="All" />
  </ItemGroup>
  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
      Version="1.1.6" />
  </ItemGroup>
</Project>
```

Une référence de package avec `PrivateAssets="All"` n'est pas exposée aux projets qui font référence à ce projet. Cette restriction est particulièrement utile pour les packages qui sont généralement utilisés uniquement pendant le développement.

## Vérifier l'installation

Exécutez les commandes suivantes pour vérifier que EF Core outils CLI sont correctement installés :

```
dotnet restore
dotnet ef
```

La sortie de la commande identifie la version des outils en cours d'utilisation :

```
 _/\_
----/ \ \
| . \ \
|_||_| | ) \ \
|_||_| \/_ | //| \\
|_||_| / \\\\
Entity Framework Core .NET Command-line Tools 2.1.3-rtm-32065
<Usage documentation follows, not shown.>
```

## Utilisation des outils

Avant d'utiliser les outils, vous devrez peut-être créer un projet de démarrage ou définir l'environnement.

### Projet cible et projet de démarrage

Les commandes font référence à un *projet* et à un *projet de démarrage*.

- Le *projet* est également appelé *projet cible*, car il s'agit de l'emplacement où les commandes ajoutent ou suppriment des fichiers. Par défaut, le projet dans le répertoire actif est le projet cible. Vous pouvez spécifier un projet différent comme projet cible à l'aide de l'option `--project` .
- Le *projet de démarrage* est celui que les outils génèrent et exécutent. Les outils doivent exécuter le code de l'application au moment de la conception pour obtenir des informations sur le projet, telles que la chaîne de connexion à la base de données et la configuration du modèle. Par défaut, le projet dans le répertoire actif est le projet de démarrage. Vous pouvez spécifier un projet différent comme projet de démarrage à l'aide de l'option `--startup-project` .

Le projet de démarrage et le projet cible sont souvent le même projet. Un scénario classique dans lequel il s'agit de projets distincts est le suivant :

- Le contexte de EF Core et les classes d'entité se trouvent dans une bibliothèque de classes .NET Core.
- Une application console .NET Core ou une application Web fait référence à la bibliothèque de classes.

Il est également possible de [Placer le code de migrations dans une bibliothèque de classes distincte du contexte de EF Core](#).

### Autres frameworks cibles

Les outils CLI fonctionnent avec les projets .NET Core et les projets de .NET Framework. Les applications qui ont le modèle EF Core dans une bibliothèque de classes .NET Standard peuvent ne pas avoir de projet .NET Core ou .NET Framework. C'est le cas, par exemple, des applications Xamarin et plateforme Windows universelle. Dans ce cas, vous pouvez créer un projet d'application console .NET Core dont le seul but est d'agir comme projet de démarrage pour les outils. Le projet peut être un projet factice sans code réel — il n'est nécessaire que pour fournir une cible pour les outils.

Pourquoi un projet factice est-il nécessaire ? Comme mentionné précédemment, les outils doivent exécuter le code de l'application au moment de la conception. Pour ce faire, ils doivent utiliser le Runtime .NET Core.

Lorsque le modèle de EF Core se trouve dans un projet qui cible .NET Core ou .NET Framework, les outils de EF Core empruntent le runtime du projet. Ils ne peuvent pas le faire si le modèle de EF Core se trouve dans une bibliothèque de classes .NET Standard. Le .NET Standard n'est pas une implémentation .NET réelle. Il s'agit d'une spécification d'un ensemble d'API que les implémentations .NET doivent prendre en charge. Par conséquent, .NET Standard n'est pas suffisant pour que les outils de EF Core exécutent le code d'application. Le projet factice que vous créez à utiliser comme projet de démarrage fournit une plateforme cible concrète dans laquelle les outils peuvent charger la bibliothèque de classes .NET Standard.

## Environnement de ASP.NET Core

Pour spécifier l'environnement pour les projets ASP.NET Core, définissez la variable d'environnement **ASPNETCORE\_ENVIRONMENT** avant d'exécuter les commandes.

## Options courantes

|    | OPTION                          | DESCRIPTION  |
|----|---------------------------------|--|
|    | --json                          | Affichez la sortie JSON.   |
| -c | --context <DBCONTEXT>           | Classe <code>DbContext</code> à utiliser. Nom de classe uniquement ou qualifié complet avec des espaces de noms. Si cette option est omise, EF Core trouvera la classe de contexte. S'il existe plusieurs classes de contexte, cette option est obligatoire. |
| -p | --project <PROJECT>             | Chemin d'accès relatif au dossier du projet cible. La valeur par défaut est le dossier actif.  |
| -s | --startup-project <PROJECT>     | Chemin d'accès relatif au dossier du projet de démarrage. La valeur par défaut est le dossier actif.   |
|    | --framework <FRAMEWORK>         | Moniker du Framework cible pour la version <a href="#">cible du .NET Framework</a> . Utilisez lorsque le fichier projet spécifie plusieurs frameworks cibles et que vous souhaitez en sélectionner un.   |
|    | --configuration <CONFIGURATION> | Configuration de build, par exemple : <code>Debug</code> ou <code>Release</code> .   |
|    | --runtime <IDENTIFIER>          | Identificateur du runtime cible pour lequel restaurer les packages. Pour connaître les identificateurs de runtime, consultez le <a href="#">catalogue des identificateurs de runtime</a> .   |
| -h | --help                          | Affichez les informations d'aide.  |
| -v | --verbose                       | Affichez la sortie détaillée.  |
|    | --no-color                      | Ne pas colorier la sortie.   |
|    | --prefix-output                 | Sortie de préfixe avec niveau.   |

## dépôt de base de données EF dotnet

Supprime la base de données.

Options :

|    | OPTION    | DESCRIPTION   |
|----|-----------|---|
| -f | --force   | Ne pas confirmer.   |
|    | --dry-run | Affichez la base de données qui sera supprimée, mais ne la supprimez pas. |

## mise à jour de la base de données dotnet EF

Met à jour la base de données jusqu'à la dernière migration ou à une migration spécifiée.

Arguments :

| ARGUMENT    | DESCRIPTION   |
|-------------|---|
| <MIGRATION> | Migration cible. Les migrations peuvent être identifiées par leur nom ou par leur ID. Le nombre 0 est un cas spécial qui signifie <i>avant la première migration</i> et entraîne la restauration de toutes les migrations. Si aucune migration n'est spécifiée, la commande prend par défaut la dernière migration. |

Les exemples suivants mettent à jour la base de données vers une migration spécifiée. Le premier utilise le nom de la migration et le second utilise l'ID de migration :

```
dotnet ef database update InitialCreate
dotnet ef database update 20180904195021_InitialCreate
```

## informations sur la DbContext dotnet EF

Obtient des informations sur un type de `DbContext`.

## liste de DbContext EF dotnet

Répertorie les types de `DbContext` disponibles.

## structure de l'DbContext dotnet EF

Génère du code pour une `DbContext` et des types d'entités pour une base de données. Pour que cette commande génère un type d'entité, la table de base de données doit avoir une clé primaire.

Arguments :

| ARGUMENT     | DESCRIPTION  |
|--------------|--|
| <CONNECTION> | Chaîne de connexion à la base de données. Pour les projets ASP.NET Core 2.x, la valeur peut être <i>Name =&lt;nom de la chaîne de connexion&gt;</i> . Dans ce cas, le nom provient des sources de configuration qui sont configurées pour le projet. |
| <PROVIDER>   | Fournisseur à utiliser. En général, il s'agit du nom du package NuGet, par exemple : <code>Microsoft.EntityFrameworkCore.SqlServer</code> .  |

Options :

|    | OPTION                     | DESCRIPTION   |
|----|----------------------------|---|
| -d | --data-annotations         | Utilisez des attributs pour configurer le modèle (dans la mesure du possible). Si cette option est omise, seule l'API Fluent est utilisée.  |
| -c | --context <NAME>           | Nom de la classe <code>DbContext</code> à générer.  |
|    | --context-dir <PATH>       | Répertoire dans lequel placer le fichier de classe <code>DbContext</code> . Les chemins d'accès sont relatifs au répertoire du projet. Les espaces de noms sont dérivés des noms de dossiers.   |
| -f | --force                    | Remplacer les fichiers existants.   |
| -o | --output-dir <PATH>        | Répertoire dans lequel placer les fichiers de classe d'entité. Les chemins d'accès sont relatifs au répertoire du projet.   |
|    | --schema <SCHEMA_NAME> ... | Schémas des tables pour lesquelles générer des types d'entité. Pour spécifier plusieurs schémas, répétez <code>--schema</code> pour chacun d'entre eux. Si cette option est omise, tous les schémas sont inclus.                        |
| -t | --table <TABLE_NAME> ... » | Tables pour lesquelles générer des types d'entité. Pour spécifier plusieurs tables, répétez <code>-t</code> ou <code>--table</code> pour chacune d'elles. Si cette option est omise, toutes les tables sont incluses.                   |
|    | --use-database-names       | Utilisez les noms de table et de colonne exactement tels qu'ils apparaissent dans la base de données. Si cette option est omise, les noms de base de données sont modifiés pour C# être plus conformes aux conventions de style de nom. |

L'exemple suivant génère la structure de tous les schémas et tables et place les nouveaux fichiers dans le dossier *Models* .

```
dotnet ef dbcontext scaffold "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;" 
Microsoft.EntityFrameworkCore.SqlServer -o Models
```

L'exemple suivant génère uniquement les tables sélectionnées et crée le contexte dans un dossier distinct avec un nom spécifié :

```
dotnet ef dbcontext scaffold "Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;" 
Microsoft.EntityFrameworkCore.SqlServer -o Models -t Blog -t Post --context-dir Context -c BlogContext
```

## ajout des migrations dotnet EF

Ajoute une nouvelle migration.

Arguments :

| ARGUMENT | DESCRIPTION          |
|----------|----------------------|
| <NAME>   | Nom de la migration. |

Options :

|    | OPTION              | DESCRIPTION   |
|----|---------------------|---|
| -o | --output-dir <PATH> | Répertoire (et sous-espace de noms) à utiliser. Les chemins d'accès sont relatifs au répertoire du projet. La valeur par défaut est « migrations ». |

## Liste des migrations dotnet EF

Répertorie les migrations disponibles.

## suppression des migrations dotnet EF

Supprime la dernière migration (restaure les modifications de code qui ont été effectuées pour la migration).

Options :

|    | OPTION  | DESCRIPTION  |
|----|---------|--|
| -f | --force | Rétablissement la migration (annulez les modifications qui ont été appliquées à la base de données). |

## script de migrations dotnet EF

Génère un script SQL à partir des migrations.

Arguments :

| ARGUMENT | DESCRIPTION  |
|----------|--|
| <FROM>   | Début de la migration. Les migrations peuvent être identifiées par leur nom ou par leur ID. Le nombre 0 est un cas spécial qui signifie <i>avant la première migration</i> . La valeur par défaut est 0. |
| <TO>     | Fin de la migration. La valeur par défaut est la dernière migration.   |

Options :

|    | OPTION          | DESCRIPTION   |
|----|-----------------|---|
| -o | --output <FILE> | Fichier dans lequel écrire le script.   |
| -i | --idempotent    | Générez un script qui peut être utilisé sur une base de données lors d'une migration. |

L'exemple suivant crée un script pour la migration InitialCreate :

```
dotnet ef migrations script 0 InitialCreate
```

L'exemple suivant crée un script pour toutes les migrations après la migration de InitialCreate.

```
dotnet ef migrations script 20180904195021_InitialCreate
```

## Ressources supplémentaires

- [Migrations](#)
- [Reconstitution de la logique des produits](#)

# Création de DbContext au moment de la conception

07/11/2019 • 4 minutes to read

Certaines des commandes des outils de EF Core (par exemple, les commandes de [migration](#)) requièrent la création d'une instance de `DbContext` dérivée au moment de la conception afin de collecter des détails sur les types d'entité de l'application et la façon dont ils sont mappés à un schéma de base de données. Dans la plupart des cas, il est souhaitable que le `DbContext` créé soit configuré de manière similaire à la façon dont il est [configuré au moment](#) de l'exécution.

Les outils essaient de créer le `DbContext` de différentes manières :

## À partir des services d'application

Si votre projet de démarrage utilise l'[hôte Web ASP.net Core](#) ou l'[hôte générique .net Core](#), les outils essaient d'obtenir l'objet `DbContext` à partir du fournisseur de services de l'application.

Les outils essaient d'abord d'obtenir le fournisseur de services en appelant `Program.CreateHostBuilder()`, en appelant `Build()`, puis en accédant à la propriété `Services`.

```
public class Program
{
    public static void Main(string[] args)
        => CreateHostBuilder(args).Build().Run();

    // EF Core uses this method at design time to access the DbContext
    public static IHostBuilder CreateHostBuilder(string[] args)
        => Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(
                webBuilder => webBuilder.UseStartup<Startup>());
}

public class Startup
{
    public void ConfigureServices(IServiceCollection services)
        => services.AddDbContext<ApplicationDbContext>();

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        => app.UseDeveloperExceptionPage();
}

public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

### NOTE

Lorsque vous créez une application de ASP.NET Core, ce hook est inclus par défaut.

La `DbContext` elle-même et toutes les dépendances de son constructeur doivent être inscrites en tant que services dans le fournisseur de services de l'application. Pour ce faire, il est facile d'avoir [un constructeur sur la `DbContext` qui accepte une instance de `DbContextOptions<TContext>` en tant qu'argument](#) et utilise la [méthode `AddDbContext<TContext>`](#).

## Utilisation d'un constructeur sans paramètres

Si `DbContext` ne peut pas être obtenu à partir du fournisseur de services d'application, les outils recherchent le type de `DbContext` dérivé dans le projet. Ensuite, ils essaient de créer une instance à l'aide d'un constructeur sans paramètres. Il peut s'agir du constructeur par défaut si la `DbContext` est configurée à l'aide de la méthode `OnConfiguring`.

## À partir d'une fabrique au moment de la conception

Vous pouvez également indiquer aux outils comment créer votre `DbContext` en implémentant l'interface `IDesignTimeDbContextFactory<TContext>` : si une classe qui implémente cette interface est trouvée dans le même projet que le `DbContext` dérivé ou dans le projet de démarrage de l'application, les outils ignorent l'autre méthodes de création de `DbContext` et d'utilisation de la fabrique au moment du Design.

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.EntityFrameworkCore.Infrastructure;

namespace MyProject
{
    public class BloggingContextFactory : IDesignTimeDbContextFactory<BloggingContext>
    {
        public BloggingContext CreateDbContext(string[] args)
        {
            var optionsBuilder = new DbContextOptionsBuilder<BloggingContext>();
            optionsBuilder.UseSqlite("Data Source=blog.db");

            return new BloggingContext(optionsBuilder.Options);
        }
    }
}
```

### NOTE

Le paramètre `args` est actuellement inutilisé. [Un problème](#) est survenu lors du suivi de la capacité à spécifier des arguments au moment du design à partir des outils.

Une fabrique au moment de la conception peut être particulièrement utile si vous devez configurer la `DbContext` différemment pour le moment de la conception plutôt qu'au moment de l'exécution, si le constructeur `DbContext` prend des paramètres supplémentaires qui ne sont pas inscrits dans DI, si vous n'utilisez pas de DI du tout ou si, pour une raison quelconque, vous préférez ne pas avoir de méthode `BuildWebHost` dans la classe `Main` de votre application ASP.NET Core.

# Services au moment du design

07/11/2019 • 2 minutes to read

Certains services utilisés par les outils sont utilisés uniquement au moment du Design. Ces services sont gérés séparément des services d'exécution de EF Core pour les empêcher d'être déployés avec votre application. Pour remplacer l'un de ces services (par exemple, le service pour générer des fichiers de migration), ajoutez une implémentation de `IDesignTimeServices` à votre projet de démarrage.

```
class MyDesignTimeServices : IDesignTimeServices
{
    public void ConfigureDesignTimeServices(IServiceCollection services)
        => services.AddSingleton<IMigrationsCodeGenerator, MyMigrationsCodeGenerator>();
}
```

# Entity Framework 6

08/11/2019 • 4 minutes to read • [Edit Online](#)

Entity Framework 6 (EF6) est un mapeur Objet Relationnel (O/RM) éprouvé pour .NET qui bénéficie de nombreuses années de développement de fonctionnalités et de stabilisation.

Tout comme un O/RM, EF6 réduit la différence d'impédance entre les mondes Relationnel et Objet. Cela permet aux développeurs d'écrire des applications qui interagissent avec des données stockées dans des bases de données relationnelles à l'aide d'objets .NET fortement typés représentant le domaine de l'application. De cette façon, ils n'ont plus besoin d'écrire une grande partie du code « de raccordement » qui sert à accéder aux données.

EF6 implémente de nombreuses fonctionnalités O/RM populaires :

- Mappage de classes d'entité [OCT](#) qui ne dépendent pas d'un type EF
- Détection automatique des changements
- Résolution d'identité et unité de travail
- Chargement hâtif, différé et explicite
- Traduction des requêtes fortement typées à l'aide de [LINQ \(Language Integrated Query\)](#)
- Fonctionnalités de mappage complètes, notamment la prise en charge des points suivants :
  - Relations un-à-un, un-à-plusieurs et plusieurs-à-plusieurs
  - Héritage (table par hiérarchie, table par type et table par classe concrète)
  - Types complexes
  - Procédures stockées
- Un concepteur visuel pour créer des modèles d'entité.
- Une expérience « Code First » pour créer des modèles d'entité en écrivant du code.
- Les modèles peuvent être générés à partir de bases de données existantes et modifiés à la main, ou créés à partir de zéro et utilisés pour générer de nouvelles bases de données.
- Intégration aux modèles d'application .NET Framework, y compris ASP.NET, et au moyen de la liaison de données avec WPF et WinForms.
- Connectivité de base de données basée sur ADO.NET et nombreux [fournisseurs](#) disponibles pour établir la connexion à SQL Server, Oracle, MySQL, SQLite, PostgreSQL, DB2, etc.

## Dois-je utiliser EF6 ou EF Core ?

EF Core est une version plus moderne, légère et extensible d'Entity Framework, qui a des avantages et des fonctionnalités très similaires à EF6. EF Core est le résultat d'une réécriture complète et contient de nombreuses nouvelles fonctionnalités non disponibles dans EF6, même s'il lui manque encore certaines des fonctionnalités de mappage les plus avancées d'EF6. Si l'ensemble des fonctionnalités correspond à vos besoins, vous pouvez utiliser EF Core dans les nouvelles applications. La section [Comparer EF Core et EF6](#) examine ce choix plus en détail.

## Bien démarrer

Ajoutez le package NuGet EntityFramework à votre projet ou installez [Entity Framework Tools pour Visual Studio](#). Ensuite, regardez les vidéos, lisez les tutoriels et la documentation avancée pour vous aider à tirer le meilleur parti d'EF6.

## Versions précédentes d'Entity Framework

Il s'agit de la documentation de la dernière version d'Entity Framework 6, bien qu'une grande partie s'applique

aussi aux versions précédentes. Consultez les rubriques [Nouveautés](#) et [Versions précédentes](#) pour obtenir la liste complète des versions EF et des fonctionnalités introduites par chacune.

# Nouveautés dans EF6

24/10/2019 • 3 minutes to read

Nous vous recommandons vivement d'utiliser la dernière version publiée d'Entity Framework pour bénéficier des dernières fonctionnalités et d'une stabilité optimale. Toutefois, vous pouvez aussi avoir besoin d'utiliser une version antérieure ou vouloir tester les nouvelles améliorations de la dernière préversion. Pour installer des versions spécifiques d'EF, consultez [Obtenir Entity Framework](#).

## EF 6.3.0

Le runtime EF 6.3.0 a été publié sur NuGet en septembre 2019. L'objectif principal de cette version était de faciliter la migration des applications existantes qui utilisent EF 6 vers .NET Core 3.0. La communauté a également contribué à plusieurs correctifs de bogues et améliorations. Pour plus d'informations, consultez les problèmes fermés à chaque [jalon](#) de la version 6.3.0. Voici les plus connus :

- Prise en charge de .NET Core 3.0
  - Le package EntityFramework cible maintenant .NET Standard 2.1 en plus de .NET Framework 4.x.
  - Cela signifie que EF 6.3 est multiplateforme et pris en charge sur d'autres systèmes d'exploitation que Windows, comme Linux et macOS.
  - Les commandes de migrations ont été réécrites pour s'exécuter hors processus et fonctionner avec des projets de type SDK.
- Prise en charge de SQL Server HierarchyId.
- Compatibilité améliorée avec Roslyn et NuGet PackageReference.
- Ajout de l'utilitaire `ef6.exe` pour l'activation, l'ajout, l'écriture de scripts et l'application de migrations à partir d'assemblies. Ceci remplace `migrate.exe`.

### Prise en charge du concepteur EF

Il n'existe actuellement aucune prise en charge pour l'utilisation du concepteur EF directement sur les projets .NET Core ou .NET Standard.

Vous pouvez contourner cette limitation en ajoutant le fichier EDMX et les classes générées pour les entités et pour DbContext en tant que fichiers liés à un projet .NET Core 3.0 ou .NET Standard 2.1 dans la même solution.

Les fichiers liés ressembleront à ce qui suit dans le fichier projet :

```
<ItemGroup>
  <EntityDeploy Include="..\EdmxDesignHost\Entities.edmx" Link="Model\Entities.edmx" />
  <Compile Include="..\EdmxDesignHost\Entities.Context.cs" Link="Model\Entities.Context.cs" />
  <Compile Include="..\EdmxDesignHost\Thing.cs" Link="Model\Thing.cs" />
  <Compile Include="..\EdmxDesignHost\Person.cs" Link="Model\Person.cs" />
</ItemGroup>
```

Notez que le fichier EDMX est lié à l'action de création EntityDeploy. Il s'agit d'une tâche MSBuild spéciale (désormais incluse dans le package EF 6.3) qui prend en charge l'ajout du modèle EF à l'assembly cible en tant que ressources incorporées (ou copiez-le en tant que fichiers dans le dossier de sortie, en fonction du paramètre de traitement des artefacts de métadonnées dans EDMX). Pour plus d'informations sur la façon d'obtenir cette configuration, consultez notre [exemple .NET Core EDMX](#).

## Versions précédentes

La page [Versions précédentes](#) contient une archive de toutes les versions précédentes d'Entity Framework et des principales fonctionnalités introduites dans chaque version.

# Versions antérieures de Entity Framework

07/11/2019 • 31 minutes to read

La première version de Entity Framework a été publiée dans 2008, dans le cadre de .NET Framework 3,5 SP1 et de Visual Studio 2008 SP1.

À partir de la version d'EF 4.1, elle a été fournie en tant que [package NuGet d'EntityFramework](#), actuellement l'un des packages les plus populaires sur NuGet.org.

Entre les versions 4,1 et 5,0, le package NuGet EntityFramework étend les bibliothèques EF fournies dans le cadre de .NET Framework.

À partir de la version 6, EF est devenu un projet open source et a également été déplacé complètement hors bande sous la .NET Framework. Cela signifie que lorsque vous ajoutez le package NuGet EntityFramework version 6 à une application, vous obtenez une copie complète de la bibliothèque EF qui ne dépend pas des bits EF fournis dans le cadre de .NET Framework. Cela nous a permis d'accélérer le rythme du développement et de la livraison de nouvelles fonctionnalités.

En juin 2016, nous avons publié EF Core 1,0. EF Core est basé sur un nouveau code base et est conçu comme une version plus légère et extensible d'EF. Actuellement EF Core est l'objectif principal du développement pour l'équipe Entity Framework chez Microsoft. Cela signifie qu'il n'y a pas de nouvelles fonctionnalités majeures planifiées pour EF6. Toutefois, EF6 est toujours géré comme un projet open source et un produit Microsoft pris en charge.

Voici la liste des versions antérieures, dans l'ordre chronologique inverse, avec les informations sur les nouvelles fonctionnalités qui ont été introduites dans chaque version.

## Mise à jour d'EF Tools dans Visual Studio 2017 15.7

En mai 2018, nous avons publié une version mise à jour d'EF Tools dans Visual Studio 2017 15.7. Cette version contient des améliorations sur certains points de problème courants :

- Correctifs pour plusieurs bogues d'accessibilité de l'interface utilisateur
- Solution de contournement pour la régression des performances de SQL Server pendant la génération de modèles à partir de bases de données existantes [#4](#)
- Prise en charge de la mise à jour des gros modèles sur SQL Server [#185](#)

Une autre amélioration de cette nouvelle version d'EF Tools est l'installation du runtime EF 6.2 quand vous créez un modèle dans un nouveau projet. Avec les versions précédentes de Visual Studio, vous pouvez utiliser le runtime EF 6.2 (ainsi que n'importe quelle version précédente d'EF) en installant la version correspondante du package NuGet.

## EF 6.2.0

Le runtime EF 6.2 a été publié sur NuGet en octobre 2017. Grâce en majeure partie aux efforts de notre communauté de contributeurs open source, EF 6.2 comprend de nombreux [correctifs de bogues](#) et [améliorations de produit](#).

Voici une courte liste des changements les plus importants du runtime EF 6.2 :

- Réduction du temps de démarrage en chargeant les modèles Code First terminés à partir d'un cache persistant [#275](#)

- API Fluent pour définir des index #274
- DbFunctions.Like() pour activer l'écriture de requêtes LINQ qui se traduisent en LIKE dans SQL #241
- Migrate.exe prend désormais en charge l'option -script #240
- EF6 peut maintenant utiliser des valeurs de clé générées par une séquence dans SQL Server #165
- Liste de mise à jour des erreurs temporaires pour la stratégie d'exécution de SQL Azure #83
- Bogue : La réexécution des requêtes ou des commandes SQL échoue avec le message « SqlParameter est déjà contenu par un autre SqlParameterCollection » #81
- Bogue : L'évaluation de DbQuery.ToString() expire dans le débogueur #73

## EF 6.1.3

Le runtime EF 6.1.3 a été publié à NuGet en octobre 2015. Cette version contient uniquement des correctifs pour les défauts de priorité élevée et les régressions signalées sur la version 6.1.2. Les correctifs sont les suivants :

- Requête : régression dans EF 6.1.2 : application externe introduite et requêtes plus complexes pour les relations 1:1 et la clause « let »
- TPT problème avec la propriété de masquage de la classe de base dans la classe héritée
- DbMigration.SQL échoue quand le mot « Go » est contenu dans le texte
- Créer un indicateur de compatibilité pour UnionAll insère et Intersect la prise en charge de l'aplatissement
- La requête avec plusieurs includes ne fonctionne pas dans 6.1.2 (travail en mode 6.1.1)
- «Vous avez une erreur dans la syntaxe SQL «exception après la mise à niveau d'EF 6.1.1 vers 6.1.2

## EF 6.1.2

Le runtime EF 6.1.2 a été publié dans NuGet en décembre de 2014. Cette version concerne essentiellement les correctifs de bogues. Nous avons également accepté quelques modifications intéressantes des membres de la communauté :

- **Les paramètres du cache de requête peuvent être configurés à partir du fichier app/Web.**  
**Configuration.**

```
<entityFramework>
  <queryCache size='1000' cleaningIntervalInSeconds=' -1' />
</entityFramework>
```

- **Les méthodes sqlfile et SqlResource sur DbMigration** vous permettent d'exécuter un script SQL stocké sous la forme d'un fichier ou d'une ressource incorporée.

## EF 6.1.1

Le runtime EF 6.1.1 a été publié en version NuGet en juin de 2014. Cette version contient des correctifs pour les problèmes rencontrés par un certain nombre de personnes. Entre autres :

- Concepteur : erreur lors de l'ouverture de EF5 edmx avec une précision décimale dans EF6 designer
- La logique de détection d'instance par défaut pour la base de données locale ne fonctionne pas avec SQL Server 2014

## EF 6.1.0

Le runtime EF 6.1.0 a été publié en version NuGet en mars de 2014. Cette mise à jour mineure comprend un nombre important de nouvelles fonctionnalités :

- **La consolidation des outils** offre un moyen cohérent de créer un modèle EF. Cette fonctionnalité étend

l'assistant Entity Data Model ADO.net pour prendre en charge la création de modèles de code **First**, y compris l'ingénierie à rebours à partir d'une base de données existante. Ces fonctionnalités étaient auparavant disponibles en version bêta dans EF Power Tools.

- La **gestion des échecs de validation de transaction** fournit le CommitFailureHandler qui utilise la capacité récemment introduite pour intercepter les opérations de transaction. Le CommitFailureHandler permet la récupération automatique à partir des échecs de connexion pendant la validation d'une transaction.
- **IndexAttribute** permet de spécifier des index en plaçant un attribut `[Index]` sur une propriété (ou des propriétés) dans votre modèle de code First. Code First crée alors un index correspondant dans la base de données.
- **L'API de mappage public** fournit l'accès aux informations EF sur la manière dont les propriétés et les types sont mappés aux colonnes et aux tables de la base de données. Dans les versions antérieures, cette API était interne.
- **La possibilité de configurer des intercepteurs via le fichier app/Web. config** permet d'ajouter des intercepteurs sans recompiler l'application.
- **System. Data. Entity. infrastructure. interception. DatabaseLogger** est un nouvel intercepteur qui facilite l'enregistrement de toutes les opérations de base de données dans un fichier. En association avec la fonctionnalité précédente, cela vous permet de [basculer facilement la journalisation des opérations de base de données pour une application déployée](#), sans avoir à recompiler.
- La **détection des modifications du modèle de migrations** a été améliorée afin que les migrations par génération de modèles automatique soient plus précises. les performances du processus de détection des modifications ont également été améliorées.
- **Améliorations des performances**, y compris les opérations de base de données réduites pendant l'initialisation, les optimisations de comparaison d'égalité null dans les requêtes LINQ, la génération de vues plus rapide (création de modèle) dans plus de scénarios et la matérialisation plus efficace de entités suivies avec plusieurs associations.

## EF 6.0.2

Le runtime EF 6.0.2 a été publié en version NuGet en décembre de 2013. Cette version de correctif est limitée à la résolution des problèmes qui ont été introduits dans la version EF6 (régressions en matière de performances/comportement depuis EF5).

## EF 6.0.1

Le runtime EF 6.0.1 a été publié en octobre en octobre 2013 avec EF 6.0.0, car ce dernier a été incorporé dans une version de Visual Studio qui avait été verrouillée quelques mois auparavant. Cette version de correctif est limitée à la résolution des problèmes qui ont été introduits dans la version EF6 (régressions en matière de performances/comportement depuis EF5). Les modifications les plus notables devaient résoudre certains problèmes de performances lors du préchauffage des modèles EF. Cela était important, car les performances de mise en route étaient une zone de focus dans EF6 et ces problèmes étaient la Niere des autres gains de performances apportés à EF6.

## EF 6,0

Le runtime EF 6.0.0 a été lancé en version NuGet en octobre 2013. Il s'agit de la première version dans laquelle un runtime EF complet est inclus dans le [package NuGet EntityFramework](#) qui ne dépend pas des bits EF qui font partie du .NET Framework. Le déplacement des parties restantes du runtime vers le package NuGet nécessitait un certain nombre de modifications avec rupture pour le code existant. Pour plus d'informations sur les étapes manuelles nécessaires à la mise à niveau, consultez la section relative à la mise à niveau [vers Entity Framework 6](#).

Cette version comprend de nombreuses nouvelles fonctionnalités. Les fonctionnalités suivantes fonctionnent pour les modèles créés avec Code First ou le concepteur EF :

- L'**enregistrement et la requête asynchrones** ajoutent la prise en charge des modèles asynchrones basés sur les tâches qui ont été introduits dans .net 4,5.
- La **résilience des connexions** permet la récupération automatique après des échecs de connexion temporaires.
- La **configuration basée sur le code** vous donne la possibilité d'effectuer la configuration, traditionnellement effectuée dans un fichier de configuration, dans le code.
- La **Résolution des dépendances** introduit la prise en charge du modèle de localisation de service et nous avons pris en compte certains éléments de fonctionnalité qui peuvent être remplacés par des implémentations personnalisées.
- L'**interception/la journalisation SQL** fournit des blocs de construction de bas niveau pour l'interception des opérations EF avec une simple journalisation SQL basée sur.
- Les améliorations de la **testabilité** facilitent la création de doubles de test pour DbContext et DbSet lors de l'utilisation d'une infrastructure fictive ou de l'écriture de vos propres doubles de test.
- **DbContext peut maintenant être créé avec un DbConnection déjà ouvert**, ce qui permet des scénarios dans lesquels il serait utile de pouvoir ouvrir la connexion lors de la création du contexte (par exemple, le partage d'une connexion entre des composants où vous ne pouvez pas garantir l'état de la connexion).
- La **prise en charge améliorée des transactions** assure la prise en charge d'une transaction externe à l'infrastructure, ainsi que des méthodes améliorées de création d'une transaction dans l'infrastructure.
- **Enums, spatiales et meilleures performances sur .net 4,0** : en déplaçant les composants principaux qui se trouvaient dans le .NET Framework dans le package NUGET d'EF, nous pouvons désormais offrir une prise en charge des énumérations, des types de données spatiales et des améliorations des performances de EF5 sur .net 4,0.
- **Amélioration des performances de Enumerable. Contains dans les requêtes LINQ.**
- **Amélioration du temps de préchauffage (génération de vues)** , en particulier pour les modèles volumineux.
- La **pluralisation enfichable & service de singularité**.
- Les **implémentations personnalisées de Equals ou GetHashCode** sur les classes d'entité sont désormais prises en charge.
- **DbSet.AddRange/RemoveRange** fournit une méthode optimisée pour ajouter ou supprimer plusieurs entités d'un ensemble.
- **DbChangeTracker. HasChanges** offre un moyen simple et efficace de déterminer si des modifications en attente doivent être enregistrées dans la base de données.
- **SqlCeFunctions** fournit un compact SQL équivalent à SqlFunctions.

Les fonctionnalités suivantes s'appliquent à Code First uniquement :

- Les **conventions de code First personnalisées** permettent d'écrire vos propres conventions afin d'éviter une configuration répétitive. Nous fournissons une API simple pour les conventions légères, ainsi que des blocs de construction plus complexes pour vous permettre de créer des conventions plus compliquées.
- **Code First mappage aux procédures stockées d'insertion/mise à jour/suppression** est désormais pris en charge.
- Les **scripts de migration idempotent** vous permettent de générer un script SQL qui peut mettre à niveau une base de données, quelle que soit la version, jusqu'à la version la plus récente.
- La table de l'**historique des migrations configurables** vous permet de personnaliser la définition de la table d'historique des migrations. Cela s'avère particulièrement utile pour les fournisseurs de bases de données qui nécessitent des types de données appropriés, etc., à spécifier pour que la table d'historique des migrations fonctionne correctement.
- **Plusieurs contextes par base de données** suppriment la limitation précédente d'un modèle de code First par base de données lors de l'utilisation de migrations ou lorsque code First crée automatiquement la base de données pour vous.

- **DbModelBuilder. HasDefaultSchema** est une nouvelle API code First qui permet de configurer le schéma de base de données par défaut d'un modèle de code First à un seul emplacement. Précédemment, le Code First schéma par défaut a été codé en dur pour "" DBO et la seule façon de configurer le schéma auquel une table appartenait était via l'API ToTable.
- La **méthode DbModelBuilder. configurations. AddFromAssembly** vous permet d'ajouter facilement toutes les classes de configuration définies dans un assembly lorsque vous utilisez des classes de configuration avec l'API Fluent code First.
- Les **opérations de migrations personnalisées** vous ont permis d'ajouter des opérations supplémentaires à utiliser dans vos migrations basées sur le code.
- Le **niveau d'isolation de la transaction par défaut est modifié en READ\_COMMITTED\_SNAPSHOT** pour les bases de données créées à l'aide de code First, ce qui permet une plus grande évolutivité et un plus petit nombre de blocages.
- **Les types d'entité et complexes peuvent maintenant être des classes nestedinside.**

## EF 5,0

Le runtime EF 5.0.0 a été publié dans NuGet en août de 2012. Cette version introduit de nouvelles fonctionnalités, notamment la prise en charge des énumérations, les fonctions table, les types de données spatiales et diverses améliorations des performances.

La Entity Framework Designer dans Visual Studio 2012 introduit également la prise en charge de plusieurs diagrammes par modèle, la coloration des formes sur l'aire de conception et l'importation par lots des procédures stockées.

Voici la liste des contenus que nous avons rassemblés spécifiquement pour la version d'EF 5 :

- [Publication de la publication EF 5](#)
- Nouvelles fonctionnalités dans EF5
  - [Prise en charge des énumérations dans Code First](#)
  - [Prise en charge des énumérations dans EF designer](#)
  - [Types de données spatiales dans Code First](#)
  - [Types de données spatiales dans le concepteur EF](#)
  - [Prise en charge des fournisseurs pour les types spatiaux](#)
  - [Fonctions table](#)
  - [Plusieurs diagrammes par modèle](#)
- Configuration de votre modèle
  - [Création d'un modèle](#)
  - [Connexions et modèles](#)
  - [Considérations sur les performances](#)
  - [Utilisation de Microsoft SQL Azure](#)
  - [Paramètres du fichier de configuration](#)
  - [Glossaire](#)
  - [Code First](#)
    - [Code First à une nouvelle base de données \(procédure pas à pas et vidéo\)](#)
    - [Code First à une base de données existante \(procédure pas à pas et vidéo\)](#)
    - [Conventions](#)
    - [Annotations de données](#)
    - [API Fluent-configuration/mappage des propriétés & types](#)
    - [API Fluent-configuration des relations](#)
    - [API Fluent avec VB.NET](#)

- [Migrations Code First](#)
- [Migrations Code First automatique](#)
- [Migrate.exe](#)
- [Définition de DbSets](#)
- EF Designer
  - [Model First \(procédure pas à pas et vidéo\)](#)
  - [Database First \(procédure pas à pas et vidéo\)](#)
  - [Types complexes](#)
  - [Associations/relations](#)
  - [Modèle d'héritage TPT](#)
  - [Modèle d'héritage TPH](#)
  - [Interroger avec des procédures stockées](#)
  - [Procédures stockées avec plusieurs jeux de résultats](#)
  - [Insérer, mettre à jour & supprimer avec des procédures stockées](#)
  - [Mapper une entité à plusieurs tables \(fractionnement d'entités\)](#)
  - [Mapper plusieurs entités à une seule table \(fractionnement de table\)](#)
  - [Définition de requêtes](#)
  - [Modèles de génération de code](#)
  - [Rétablissement d'ObjectContext](#)
- Utilisation de votre modèle
  - [Utilisation de DbContext](#)
  - [Interrogation/recherche d'entités](#)
  - [Utilisation des relations](#)
  - [Chargement des entités associées](#)
  - [Utilisation des données locales](#)
  - [Applications multicouches](#)
  - [Requêtes SQL brutes](#)
  - [Modèles d'accès concurrentiel optimiste](#)
  - [Utilisation des proxies](#)
  - [Détection automatique des modifications](#)
  - [Requêtes de suivi sans](#)
  - [Méthode Load](#)
  - [Ajouter/attacher des États et des entités](#)
  - [Utilisation des valeurs de propriété](#)
  - [Liaison de données avec WPF \(Windows Presentation Foundation\)](#)
  - [Liaison de données avec WinForms \(Windows Forms\)](#)

## EF 4.3.1

Le runtime EF 4.3.1 a été publié en NuGet en février 2012 peu après EF 4.3.0. Cette version de correctif inclut des correctifs de bogues pour la version d'EF 4.3 et a introduit une meilleure prise en charge de la base de données locale pour les clients utilisant EF 4.3 avec Visual Studio 2012.

Voici une liste de contenu spécifiquement mis en place pour la version d'EF 4.3.1. La majeure partie du contenu fourni pour EF 4.1 s'applique également à EF 4.3 :

- [Billet de blog sur la version EF 4.3.1](#)

## EF 4,3

Le runtime EF 4.3.0 a été publié en version NuGet en février de 2012. Cette version comprenait la nouvelle fonctionnalité de Migrations Code First qui permet de modifier de façon incrémentielle une base de données créée par Code First à mesure que votre modèle de Code First évolue.

Voici une liste de contenu spécifiquement mis en place pour la version d'EF 4,3, mais la plupart du contenu fourni pour EF 4,1 s'applique également à EF 4,3 :

- [Publication de la publication EF 4,3](#)
- [Procédure pas à pas de migrations basées sur du code EF 4,3](#)
- [Procédure pas à pas de migration automatique EF 4,3](#)

## EF 4,2

Le runtime EF 4.2.0 a été publié en version NuGet en novembre 2011. Cette version comprend des correctifs de bogues pour la version d'EF 4.1.1. Étant donné que cette version ne comprenait que des correctifs de bogues, il aurait peut-être été le correctif EF 4.1.2, mais nous avons choisi de passer à 4,2 pour nous permettre de quitter les numéros de version de correctifs basés sur la date que nous avons utilisés dans les versions 4.1.x et d'adopter la norme de [version sémantique](#) pour les contrôle de version emantic.

Voici une liste de contenu spécifiquement mis en place pour la version d'EF 4,2, le contenu fourni pour EF 4,1 s'applique également à EF 4,2 :

- [Publication de la publication EF 4,2](#)
- [Code First procédure pas à pas](#)
- [Guide pas à pas de Database First & de modèle](#)

## EF 4.1.1

Le runtime EF 4.1.10715 a été publié en NuGet en juillet de 2011. En plus des correctifs de bogues, cette version de correctif a introduit certains composants pour faciliter l'utilisation des outils au moment du design avec un modèle de Code First. Ces composants sont utilisés par Migrations Code First (inclus dans EF 4,3) et EF Power Tools.

Vous remarquerez que le numéro de version étrange 4.1.10715 du package. Nous avons utilisé des versions correctives basées sur la date avant de décider d'adopter le contrôle de [version sémantique](#). Considérez cette version comme EF 4,1 patch 1 (ou EF 4.1.1).

Voici une liste de contenu que nous avons rassemblés pour la version 4.1.1 :

- [Publication de la publication EF 4.1.1](#)

## EF 4,1

Le runtime 4.1.10331 EF était le premier à être publié sur NuGet, en avril de 2011. Cette version incluait l'API DbContext simplifiée et le flux de travail Code First.

Vous remarquerez le numéro de version étrange, 4.1.10331, qui doit vraiment être 4,1. En outre, il existe une version de 4.1.10311 qui devrait avoir été 4.1.0-RC (le « RC » signifie « version finale »). Nous avons utilisé des versions correctives basées sur la date avant de décider d'adopter le contrôle de [version sémantique](#).

Voici une liste de contenu que nous avons rassemblés pour la version 4,1. La plupart d'entre elles s'appliquent toujours aux versions ultérieures de Entity Framework :

- [Publication de la publication EF 4,1](#)

- [Code First procédure pas à pas](#)
- [Guide pas à pas de Database First & de modèle](#)
- [SQL Azure les fédérations et les Entity Framework](#)

## EF 4,0

Cette version a été incluse dans .NET Framework 4 et Visual Studio 2010, en avril 2010. Les nouvelles fonctionnalités importantes de cette version incluent la prise en charge d'POCO, le mappage de clé étrangère, le chargement différé, les améliorations de test, la génération de code personnalisable et le flux de travail de Model First.

Bien qu'il s'agisse de la deuxième version de Entity Framework, elle s'appelait EF 4 pour s'aligner sur la version de .NET Framework qu'elle a fournie avec. Après cette version, nous avons commencé à rendre Entity Framework disponibles sur NuGet et adopté le contrôle de version sémantique, car nous n'avons plus été liés à la version .NET Framework.

Notez que certaines versions ultérieures de .NET Framework ont été fournies avec des mises à jour significatives des bits EF inclus. En fait, la plupart des nouvelles fonctionnalités d'EF 5,0 ont été implémentées en tant qu'améliorations de ces bits. Toutefois, afin de rationaliser le contrôle de version pour EF, nous continuons à faire référence aux bits EF qui font partie du .NET Framework en tant que runtime EF 4,0, tandis que toutes les versions plus récentes se composent du [package NuGet EntityFramework](#).

## EF 3,5

La version initiale de Entity Framework a été incluse dans .NET 3,5 Service Pack 1 et Visual Studio 2008 SP1, publiée en août de 2008. Cette version a fourni la prise en charge de base de O/RM à l'aide du flux de travail Database First.

# Mise à niveau vers Entity Framework 6

23/11/2019 • 8 minutes to read

Dans les versions précédentes d'EF, le code était fractionné entre les bibliothèques principales (principalement System. Data. Entity. dll) fournies dans le cadre de la .NET Framework et les bibliothèques hors-bande (OOB) fournies dans un package NuGet. EF6 prend le code des bibliothèques principales et l'intègre dans les bibliothèques OOB. Cela était nécessaire pour permettre à EF d'être rendu Open source et pour pouvoir évoluer à un rythme différent de .NET Framework. Cela est dû au fait que les applications devront être reconstruites sur les types déplacés.

Cela doit être simple pour les applications qui utilisent DbContext comme fourni dans EF 4,1 et versions ultérieures. Un peu plus de travail est nécessaire pour les applications qui utilisent ObjectContext, mais il n'est pas encore difficile de le faire.

Voici une liste de contrôle des éléments que vous devez effectuer pour mettre à niveau une application existante vers EF6.

## 1. installer le package NuGet EF6

Vous devez effectuer une mise à niveau vers le nouveau runtime Entity Framework 6.

1. Cliquez avec le bouton droit sur votre projet et sélectionnez **gérer les packages NuGet...**
2. Sous l'onglet **en ligne**, sélectionnez **EntityFramework**, puis cliquez sur **installer**.

### NOTE

Si une version précédente du package NuGet EntityFramework a été installée, cette opération est mise à niveau vers EF6.

Vous pouvez également exécuter la commande suivante à partir de la console du gestionnaire de package :

```
Install-Package EntityFramework
```

## 2. Vérifiez que les références d'assembly à System. Data. Entity. dll sont supprimées

L'installation du package NuGet EF6 doit supprimer automatiquement toutes les références à System. Data. Entity de votre projet.

## 3. permutez n'importe quel modèle EF Designer (EDMX) pour utiliser la génération de code EF 6. x

Si vous avez créé des modèles avec le concepteur EF, vous devrez mettre à jour les modèles de génération de code pour générer du code compatible EF6.

### NOTE

Il n'existe actuellement que les modèles de générateur de DbContext EF 6. x disponibles pour Visual Studio 2012 et 2013.

1. Supprimer les modèles de génération de code existants. Ces fichiers sont généralement nommés `<edmx_file_name>.TT` et `<edmx_file_name>.Context.tt` et sont imbriqués sous votre fichier edmx dans Explorateur de solutions. Vous pouvez sélectionner les modèles dans Explorateur de solutions et appuyer sur la touche **Suppr** pour les supprimer.

**NOTE**

Dans les projets de site Web, les modèles ne sont pas imbriqués sous votre fichier edmx, mais sont listés en même temps dans Explorateur de solutions.

**NOTE**

Dans les projets VB.NET, vous devez activer « afficher tous les fichiers » pour pouvoir voir les fichiers de modèles imbriqués.

2. Ajoutez le modèle de génération de code EF 6.x approprié. Ouvrez votre modèle dans le concepteur EF, cliquez avec le bouton droit sur l'aire de conception, puis sélectionnez **Ajouter un élément de génération de code...**

- Si vous utilisez l'API DbContext (recommandé), le **Générateur de DbContext EF 6.x** sera disponible sous l'onglet **données**.

**NOTE**

Si vous utilisez Visual Studio 2012, vous devrez installer les outils EF 6 pour obtenir ce modèle. Pour plus d'informations, consultez [obtenir Entity Framework](#).

- Si vous utilisez l'API ObjectContext, vous devrez sélectionner l'onglet **Online** et rechercher le **Générateur EntityObject EF 6.x**.

3. Si vous avez appliqué des personnalisations aux modèles de génération de code, vous devez les réappliquer aux modèles mis à jour.

## 4. mettre à jour les espaces de noms pour tous les types EF de base utilisés

Les espaces de noms pour DbContext et les types Code First n'ont pas changé. Cela signifie que pour de nombreuses applications qui utilisent EF 4.1 ou une version ultérieure, vous n'aurez pas besoin de modifier quoi que ce soit.

Les types tels que ObjectContext qui étaient précédemment dans System. Data. Entity. dll ont été déplacés vers de nouveaux espaces de noms. Cela signifie que vous devrez peut-être mettre à jour vos directives *using* ou *Import* pour créer des EF6.

La règle générale pour les modifications d'espace de noms est que tout type dans System. Data.\* est déplacé vers System. Data. Entity. Core.\*. En d'autres termes, il suffit d'insérer **Entity. Core**. après System. Data. Exemple :

- System. Data. EntityException = > System. Data. **Entity. Core**. EntityException
- System. Data. Objects. ObjectContext = > System. Data. **Entity. Core**. Objects. ObjectContext
- System. Data. Objects. DataClasses. RelationshipManager = > System. Data. **Entity. Core**. Objets. DataClasses. RelationshipManager

Ces types se trouvent dans les espaces de noms de *base*, car ils ne sont pas utilisés directement pour la plupart

des applications basées sur DbContext. Certains types qui faisaient partie de System. Data. Entity. dll sont toujours utilisés communément et directement pour les applications basées sur DbContext et n'ont donc pas été déplacés dans les espaces de noms de *base*. Il s'agit des paramètres suivants :

- System. Data. EntityState = > System. Data. **Entité**. EntityState
- System. Data. Objects. DataClasses. EdmFunctionAttribute = > System. Data. **Entité**. **DbFunctionAttribute**

**NOTE**

Cette classe a été renommée ; une classe portant l'ancien nom existe toujours et fonctionne, mais elle est désormais marquée comme obsolète.

- System. Data. Objects. EntityFunctions = > System. Data. **Entité**. **DbFunctions**

**NOTE**

Cette classe a été renommée ; une classe portant l'ancien nom existe toujours et fonctionne, mais elle est désormais marquée comme obsolète.)

- Les classes spatiales (par exemple, DbGeography, DbGeometry) ont été déplacées de System. Data. spatial = > System. Data. **Entité**. Adjacent

**NOTE**

Certains types de l'espace de noms System. Data se trouvent dans System. Data. dll, qui n'est pas un assembly EF. Ces types n'ont pas été déplacés et leurs espaces de noms restent inchangés.

# Versions de Visual Studio

06/05/2019 • 8 minutes to read

Nous vous recommandons de toujours utiliser la dernière version de Visual Studio, car elle contient les derniers outils pour .NET, NuGet et Entity Framework. En fait, les différents exemples et procédures pas à pas dans la documentation d'Entity Framework supposent que vous utilisez une version récente de Visual Studio.

Il est possible cependant, pour utiliser les versions antérieures de Visual Studio avec différentes versions d'Entity Framework, que vous prenez en compte de certaines différences :

## Visual Studio 2017 15.7 et ultérieures

- Cette version de Visual Studio inclut la dernière version des outils Entity Framework et le runtime EF 6.2 et ne nécessite pas d'étapes de configuration supplémentaires. Consultez [What's New](#) pour plus d'informations sur ces versions.
- Ajout d'Entity Framework pour les nouveaux projets à l'aide des outils Entity Framework ajoute automatiquement le package NuGet EF 6.2. Vous pouvez manuellement installer ou mettre à niveau vers n'importe quel package NuGet d'EF disponible en ligne.
- Par défaut, l'instance de SQL Server disponible avec cette version de Visual Studio est une instance de base de données locale appelée MSSQLLocalDB. La section serveur de la chaîne de connexion que vous devez utiliser est « (localdb)\MSSQLLocalDB ». N'oubliez pas d'utiliser un préfixe de chaîne textuelle @ ou doubles barres obliques inverses «\\» lorsque vous spécifiez une chaîne de connexion dans le code C#.

## Visual Studio 2015 vers Visual Studio 2017 15.6

- Ces versions de Visual Studio incluent des outils Entity Framework et runtime 6.1.3. Consultez [libère dernières](#) pour plus d'informations sur ces versions.
- Ajout d'Entity Framework pour les nouveaux projets à l'aide des outils Entity Framework ajoute automatiquement le EF 6.1.3 package NuGet. Vous pouvez manuellement installer ou mettre à niveau vers n'importe quel package NuGet d'EF disponible en ligne.
- Par défaut, l'instance de SQL Server disponible avec cette version de Visual Studio est une instance de base de données locale appelée MSSQLLocalDB. La section serveur de la chaîne de connexion que vous devez utiliser est « (localdb)\MSSQLLocalDB ». N'oubliez pas d'utiliser un préfixe de chaîne textuelle @ ou doubles barres obliques inverses «\\» lorsque vous spécifiez une chaîne de connexion dans le code C#.

## Visual Studio 2013

- Cette version de Visual Studio inclut et version antérieure du runtime et les outils Entity Framework. Il est recommandé de mettre à niveau vers Entity Framework Tools 6.1.3, à l'aide de [le programme d'installation](#) disponibles dans du Microsoft Download Center. Consultez [libère dernières](#) pour plus d'informations sur ces versions.
- Ajout d'Entity Framework pour les nouveaux projets à l'aide des outils Entity Framework mis à niveau ajoute automatiquement le EF 6.1.3 package NuGet. Vous pouvez manuellement installer ou mettre à niveau vers n'importe quel package NuGet d'EF disponible en ligne.
- Par défaut, l'instance de SQL Server disponible avec cette version de Visual Studio est une instance de base de données locale appelée MSSQLLocalDB. La section serveur de la chaîne de connexion que vous devez utiliser est « (localdb)\MSSQLLocalDB ». N'oubliez pas d'utiliser un préfixe de chaîne textuelle @ ou doubles barres obliques inverses «\\» lorsque vous spécifiez une chaîne de connexion dans le code C#.

## Visual Studio 2012

- Cette version de Visual Studio inclut et version antérieure du runtime et les outils Entity Framework. Il est recommandé de mettre à niveau vers Entity Framework Tools 6.1.3, à l'aide de [le programme d'installation](#) disponibles dans du Microsoft Download Center. Consultez [libère dernières](#) pour plus d'informations sur ces versions.
- Ajout d'Entity Framework pour les nouveaux projets à l'aide des outils Entity Framework mis à niveau ajoute automatiquement le EF 6.1.3 package NuGet. Vous pouvez manuellement installer ou mettre à niveau vers n'importe quel package NuGet d'EF disponible en ligne.
- Par défaut, l'instance de SQL Server disponible avec cette version de Visual Studio est une instance de base de données locale appelée v11.0. La section serveur de la chaîne de connexion que vous devez utiliser est « (localdb)\v11.0 ». N'oubliez pas d'utiliser un préfixe de chaîne textuelle @ ou doubles barres obliques inverses «\\» lorsque vous spécifiez une chaîne de connexion dans le code C#.

## Visual Studio 2010

- La version d'Entity Framework Tools est disponible avec cette version de Visual Studio n'est pas compatible avec le runtime Entity Framework 6 et ne peut pas être mis à niveau.
- Par défaut, les outils Entity Framework ajoute Entity Framework 4.0 à vos projets. Pour créer des applications à l'aide des versions les plus récentes d'EF, vous devez d'abord installer le [extension du Gestionnaire de Package NuGet](#).
- Par défaut, tous les génération de code dans la version des outils Entity Framework est basé sur EntityObject et Entity Framework 4. Nous vous recommandons de passer la génération de code doit être basé sur DbContext et Entity Framework 5, en installant les modèles de génération de code de DbContext pour [C#](#) ou [Visual Basic](#).
- Une fois que vous avez installé les extensions du Gestionnaire de Package NuGet, vous pouvez manuellement installer ou mettre à niveau vers n'importe quel package NuGet d'EF disponible en ligne et utiliser EF6 avec Code First, qui ne nécessite pas d'un concepteur.
- Par défaut, l'instance de SQL Server disponible avec cette version de Visual Studio est SQL Server Express nommée SQLEXPRESS. La section serveur de la chaîne de connexion que vous devez utiliser est ». \SQLEXPRESS ». N'oubliez pas d'utiliser un préfixe de chaîne textuelle @ ou doubles barres obliques inverses «\\» lorsque vous spécifiez une chaîne de connexion dans le code C#.

# Bien démarrer avec Entity Framework 6

07/11/2019 • 3 minutes to read • [Edit Online](#)

Ce guide contient un ensemble de liens vers des articles de documentation sélectionnés, des procédures pas à pas et des vidéos qui peuvent vous aider à démarrer rapidement.

## Notions de base

- [Obtenir Entity Framework](#)

Vous apprendrez ici à ajouter Entity Framework à vos applications. Par ailleurs, si vous voulez utiliser EF Designer, vérifiez qu'il est installé dans Visual Studio.

- [Création d'un modèle : Code First, EF Designer et les flux de travail EF](#)

Préférez-vous spécifier votre modèle EF en écrivant du code ou en traçant des zones et des lignes ? Allez-vous utiliser EF pour mapper vos objets à une base de données existante ou voulez-vous qu'EF crée une base de données adaptée à vos objets ? Vous découvrez ici deux approches différentes pour utiliser EF6 : EF Designer et Code First. Veillez à suivre la discussion et à regarder la vidéo présentant la différence.

- [Utilisation de DbContext](#)

DbContext est le premier et le plus important type EF dont vous avez besoin pour apprendre à utiliser Entity Framework. Il sert de tremplin pour les requêtes de base de données et effectue le suivi des changements que vous apportez aux objets afin qu'ils puissent être conservés dans la base de données.

- [Poser une question](#)

Découvrez comment obtenir de l'aide des experts et fournir vos propres réponses à la Communauté.

- [Contribuer](#)

Entity Framework 6 utilise un modèle de développement ouvert. Découvrez comment vous pouvez aider à améliorer EF en consultant notre dépôt GitHub.

## Ressources Code First

- [Code First ciblant un flux de travail de base de données existant](#)
- [Code First ciblant un nouveau flux de travail de base de données](#)
- [Mappage d'enums avec Code First](#)
- [Mappage de types spatiaux avec Code First](#)
- [Écriture de conventions Code First personnalisées](#)
- [Utilisation de la configuration Fluent Code First avec Visual Basic](#)
- [Migrations Code First](#)
- [Migrations Code First dans les environnements d'équipe](#)
- [Migrations Code First automatiques](#) (désormais déconseillées)

## Ressources EF Designer

- [Flux de travail Database First](#)
- [Flux de travail Model First](#)
- [Mappage d'enums](#)

- [Mappage de types spatiaux](#)
- [Mappage d'héritage de table par hiérarchie](#)
- [Mappage d'héritage de table par type](#)
- [Mappage de procédures stockées pour les mises à jour](#)
- [Mappage de procédures stockées pour la requête](#)
- [Fractionnement d'entité](#)
- [Fractionnement de table](#)
- [Requête de définition \(Avancé\)](#)
- [Fonctions table \(Avancé\)](#)

## Autres ressources

- [Requête et enregistrement asynchrones](#)
- [Liaison de données avec WinForms](#)
- [Liaison de données avec WPF](#)
- [Scénarios déconnectés avec des entités de suivi automatique](#) (désormais déconseillés)

# Obtenir Entity Framework

11/10/2019 • 4 minutes to read

Entity Framework se compose des outils EF pour Visual Studio et du runtime EF.

## Outils EF pour Visual Studio

Le Entity Framework Tools pour Visual Studio inclut le concepteur EF et l'Assistant Modèle EF et sont requis pour les flux de travail First et Model First. Les outils EF sont inclus dans toutes les versions récentes de Visual Studio. Si vous effectuez une installation personnalisée de Visual Studio, vous devez vous assurer que l'élément « outils Entity Framework 6 » est sélectionné en choisissant une charge de travail qui l'y ajoute ou en la sélectionnant en tant que composant individuel.

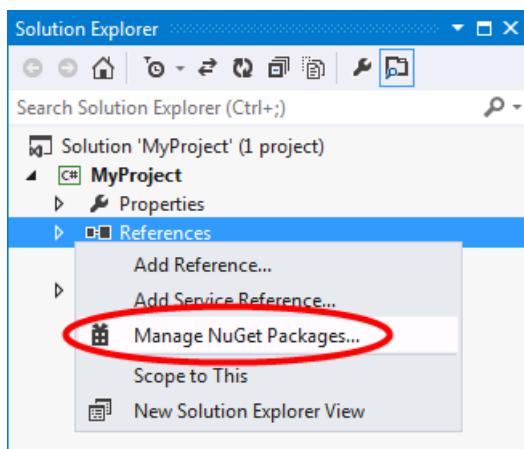
Pour certaines versions antérieures de Visual Studio, les outils EF mis à jour sont disponibles en téléchargement. Pour obtenir des conseils sur la façon d'obtenir la dernière version des outils EF disponibles pour votre version de Visual Studio, consultez les [versions de Visual Studio](#).

## Runtime EF

La dernière version de Entity Framework est disponible en tant que [package NuGet](#) de l'EntityFramework. Si vous n'êtes pas familiarisé avec le gestionnaire de package NuGet, nous vous invitons à lire la [vue d'ensemble de NuGet](#).

### Installation du package NuGet d'EF

Vous pouvez installer le package EntityFramework en cliquant avec le bouton droit sur le dossier **références** de votre projet et en sélectionnant **gérer les packages NuGet...**



### Installation à partir de la console du gestionnaire de package

Vous pouvez également installer EntityFramework en exécutant la commande suivante dans la console du [Gestionnaire de package](#).

```
Install-Package EntityFramework
```

## Installation d'une version spécifique d'EF

À partir d'EF 4.1, les nouvelles versions du runtime EF ont été publiées en tant que [package NuGet](#) de l'EntityFramework. N'importe laquelle de ces versions peut être ajoutée à un projet basé sur le .NET Framework

en exécutant la commande suivante dans la [console du gestionnaire de package](#) de Visual Studio :

```
Install-Package EntityFramework -Version <number>
```

Notez que `<number>` représente la version spécifique d'EF à installer. Par exemple, 6.2.0 est la version du numéro pour EF 6.2.

Les exécutions EF antérieures à 4.1 faisaient partie de .NET Framework et ne peuvent pas être installées séparément.

### Installation de la dernière version préliminaire

Les méthodes ci-dessus vous offriront la toute dernière version de Entity Framework entièrement prise en charge. Il existe souvent des versions préliminaires de Entity Framework disponibles que nous aimerais pouvoir essayer et nous envoyer vos commentaires.

Pour installer la dernière version préliminaire d'EntityFramework, vous pouvez sélectionner **inclure la version préliminaire** dans la fenêtre gérer les packages NuGet. Si aucune version préliminaire n'est disponible, vous obtiendrez automatiquement la dernière version entièrement prise en charge de Entity Framework.



Vous pouvez également exécuter la commande suivante dans la console du [Gestionnaire de package](#).

```
Install-Package EntityFramework -Pre
```

# Utilisation de DbContext

13/09/2018 • 7 minutes to read

Pour utiliser Entity Framework pour interroger, insérer, mettre à jour et supprimer des données à l'aide d'objets .NET, vous devez d'abord [créer un modèle](#) qui mappe les entités et les relations qui sont définies dans votre modèle à des tables dans une base de données.

Une fois que vous avez un modèle, la classe principale, votre application interagit avec est `System.Data.Entity.DbContext` (souvent appelé la classe de contexte). Vous pouvez utiliser un DbContext associé à un modèle pour :

- Écrire et exécuter des requêtes
- Matérialiser les résultats de la requête en tant qu'objets d'entité
- Le suivi des modifications apportées à ces objets.
- Conserver les modifications de l'objet dans la base de données
- Lier des objets en mémoire pour les contrôles d'interface utilisateur

Cette page fournit des conseils sur la gestion de la classe de contexte.

## Définition d'une classe DbContext dérivée

La méthode recommandée pour travailler avec contexte consiste à définir une classe qui dérive de DbContext et expose les propriétés DbSet qui représentent des collections d'entités dans le contexte spécifiées. Si vous travaillez avec le Concepteur EF, le contexte sera généré pour vous. Si vous utilisez Code First, vous allez généralement écrire le contexte vous-même.

```
public class ProductContext : DbContext
{
    public DbSet<Category> Categories { get; set; }
    public DbSet<Product> Products { get; set; }
}
```

Une fois que vous disposez d'un contexte, vous interroger, ajouter (à l'aide de `Add` ou `Attach` méthodes) ou supprimer (à l'aide de `Remove`) les entités dans le contexte via ces propriétés. L'accès à un `DbSet` propriété sur un objet de contexte représentent une requête initiale qui retourne toutes les entités du type spécifié. Notez que l'accès à une propriété pas exécute la requête. Une requête est exécutée lorsque :

- elle est énumérée par une instruction `foreach` (C#) ou `For Each` (Visual Basic) ;
- Elle est énumérée par une opération de collection comme `ToDictionary`, `ToDictionary`, ou `ToList`.
- Les opérateurs LINQ, tels que `First` ou `Any` sont spécifiés dans la partie la plus extérieure de la requête.
- Une des méthodes suivantes sont appelées : le `Load` méthode d'extension, `DbEntityEntry.Reload`, `Database.ExecuteSqlCommand`, et `DbSet<T>.Find`, si une entité avec la clé spécifiée se trouve pas déjà chargée dans le contexte.

## Durée de vie

La durée de vie du contexte commence lorsque l'instance est créée et se termine lorsque l'instance est supprimée ou opération de garbage collection. Utilisez **à l'aide de** si vous voulez que toutes les ressources contrôlées par le contexte soient supprimées à la fin du bloc. Lorsque vous utilisez **à l'aide de**, le compilateur crée automatiquement un bloc try/finally et appelle la méthode dispose dans le **enfin** bloc.

```
public void UseProducts()
{
    using (var context = new ProductContext())
    {
        // Perform data access using the context
    }
}
```

Voici quelques instructions générales lorsque vous décidez de la durée de vie du contexte :

- Lorsque vous travaillez avec les applications Web, utilisez une instance de contexte par demande.
- Lorsque vous travaillez avec Windows Presentation Foundation (WPF) ou Windows Forms, utilisez une instance de contexte par formulaire. Cela vous permet d'utiliser des fonctionnalités de suivi des modifications fournies par ce contexte.
- Si l'instance de contexte est créée par un conteneur d'injection de dépendance, il est généralement la responsabilité du conteneur de supprimer le contexte.
- Si le contexte est créé dans le code d'application, pensez à supprimer le contexte lorsqu'il n'est plus nécessaire.
- Lorsque vous travaillez avec le contexte d'exécution longue, considérez les points suivants :
  - Comme vous chargez des objets et leurs références dans la mémoire, du contexte de la consommation de mémoire peut augmenter rapidement. Cela peut provoquer des problèmes de performances.
  - Le contexte n'est pas thread-safe, par conséquent, il ne doit pas être partagé entre plusieurs threads travaillant simultanément sur celui-ci.
  - Si une exception provoque le contexte dans un état irrécupérable, l'application entière peut mettre fin.
  - Les risques liés à l'exécution dans un état d'accès concurrentiel augmentent à mesure que l'intervalle entre le moment où les données sont interrogées et mises à jour s'allonge.

## Connexions

Par défaut, le contexte gère les connexions à la base de données. Le contexte s'ouvre et ferme les connexions en fonction des besoins. Par exemple, le contexte ouvre une connexion pour exécuter une requête, puis ferme la connexion lorsque tous les jeux de résultats ont été traitées.

Dans certains cas, vous souhaiterez avoir davantage de contrôle sur l'ouverture et la fermeture des connexions. Par exemple, lorsque vous travaillez avec SQL Server Compact, il est souvent recommandé de maintenir une connexion à la base de données ouverte distincte pour la durée de vie de l'application pour améliorer les performances. Vous pouvez gérer ce processus manuellement à l'aide de la propriété `Connection`.

# Relations, propriétés de navigation et clés étrangères

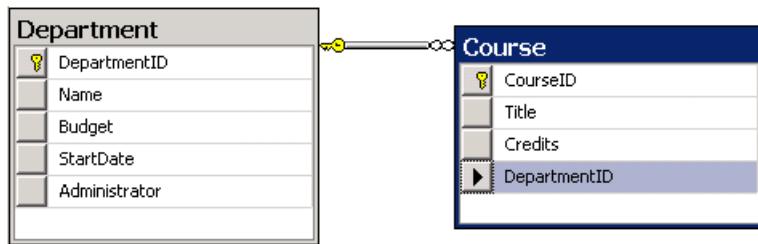
07/11/2019 • 17 minutes to read

Cette rubrique donne une vue d'ensemble de la façon dont Entity Framework gère les relations entre les entités. Il fournit également des conseils sur la façon de mapper et manipuler les relations.

## Relations dans EF

Dans les bases de données relationnelles, les relations (également appelées associations) entre les tables sont définies via des clés étrangères. Une clé étrangère (FK) est une colonne ou une combinaison de colonnes utilisée pour établir et appliquer un lien entre les données de deux tables. Il existe généralement trois types de relations : un-à-un, un-à-plusieurs et plusieurs-à-plusieurs. Dans une relation un-à-plusieurs, la clé étrangère est définie sur la table qui représente la fin de la relation. La relation plusieurs-à-plusieurs implique la définition d'une troisième table (appelée table de jonction ou de jointure), dont la clé primaire est composée des clés étrangères des deux tables associées. Dans une relation un-à-un, la clé primaire agit en outre comme une clé étrangère et il n'existe pas de colonne clé étrangère distincte pour l'une ou l'autre des tables.

L'illustration suivante montre deux tables qui participent à une relation un-à-plusieurs. La table **course** est la table dépendante, car elle contient la colonne **DepartmentID** qui la lie à la table **Department**.



Dans Entity Framework, une entité peut être associée à d'autres entités par le biais d'une association ou d'une relation. Chaque relation contient deux extrémités qui décrivent le type d'entité et la multiplicité du type (un, zéro-ou-un ou plusieurs) pour les deux entités de cette relation. La relation peut être régie par une contrainte référentielle, qui décrit la terminaison de la relation comme un rôle principal et qui est un rôle dépendant.

Les propriétés de navigation permettent de parcourir une association entre deux types d'entités. Chaque objet peut avoir une propriété de navigation pour chaque relation à laquelle il participe. Les propriétés de navigation vous permettent de parcourir et de gérer les relations dans les deux directions, en retournant un objet de référence (si la multiplicité est un ou zéro-ou-un) ou une collection (si la multiplicité est plusieurs). Vous pouvez également choisir d'avoir une navigation unidirectionnelle, auquel cas vous définissez la propriété de navigation sur un seul des types qui participe à la relation et non sur les deux.

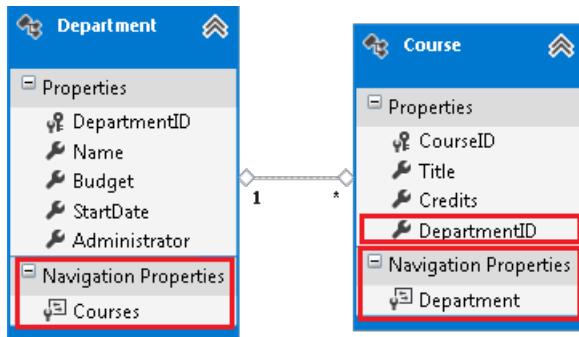
Il est recommandé d'inclure dans le modèle des propriétés qui mappent à des clés étrangères dans la base de données. Lorsque les propriétés de clé étrangère sont incluses, vous pouvez créer ou modifier une relation en changeant la valeur de clé étrangère d'un objet dépendant. Ce type d'association s'appelle une association de clé étrangère. L'utilisation de clés étrangères est encore plus essentielle lorsque vous travaillez avec des entités déconnectées. Notez que lors de l'utilisation de 1 à 1 ou de 1 à 0.. 1 les relations, il n'existe pas de colonne clé étrangère distincte, la propriété de clé primaire agit comme clé étrangère et est toujours incluse dans le modèle.

Lorsque les colonnes clés étrangères ne sont pas incluses dans le modèle, les informations d'association sont gérées en tant qu'objet indépendant. Les relations sont suivies via des références d'objet plutôt que des propriétés de clé étrangère. Ce type d'association s'appelle une *Association indépendante*. La méthode la plus courante pour modifier une *Association indépendante* est de modifier les propriétés de navigation générées pour chaque entité qui participe à l'Association.

Vous pouvez choisir d'utiliser un type d'association dans votre modèle ou les deux à la fois. Toutefois, si vous avez une relation plusieurs-à-plusieurs pure qui est connectée par une table de jointure qui contient uniquement des clés étrangères, EF utilise une association indépendante pour gérer une relation plusieurs-à-plusieurs.

L'illustration suivante montre un modèle conceptuel créé avec l'Entity Framework Designer. Le modèle contient deux entités qui participent à une relation un-à-plusieurs. Les deux entités ont des propriétés de navigation.

**Course** est l'entité dépendante et la propriété de clé étrangère **DepartmentID** est définie.



L'extrait de code suivant montre le modèle qui a été créé avec Code First.

```
public class Course
{
    public int CourseID { get; set; }
    public string Title { get; set; }
    public int Credits { get; set; }
    public int DepartmentID { get; set; }
    public virtual Department Department { get; set; }
}

public class Department
{
    public Department()
    {
        this.Courses = new HashSet<Course>();
    }
    public int DepartmentID { get; set; }
    public string Name { get; set; }
    public decimal Budget { get; set; }
    public DateTime StartDate { get; set; }
    public int? Administrator { get; set; }
    public virtual ICollection<Course> Courses { get; set; }
}
```

## Configuration ou mappage des relations

Le reste de cette page explique comment accéder aux données et les manipuler à l'aide de relations. Pour plus d'informations sur la configuration des relations dans votre modèle, consultez les pages suivantes.

- Pour configurer les relations dans Code First, consultez [Annotations de données](#) et [API Fluent – relations](#).
- Pour configurer les relations à l'aide de la Entity Framework Designer, consultez [relations avec le concepteur EF](#).

## Création et modification de relations

Dans une *Association de clé étrangère*, lorsque vous modifiez la relation, l'état d'un objet dépendant avec un état de  `EntityState.Unchanged` passe à  `EntityState.Modified`. Dans une relation indépendante, la modification de la relation ne met pas à jour l'état de l'objet dépendant.

Les exemples suivants montrent comment utiliser les propriétés de clé étrangère et les propriétés de navigation

pour associer les objets connexes. Avec les associations de clé étrangère, vous pouvez utiliser l'une ou l'autre méthode pour modifier, créer ou modifier des relations. Avec les associations indépendantes, vous ne pouvez pas utiliser la propriété de clé étrangère.

- En affectant une nouvelle valeur à une propriété de clé étrangère, comme dans l'exemple suivant.

```
course.DepartmentID = newCourse.DepartmentID;
```

- Le code suivant supprime une relation en affectant à la clé étrangère la **valeur null**. Notez que la propriété de clé étrangère doit avoir la valeur null.

```
course.DepartmentID = null;
```

#### NOTE

Si la référence est à l'état ajouté (dans cet exemple, l'objet `course`), la propriété de navigation de référence n'est pas synchronisée avec les valeurs de clé d'un nouvel objet tant que `SaveChanges` n'est pas appelé. La synchronisation n'a pas lieu, car le contexte de l'objet ne contient pas de clés permanentes pour les objets ajoutés tant qu'ils ne sont pas enregistrés. Si vous devez avoir de nouveaux objets synchronisés entièrement dès que vous définissez la relation, utilisez l'une des méthodes suivantes. \*

- En affectant un nouvel objet à une propriété de navigation. Le code suivant crée une relation entre un cours et un `department`. Si les objets sont attachés au contexte, le `course` est également ajouté à la collection de `department.Courses`, et la propriété de clé étrangère correspondante sur l'objet `course` est définie sur la valeur de propriété de clé du service.

```
course.Department = department;
```

- Pour supprimer la relation, affectez à la propriété de navigation la valeur `null`. Si vous utilisez Entity Framework qui est basé sur .NET 4,0, la terminaison connexe doit être chargée avant de lui affecter la valeur null. Exemple :

```
context.Entry(course).Reference(c => c.Department).Load();
course.Department = null;
```

À partir de Entity Framework 5,0, basé sur .NET 4,5, vous pouvez définir la relation sur null sans charger la terminaison connexe. Vous pouvez également définir la valeur actuelle sur null à l'aide de la méthode suivante.

```
context.Entry(course).Reference(c => c.Department).CurrentValue = null;
```

- En supprimant ou en ajoutant un objet dans une collection d'entités. Par exemple, vous pouvez ajouter un objet de type `Course` à la collection de `department.Courses`. Cette opération crée une relation entre un `cours` particulier et un `department` particulier. Si les objets sont attachés au contexte, la référence de service et la propriété de clé étrangère sur l'objet `course` sont définies sur le `department` approprié.

```
department.Courses.Add(newCourse);
```

- En utilisant la méthode `ChangeRelationshipState` pour modifier l'état de la relation spécifiée entre deux objets d'entité. Cette méthode est généralement utilisée lorsque vous travaillez avec des applications

multicouches et une *Association indépendante* (elle ne peut pas être utilisée avec une association de clé étrangère). En outre, pour utiliser cette méthode, vous devez dérouler jusqu'à `ObjectContext`, comme illustré dans l'exemple ci-dessous.

Dans l'exemple suivant, il existe une relation plusieurs-à-plusieurs entre les instructeurs et les cours. L'appel de la méthode `ChangeRelationshipState` et la transmission du paramètre  `EntityState.Added` permettent à l'`SchoolContext` de savoir qu'une relation a été ajoutée entre les deux objets :

```
((IObjectContextAdapter)context).ObjectContext.  
    ObjectStateManager.  
    ChangeRelationshipState(course, instructor, c => c.Instructor, EntityState.Added);
```

Notez que, si vous mettez à jour (et non pas simplement) une relation, vous devez supprimer l'ancienne relation après avoir ajouté la nouvelle.

```
((IObjectContextAdapter)context).ObjectContext.  
    ObjectStateManager.  
    ChangeRelationshipState(course, oldInstructor, c => c.Instructor, EntityState.Deleted);
```

## Synchronisation des modifications entre les clés étrangères et les propriétés de navigation

Lorsque vous modifiez la relation des objets attachés au contexte à l'aide de l'une des méthodes décrites ci-dessus, Entity Framework doit conserver les clés étrangères, les références et les collections synchronisées. Entity Framework gère automatiquement cette synchronisation (également appelée « correction des relations ») pour les entités POCO avec proxys. Pour plus d'informations, consultez [utilisation des proxies](#).

Si vous utilisez des entités POCO sans proxys, vous devez vous assurer que la méthode **DetectChanges** est appelée pour synchroniser les objets connexes dans le contexte. Notez que les API suivantes déclenchent automatiquement un appel **DetectChanges** .

- `DbSet.Add`
- `DbSet.AddRange`
- `DbSet.Remove`
- `DbSet.RemoveRange`
- `DbSet.Find`
- `DbSet.Local`
- `DbContext.SaveChanges`
- `DbSet.Attach`
- `DbContext.GetValidationErrors`
- `DbContext.Entry`
- `DbChangeTracker.Entries`
- Exécution d'une requête LINQ sur un `DbSet`

## Chargement d'objets connexes

Dans Entity Framework vous utilisez couramment des propriétés de navigation pour charger des entités associées à l'entité renvoyée par l'Association définie. Pour plus d'informations, consultez [chargement d'objets connexes](#).

#### NOTE

Dans une association de clé étrangère, lorsque vous chargez une terminaison connexe d'un objet dépendant, l'objet connexe est chargé en fonction de la valeur de clé étrangère du dépendant actuellement en mémoire :

```
// Get the course where currently DepartmentID = 2.  
Course course2 = context.Courses.First(c=>c.DepartmentID == 2);  
  
// Use DepartmentID foreign key property  
// to change the association.  
course2.DepartmentID = 3;  
  
// Load the related Department where DepartmentID = 3  
context.Entry(course).Reference(c => c.Department).Load();
```

Dans une association indépendante, la terminaison connexe d'un objet dépendant est interrogée en fonction de la valeur de clé étrangère actuellement présente dans la base de données. Toutefois, si la relation a été modifiée et que la propriété de référence sur l'objet dépendant pointe vers un objet principal différent qui est chargé dans le contexte de l'objet, Entity Framework essaiera de créer une relation, car elle est définie sur le client.

## Gestion de l'accès concurrentiel

Dans les associations de clé étrangère et indépendantes, les contrôles d'accès concurrentiel sont basés sur les clés d'entité et d'autres propriétés d'entité définies dans le modèle. Lorsque vous utilisez le concepteur EF pour créer un modèle, affectez la valeur **fixed** à l'attribut `ConcurrencyMode` pour spécifier que la propriété doit être vérifiée pour l'accès concurrentiel. Quand vous utilisez Code First pour définir un modèle, utilisez l'annotation `ConcurrencyCheck` sur les propriétés dont vous souhaitez vérifier l'accès concurrentiel. Lorsque vous utilisez Code First vous pouvez également utiliser l'annotation `TimeStamp` pour spécifier que la propriété doit être vérifiée pour l'accès concurrentiel. Vous ne pouvez avoir qu'une seule propriété `Timestamp` dans une classe donnée. Code First mappe cette propriété à un champ qui n'accepte pas les valeurs `NULL` dans la base de données.

Nous vous recommandons de toujours utiliser l'Association de clé étrangère lorsque vous travaillez avec des entités qui participent à la vérification et à la résolution de l'accès concurrentiel.

Pour plus d'informations, consultez [gestion des conflits d'accès concurrentiel](#).

## Utilisation des touches qui se chevauchent

Des clés qui se chevauchent sont des clés composites qui ont certaines propriétés en commun dans l'entité qu'elles constituent. Une association indépendante ne peut pas contenir de clés qui se chevauchent. Pour modifier une association de clé étrangère qui comporte des clés qui se chevauchent, nous vous conseillons de modifier les valeurs de clé étrangère plutôt que d'utiliser les références d'objet.

# Requête asynchrone et enregistrement

11/10/2019 • 11 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

EF6 a introduit la prise en charge de la requête asynchrone et l'enregistre à l'aide des [Mots clés Async et await](#) qui ont été introduits dans .net 4,5. Même si toutes les applications ne peuvent pas tirer parti de l'asynchronie, elles peuvent être utilisées pour améliorer la réactivité du client et l'évolutivité du serveur lors du traitement des tâches de longue durée, réseau ou liées aux e/s.

## Quand vraiment utiliser Async

L'objectif de cette procédure pas à pas est d'introduire les concepts Async d'une manière qui facilite l'observation de la différence entre l'exécution asynchrone et synchrone du programme. Cette procédure pas à pas n'est pas destinée à illustrer les principaux scénarios où la programmation asynchrone offre des avantages.

La programmation asynchrone est principalement axée sur la libération du thread managé actuel (thread exécutant du code .NET) pour effectuer d'autres tâches pendant qu'il attend une opération qui ne nécessite pas de temps de calcul d'un thread managé. Par exemple, pendant que le moteur de base de données traite une requête, il n'y a rien à faire par le code .NET.

Dans les applications clientes (WinForms, WPF, etc.), le thread actuel peut être utilisé pour maintenir la réactivité de l'interface utilisateur pendant l'exécution de l'opération asynchrone. Dans les applications serveur (ASP.NET, etc.), le thread peut être utilisé pour traiter d'autres demandes entrantes, ce qui peut réduire l'utilisation de la mémoire et/ou augmenter le débit du serveur.

Dans la plupart des applications utilisant Async n'auront pas d'avantages significatifs et même pourrait être nuisible. Utilisez les tests, le profilage et le bon sens pour mesurer l'impact de Async dans votre scénario particulier avant de le valider.

Voici d'autres ressources pour en savoir plus sur Async :

- [Vue d'ensemble de Brandon Bray de Async/await dans .NET 4,5](#)
- Pages de [programmation asynchrones](#) dans MSDN Library
- [Comment créer des applications Web ASP.net à l'aide de Async](#) (comprend une démonstration du débit accru du serveur)

## Créer le modèle

Nous utiliserons le flux de travail [Code First](#) pour créer notre modèle et générer la base de données. Toutefois, les fonctionnalités asynchrones fonctionnent avec tous les modèles EF, y compris ceux créés avec le concepteur EF.

- Créez une application console et l'appeler **AsyncDemo**
- Ajouter le package NuGet EntityFramework
  - Dans Explorateur de solutions, cliquez avec le bouton droit sur le projet **AsyncDemo**
  - Sélectionnez **gérer les packages NuGet...**
  - Dans la boîte de dialogue gérer les packages NuGet, sélectionnez l'onglet **en ligne** et choisissez le

package **EntityFramework** .

- Cliquez sur **installer**
- Ajoutez une classe **Model.cs** avec l'implémentation suivante

```
using System.Collections.Generic;
using System.Data.Entity;

namespace AsyncDemo
{
    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
}
```

## Créer un programme synchrone

Maintenant que nous disposons d'un modèle EF, nous allons écrire du code qui l'utilise pour effectuer un accès aux données.

- Remplacez le contenu de **Program.cs** par le code suivant :

```

using System;
using System.Linq;

namespace AsyncDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            PerformDatabaseOperations();

            Console.WriteLine("Quote of the day");
            Console.WriteLine(" Don't worry about the world coming to an end today... ");
            Console.WriteLine(" It's already tomorrow in Australia.");

            Console.WriteLine();
            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }

        public static void PerformDatabaseOperations()
        {
            using (var db = new BloggingContext())
            {
                // Create a new blog and save it
                db.Blogs.Add(new Blog
                {
                    Name = "Test Blog #" + (db.Blogs.Count() + 1)
                });
                Console.WriteLine("Calling SaveChanges.");
                db.SaveChanges();
                Console.WriteLine("SaveChanges completed.");

                // Query for all blogs ordered by name
                Console.WriteLine("Executing query.");
                var blogs = (from b in db.Blogs
                            orderby b.Name
                            select b).ToList();

                // Write all blogs out to Console
                Console.WriteLine("Query completed with following results:");
                foreach (var blog in blogs)
                {
                    Console.WriteLine(" " + blog.Name);
                }
            }
        }
    }
}

```

Ce code appelle la méthode **PerformDatabaseOperations** qui enregistre un nouveau **blog** dans la base de données, puis récupère tous les **blogs** de la base de données et les imprime sur la **console**. Après cela, le programme écrit un guillemet du jour sur la **console**.

Étant donné que le code est synchrone, nous pouvons observer le déroulement de l'exécution suivant lorsque nous exécutons le programme :

1. **SaveChanges** commence à envoyer le nouveau **blog** à la base de données
2. **SaveChanges** se termine
3. La requête de tous les **blogs** est envoyée à la base de données
4. La requête retourne et les résultats sont écrits dans la **console**
5. Le devis du jour est écrit dans la **console**

```
Calling SaveChanges.  
SaveChanges completed.  
Executing query.  
Query completed with following results:  
- Test Blog #1  
Quote of the day  
Don't worry about the world coming to an end today...  
It's already tomorrow in Australia.  
Press any key to exit...
```

## Rendre asynchrone

Maintenant que notre programme est opérationnel, nous pouvons commencer à utiliser les nouveaux mots clés `Async` et `await`. Nous avons apporté les modifications suivantes à `Program.cs`

1. Ligne 2 : L'instruction `using` de l'espace de noms **System. Data. Entity** nous donne accès aux méthodes d'extension EF Async.
2. Ligne 4 : L'instruction `using` pour l'espace de noms **System. Threading. Tasks** nous permet d'utiliser le type de **tâche**.
3. Ligne 12 & 18 : Nous effectuons une capture en tant que tâche qui surveille la progression de **PerformSomeDatabaseOperations** (ligne 12), puis bloquent l'exécution du programme pour que cette tâche se termine une fois que tout le travail du programme est terminé (ligne 18).
4. Ligne 25 : Nous mettons à jour **PerformSomeDatabaseOperations** pour qu'elle soit marquée comme **Async** et retourne une **tâche**.
5. Ligne 35 : Nous appelons maintenant la version Async de `SaveChanges` et en attendant son achèvement.
6. Ligne 42 : Nous appelons maintenant la version Async de `ToList` et en attendant le résultat.

Pour obtenir la liste complète des méthodes d'extension disponibles dans l'espace de noms `System. Data. Entity`, reportez-vous à la classe `QueryableExtensions`. *Vous devez également ajouter « using System. Data. Entity » à vos instructions using.*

```

using System;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;

namespace AsyncDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            var task = PerformDatabaseOperations();

            Console.WriteLine("Quote of the day");
            Console.WriteLine(" Don't worry about the world coming to an end today... ");
            Console.WriteLine(" It's already tomorrow in Australia.");

            task.Wait();

            Console.WriteLine();
            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }

        public static async Task PerformDatabaseOperations()
        {
            using (var db = new BloggingContext())
            {
                // Create a new blog and save it
                db.Blogs.Add(new Blog
                {
                    Name = "Test Blog #" + (db.Blogs.Count() + 1)
                });
                Console.WriteLine("Calling SaveChanges.");
                await db.SaveChangesAsync();
                Console.WriteLine("SaveChanges completed.");

                // Query for all blogs ordered by name
                Console.WriteLine("Executing query.");
                var blogs = await (from b in db.Blogs
                                   orderby b.Name
                                   select b).ToListAsync();

                // Write all blogs out to Console
                Console.WriteLine("Query completed with following results:");
                foreach (var blog in blogs)
                {
                    Console.WriteLine(" - " + blog.Name);
                }
            }
        }
    }
}

```

Maintenant que le code est asynchrone, nous pouvons observer un autre workflow d'exécution lorsque nous exécutons le programme :

1. **SaveChanges** commence à envoyer le nouveau **blog** à la base de données

*Once la commande est envoyée à la base de données, aucune durée de calcul supplémentaire n'est nécessaire sur le thread managé actuel. La méthode **PerformDatabaseOperations** retourne (même si elle n'a pas fini de s'exécuter) et le déroulement du programme dans la méthode main se poursuit.*

2. **Le devis du jour est écrit dans la console**

*Since il n'y a plus de travail à effectuer dans la méthode main, le thread managé est bloqué sur l'appel d'attente jusqu'à ce que l'opération de base de données se termine. Une fois l'opération terminée, le reste de*

notre **PerformDatabaseOperations** est exécuté.

3. **SaveChanges** se termine
4. La requête de tous les **blogs** est envoyée à la base de données  
*Again, le thread managé est libre d'effectuer d'autres tâches pendant que la requête est traitée dans la base de données. Étant donné que toutes les autres exécutions sont terminées, le thread s'arrêtera simplement sur l'appel d'attente.*
5. La requête retourne et les résultats sont écrits dans la **console**



```
Calling SaveChanges.
Quote of the day
Don't worry about the world coming to an end today...
It's already tomorrow in Australia.
SaveChanges completed.
Executing query.
Query completed with following results:
- Test Blog #1
- Test Blog #2

Press any key to exit...
```

## Le

Nous avons maintenant vu combien il est facile d'utiliser les méthodes asynchrones d'EF. Bien que les avantages de Async ne soient pas très évidents avec une application console simple, ces mêmes stratégies peuvent être appliquées dans les situations où des activités longues ou liées au réseau pourraient bloquer l'application, ou entraîner l'utilisation d'un grand nombre de threads Augmentez l'encombrement mémoire.

# Configuration basée sur le code

17/04/2019 • 8 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Configuration d'une application Entity Framework peut être spécifiée dans un fichier de configuration (app.config/web.config) ou via le code. Ce dernier est appelé configuration basée sur le code.

Configuration dans un fichier de configuration est décrite dans un [article distinct](#). Le fichier de configuration est prioritaire sur la configuration basée sur le code. En d'autres termes, si une option de configuration est définie dans le code et dans le fichier de configuration, le paramètre dans le fichier de configuration est utilisé.

## À l'aide de DbConfiguration

Configuration par le code dans EF6 et versions ultérieures est obtenue en créant une sous-classe de System.Data.Entity.Config.DbConfiguration. Les instructions suivantes doivent être suivies lors du sous-classement DbConfiguration :

- Créer qu'une seule classe DbConfiguration pour votre application. Cette classe spécifie les paramètres à l'échelle du domaine d'application.
- Placez votre classe DbConfiguration dans le même assembly que votre classe DbContext. (Consultez la *DbConfiguration déplacement* section si vous souhaitez modifier cela.)
- Donnez à votre classe DbConfiguration un constructeur sans paramètre public.
- Définir les options de configuration en appelant des méthodes DbConfiguration protégées à partir de ce constructeur.

Les recommandations suivantes permet d'EF découvrir et d'utiliser votre configuration automatiquement par les deux outils qui a besoin d'accéder à votre modèle et quand votre application est exécutée.

## Exemple

Une classe dérivée de DbConfiguration peut ressembler à ceci :

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.SqlServer;

namespace MyNamespace
{
    public class MyConfiguration : DbConfiguration
    {
        public MyConfiguration()
        {
            SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
            SetDefaultConnectionFactory(new LocalDbConnectionFactory("mssqllocaldb"));
        }
    }
}
```

Cette classe définit EF pour utiliser la stratégie d'exécution de SQL Azure - réessayer automatiquement les opérations de base de données ayant échoué - et à utiliser la base de données locale pour les bases de données créées par convention à partir de Code First.

## Déplacement DbConfiguration

Il existe des cas où il n'est pas possible de placer votre classe DbConfiguration dans le même assembly que votre classe DbContext. Par exemple, peut avoir deux classes DbContext dans des assemblies différents. Il existe deux options pour gérer cela.

La première option consiste à utiliser le fichier de configuration pour spécifier l'instance DbConfiguration à utiliser. Pour ce faire, définissez l'attribut codeConfigurationType de la section entityFramework. Exemple :

```
<entityFramework codeConfigurationType="MyNamespace.MyDbConfiguration, MyAssembly">
    ...
</entityFramework>
```

La valeur de codeConfigurationType doit être l'assembly et le nom qualifié d'espace de noms de votre classe DbConfiguration.

La deuxième option consiste à placer des DbConfigurationTypeAttribute sur votre classe de contexte. Exemple :

```
[DbConfigurationType(typeof(MyDbConfiguration))]
public class MyContextContext : DbContext
{
}
```

La valeur passée à l'attribut peut être votre type DbConfiguration - comme indiqué ci-dessus - ou l'assembly et l'espace de noms de chaîne de nom de type qualifié. Exemple :

```
[DbConfigurationType("MyNamespace.MyDbConfiguration, MyAssembly")]
public class MyContextContext : DbContext
{
}
```

## Définition explicite de DbConfiguration

Il existe certaines situations où la configuration peut être nécessaire avant que n'importe quel type DbContext a été utilisé. Voici quelques exemples :

- À l'aide de DbModelBuilder pour générer un modèle sans un contexte
- À l'aide d'un autre code d'utilitaire/framework qui utilise un DbContext où ce contexte est utilisé avant votre contexte de l'application est utilisée.

Dans de telles situations EF ne peut pas découvrir automatiquement de la configuration et à la place effectuez l'une des opérations suivantes :

- Définissez le type de DbConfiguration dans le fichier de configuration, comme décrit dans la *DbConfiguration déplacement* section ci-dessus
- Appelle la méthode DbConfiguration.SetConfiguration statique au cours du démarrage de l'application

## Substitution de DbConfiguration

Il existe certaines situations où vous devez remplacer la jeu de configuration dans la DbConfiguration. Cela n'est pas généralement effectuée par les développeurs d'applications, mais plutôt par les fournisseurs tiers et les plug-

ins qui ne peut pas utiliser une classe dérivée de DbConfiguration.

Pour ce faire, Entity Framework permet à inscrire un gestionnaire d'événements qui permettre modifier la configuration existante juste avant qu'il est verrouillé. Il fournit également une méthode sugar spécifiquement pour remplacer n'importe quel service retourné par le localisateur de service EF. Voici comment elle est destinée à être utilisée :

- Au démarrage de l'application (avant EF est utilisé) le plug-in ou le fournisseur doit s'inscrire à la méthode de gestionnaire d'événements pour cet événement. (Notez que cela doit se produire avant que l'application utilise Entity Framework).
- Le Gestionnaire d'événements effectue un appel à ReplaceService pour chaque service qui doit être remplacé.

Par exemple, pour remplacer IDbConnectionFactory et DbProviderService s'inscrire un gestionnaire de quelque chose comme suit :

```
DbConfiguration.Loaded += (_, a) =>
{
    a.ReplaceService<DbProviderServices>((s, k) => new MyProviderServices(s));
    a.ReplaceService<IDbConnectionFactory>((s, k) => new MyConnectionFactory(s));
};
```

Dans le code ci-dessus MyProviderServices et MyConnectionFactory représentent vos implémentations du service.

Vous pouvez également ajouter des gestionnaires de dépendance supplémentaire pour obtenir le même effet.

Notez que vous pouvez également encapsuler DbProviderFactory de cette façon, mais cela sera uniquement d'affecter les EF et pas les utilisations de DbProviderFactory en dehors d'Entity Framework. C'est pourquoi vous vous souhaitez probablement continuer à encapsuler DbProviderFactory que vous disposez avant.

Vous devez également garder à l'esprit les services que vous exécutez en externe pour votre application, par exemple, lors de l'exécution des migrations à partir de la Console du Gestionnaire de Package. Lorsque vous exécutez migrer à partir de la console, qu'il tente de trouver votre DbConfiguration. Toutefois, s'il faut ou non il obtiendra le service encapsulé dépend où il inscrit le Gestionnaire d'événements. Si elle est inscrite dans le cadre de la construction de votre DbConfiguration ensuite le code doit s'exécuter et le service doit obtenir encapsulé. Généralement, ce ne sera pas le cas et cela signifie que les outils ne sont pas obtenir le service encapsulé.

# Paramètres du fichier de configuration

22/08/2019 • 13 minutes to read

Entity Framework permet de spécifier un certain nombre de paramètres à partir du fichier de configuration. En général, respecte un principe de «Convention sur la configuration»: tous les paramètres abordés dans ce billet ont un comportement par défaut, vous n'avez plus à vous soucier de modifier le paramètre lorsque la valeur par défaut ne répond plus à vos besoins.

## Une alternative basée sur le code

Tous ces paramètres peuvent également être appliqués à l'aide de code. À partir de EF6, nous avons introduit une [configuration basée sur le code](#), qui offre un moyen central d'appliquer la configuration à partir du code. Avant EF6, la configuration peut toujours être appliquée à partir du code, mais vous devez utiliser différentes API pour configurer différentes zones. L'option fichier de configuration permet de modifier facilement ces paramètres lors du déploiement sans mettre à jour votre code.

## Section de configuration Entity Framework

À compter d'EF 4.1, vous pouvez définir l'initialiseur de base de données pour un contexte à l'aide de la section **appSettings** du fichier de configuration. Dans EF 4.3, nous avons introduit la section **entityFramework** personnalisée pour gérer les nouveaux paramètres. Entity Framework reconnaît toujours les initialiseurs de base de données définis à l'aide de l'ancien format, mais nous vous recommandons de passer au nouveau format dans la mesure du possible.

La section **EntityFramework** a été ajoutée automatiquement au fichier de configuration de votre projet lorsque vous avez installé le package NuGet entityFramework.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework"
      type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=4.3.0.0,
      Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  </configSections>
</configuration>
```

## Chaînes de connexion

Cette page fournit plus d'informations sur la façon dont Entity Framework détermine la base de données à utiliser, y compris les chaînes de connexion dans le fichier de configuration.

Les chaînes de connexion sont placées dans l'élément **connectionStrings** standard et ne nécessitent pas la section **entityFramework**.

Les modèles basés sur Code First utilisent des chaînes de connexion ADO.NET normales. Par exemple :

```
<connectionStrings>
  <add name="BlogContext"
    providerName="System.Data.SqlClient"
    connectionString="Server=.\SQLEXPRESS;Database=Blogging;Integrated Security=True;" />
</connectionStrings>
```

Les modèles basés sur le concepteur EF utilisent des chaînes de connexion EF spéciales. Par exemple :

```
<connectionStrings>
  <add name="BlogContext"
    connectionString=
      "metadata=
        res://*/BloggingModel.csdl|
        res://*/BloggingModel.ssdl|
        res://*/BloggingModel.msl;
      provider=System.Data.SqlClient;
      provider connection string=
        "data source=(localdb)\mssqllocaldb;
        initial catalog=Blogging;
        integrated security=True;
        multipleactiveresultsets=True;"
      providerName="System.Data.EntityClient" />
</connectionStrings>
```

## Type de configuration basée sur le code (EF6)

À partir de EF6, vous pouvez spécifier le DbConfiguration pour EF à utiliser pour la [configuration basée sur le code](#) dans votre application. Dans la plupart des cas, vous n'avez pas besoin de spécifier ce paramètre, car EF détecte automatiquement votre DbConfiguration. Pour plus d'informations sur le moment où vous devrez peut-être spécifier DbConfiguration dans votre fichier de configuration, consultez la section **déplacement de DbConfiguration** de la [configuration basée sur le code](#).

Pour définir un type de DbConfiguration, vous spécifiez le nom de type qualifié d'assembly dans l'élément **codeConfigurationType** .

### NOTE

Un nom qualifié d'assembly est le nom complet de l'espace de noms, suivi d'une virgule, de l'assembly dans lequel le type réside. Vous pouvez également spécifier la version, la culture et le jeton de clé publique de l'assembly.

```
<entityFramework codeConfigurationType="MyNamespace.MyConfiguration, MyAssembly">
</entityFramework>
```

## Fournisseurs de base de données EF (EF6)

Avant EF6, les parties spécifiques à Entity Framework d'un fournisseur de base de données devaient être incluses dans le cadre du fournisseur ADO.NET principal. À compter de EF6, les parties spécifiques d'EF sont désormais gérées et enregistrées séparément.

Normalement, vous n'avez pas besoin d'inscrire des fournisseurs vous-même. Cette opération est généralement effectuée par le fournisseur lorsque vous l'installez.

Les fournisseurs sont inscrits en incluant un élément **Provider** sous la section des **fournisseurs** enfant de la section **entityFramework** . Il existe deux attributs obligatoires pour une entrée de fournisseur:

- **invariantName** identifie le fournisseur ADO.net principal que ce fournisseur EF cible
- le **type** est le nom de type qualifié d'assembly de l'implémentation du fournisseur EF

#### NOTE

Un nom qualifié d'assembly est le nom complet de l'espace de noms, suivi d'une virgule, de l'assembly dans lequel le type réside. Vous pouvez également spécifier la version, la culture et le jeton de clé publique de l'assembly.

À titre d'exemple, voici l'entrée créée pour inscrire le fournisseur de SQL Server par défaut lorsque vous installez Entity Framework.

```
<providers>
  <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices,
    EntityFramework.SqlServer" />
</providers>
```

## Intercepteurs (EF 6.1 et versions ultérieures)

À compter d'EF 6.1, vous pouvez enregistrer des intercepteurs dans le fichier de configuration. Les intercepteurs vous permettent d'exécuter une logique supplémentaire quand EF effectue certaines opérations, telles que l'exécution de requêtes de base de données, l'ouverture de connexions, etc.

Les intercepteurs sont enregistrés en incluant un élément d'intercepteur sous la section d'intercepteurs enfant de la section **entityFramework**. Par exemple, la configuration suivante inscrit l'intercepteur **DatabaseLogger** intégré qui journalise toutes les opérations de base de données sur la console.

```
<interceptors>
  <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework"/>
</interceptors>
```

### Enregistrement des opérations de base de données dans un fichier (EF 6.1 et versions ultérieures)

L'inscription d'intercepteurs via le fichier de configuration est particulièrement utile lorsque vous souhaitez ajouter la journalisation à une application existante pour aider à déboguer un problème. **DatabaseLogger** prend en charge la journalisation dans un fichier en fournissant le nom de fichier en tant que paramètre de constructeur.

```
<interceptors>
  <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
    <parameters>
      <parameter value="C:\Temp\LogOutput.txt"/>
    </parameters>
  </interceptor>
</interceptors>
```

Par défaut, le fichier journal est remplacé par un nouveau fichier chaque fois que l'application démarre. Pour ajouter à la place le fichier journal s'il existe déjà, utilisez ce qui suit:

```
<interceptors>
  <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
    <parameters>
      <parameter value="C:\Temp\LogOutput.txt"/>
      <parameter value="true" type="System.Boolean"/>
    </parameters>
  </interceptor>
</interceptors>
```

Pour plus d'informations sur **DatabaseLogger** et l'inscription des intercepteurs, consultez le billet de blog EF 6.1: Activation de la journalisation sans recompilation.

## Fabrique de connexion par défaut Code First

La section Configuration vous permet de spécifier une fabrique de connexion par défaut que Code First deveze utiliser pour rechercher une base de données à utiliser pour un contexte. La fabrique de connexion par défaut est utilisée uniquement quand aucune chaîne de connexion n'a été ajoutée au fichier de configuration pour un contexte.

Lorsque vous avez installé le package NuGet d'EF, une fabrique de connexion par défaut a été inscrite qui pointe vers SQL Express ou la base de données locale, en fonction de celui que vous avez installé.

Pour définir une fabrique de connexion, vous spécifiez le nom de type qualifié d'assembly dans l'élément **defaultConnectionFactory**.

### NOTE

Un nom qualifié d'assembly est le nom complet de l'espace de noms, suivi d'une virgule, de l'assembly dans lequel le type réside. Vous pouvez également spécifier la version, la culture et le jeton de clé publique de l'assembly.

Voici un exemple de définition de votre propre fabrique de connexion par défaut:

```
<entityFramework>
  <defaultConnectionFactory type="MyNamespace.MyCustomFactory, MyAssembly"/>
</entityFramework>
```

L'exemple ci-dessus requiert que la fabrique personnalisée ait un constructeur sans paramètre. Si nécessaire, vous pouvez spécifier des paramètres de constructeur à l'aide de l'élément Parameters.

Par exemple, le SqlCeConnectionFactory, qui est inclus dans Entity Framework, requiert que vous fournissiez un nom invariant de fournisseur au constructeur. Le nom invariant du fournisseur identifie la version de SQL compact que vous souhaitez utiliser. La configuration suivante entraîne l'utilisation par défaut de SQL Compact version 4,0 pour les contextes.

```
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlCeConnectionFactory, EntityFramework">
    <parameters>
      <parameter value="System.Data.SqlClient.4.0" />
    </parameters>
  </defaultConnectionFactory>
</entityFramework>
```

Si vous ne définissez pas une fabrique de connexion par défaut, Code First utilise le SqlConnectionFactory .\SQLEXPRESS , pointant vers. SqlConnectionFactory possède également un constructeur qui vous permet de substituer des parties de la chaîne de connexion. Si vous souhaitez utiliser une instance de SQL Server autre que .\SQLEXPRESS vous pouvez utiliser ce constructeur pour définir le serveur.

La configuration suivante entraîne l'utilisation par Code First de **MyDatabaseServer** pour les contextes qui n'ont pas de chaîne de connexion explicite définie.

```
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework">
    <parameters>
      <parameter value="Data Source=MyDatabaseServer; Integrated Security=True;
MultipleActiveResultSets=True" />
    </parameters>
  </defaultConnectionFactory>
</entityFramework>
```

Par défaut, il est supposé que les arguments de constructeur sont de type chaîne. Vous pouvez utiliser l'attribut **type** pour modifier cela.

```
<parameter value="2" type="System.Int32" />
```

## Initialiseurs de base de données

Les initialiseurs de base de données sont configurés par contexte. Ils peuvent être définis dans le fichier de configuration à l'aide de l'élément **Context**. Cet élément utilise le nom qualifié d'assembly pour identifier le contexte en cours de configuration.

Par défaut, Code First contextes sont configurés pour utiliser l'initialiseur `CreateDatabaseIfNotExists`. Il existe un attribut **disableDatabaseInitialization** sur l'élément **Context** qui peut être utilisé pour désactiver l'initialisation de la base de données.

Par exemple, la configuration suivante désactive l'initialisation de la base de données pour le contexte blog. BlogContext défini dans MyAssembly. dll.

```
<contexts>
  <context type=" Blogging.BlogContext, MyAssembly" disableDatabaseInitialization="true" />
</contexts>
```

Vous pouvez utiliser l'élément **databaseInitializer** pour définir un initialiseur personnalisé.

```
<contexts>
  <context type=" Blogging.BlogContext, MyAssembly">
    <databaseInitializer type="Blogging.MyCustomBlogInitializer, MyAssembly" />
  </context>
</contexts>
```

Les paramètres de constructeur utilisent la même syntaxe que les fabriques de connexion par défaut.

```
<contexts>
  <context type=" Blogging.BlogContext, MyAssembly">
    <databaseInitializer type="Blogging.MyCustomBlogInitializer, MyAssembly">
      <parameters>
        <parameter value="MyConstructorParameter" />
      </parameters>
    </databaseInitializer>
  </context>
</contexts>
```

Vous pouvez configurer l'un des initialiseurs de base de données génériques inclus dans Entity Framework. L'attribut **type** utilise le format .NET Framework pour les types génériques.

Par exemple, si vous utilisez migrations code First, vous pouvez configurer la base de données pour qu'elle soit

automatiquement migrée à l'aide de l'`MigrateDatabaseToLatestVersion<TContext, TMigrationsConfiguration>` initialiseur.

```
<contexts>
  <context type="Blogging.BlogContext, MyAssembly">
    <databaseInitializer type="System.Data.Entity.MigrateDatabaseToLatestVersion`2[[Blogging.BlogContext,
MyAssembly], [Blogging.Migrations.Configuration, MyAssembly]], EntityFramework" />
  </context>
</contexts>
```

# Modèles et les chaînes de connexion

13/09/2018 • 10 minutes to read

Cette rubrique décrit comment Entity Framework détecte la connexion de base de données à utiliser, et comment vous pouvez le modifier. Les modèles créés avec Code First et le Concepteur EF sont traités dans cette rubrique.

En général, une application Entity Framework utilise une classe dérivée de DbContext. Cette classe dérivée appellera un des constructeurs de la classe DbContext de base au contrôle :

- Comment le contexte doit se connecter à une base de données, autrement dit, comment une chaîne de connexion est trouvé/utilisé
- Si le contexte utilisera calculer un modèle à l'aide de Code First ou charger un modèle créé avec le Concepteur EF
- Options avancées supplémentaires

Les fragments suivants illustrent que certaines des façons les constructeurs DbContext peuvent être utilisées.

## Utiliser Code First par convention avec connexion

Si vous n'avez pas fait toute autre configuration dans votre application, puis en appelant le constructeur sans paramètre sur DbContext entraîne DbContext exécuter en mode Code First avec une connexion de base de données créée par convention. Exemple :

```
namespace Demo.EF
{
    public class BloggingContext : DbContext
    {
        public BloggingContext()
            // C# will call base class parameterless constructor by default
        {
        }
    }
}
```

Dans cet exemple DbContext utilise le nom qualifié d'espace de noms de votre class—Demo.EF.BloggingContext—as de contexte dérivé du nom de la base de données et crée une chaîne de connexion pour cette base de données à l'aide de SQL Express ou LocalDB. Si les deux sont installés, SQL Express sera utilisé.

Visual Studio 2010 inclut SQL Express par défaut et Visual Studio 2012 et versions ultérieures inclut LocalDB. Pendant l'installation, le package EntityFramework NuGet vérifie le serveur de base de données est disponible. Le package NuGet est puis mettez à jour le fichier de configuration en définissant le serveur de base de données par défaut qui utilise Code First lors de la création d'une connexion par convention. Si SQL Express est en cours d'exécution, il sera utilisé. Si SQL Express n'est pas disponible, base de données locale sera être enregistré en tant que la valeur par défaut à la place. Aucun modifications ne sont apportées au fichier de configuration si elle contient déjà un paramètre pour la fabrique de connexion par défaut.

## Utiliser Code First avec connexion par convention et le nom de la base de données spécifiée

Si vous n'avez pas fait toute autre configuration dans votre application, puis en appelant le constructeur string sur DbContext avec le nom de base de données que vous souhaitez utiliser entraîne DbContext exécuter en mode Code First avec une connexion de base de données créée par convention à la base de données Ce nom. Exemple :

```

public class BloggingContext : DbContext
{
    public BloggingContext()
        : base("BloggingDatabase")
    {
    }
}

```

Dans cet exemple DbContext utilise « BloggingDatabase » comme nom de base de données et crée une chaîne de connexion pour cette base de données à l'aide de SQL Express (installé avec Visual Studio 2010) ou base de données locale (installée avec Visual Studio 2012). Si les deux sont installés, SQL Express sera utilisé.

## Utiliser Code First avec la chaîne de connexion dans le fichier de app.config/web.config

Vous pouvez choisir de placer une chaîne de connexion dans votre fichier app.config ou web.config. Exemple :

```

<configuration>
    <connectionStrings>
        <add name="BloggingCompactDatabase"
            providerName="System.Data.SqlServerCe.4.0"
            connectionString="Data Source=Blogging.sdf"/>
    </connectionStrings>
</configuration>

```

Il s'agit d'un moyen simple pour indiquer à DbContext d'utiliser un serveur de base de données autre que SQL Express ou de la base de données locale, l'exemple ci-dessus spécifie une base de données SQL Server Compact Edition.

Si le nom de la chaîne de connexion correspond au nom de votre contexte (avec ou sans qualification d'espace de noms) puis il se trouve par DbContext lorsque le constructeur sans paramètre est utilisé. Si le nom de chaîne de connexion est différent du nom de votre contexte vous pouvez indiquer DbContext pour utiliser cette connexion en mode Code First en passant le nom de chaîne de connexion au constructeur DbContext. Exemple :

```

public class BloggingContext : DbContext
{
    public BloggingContext()
        : base("BloggingCompactDatabase")
    {
    }
}

```

Vous pouvez également utiliser la forme « nom =<nom de chaîne de connexion> » pour la chaîne passée au constructeur DbContext. Exemple :

```

public class BloggingContext : DbContext
{
    public BloggingContext()
        : base("name=BloggingCompactDatabase")
    {
    }
}

```

Ce formulaire rend explicite à laquelle la chaîne de connexion dans votre fichier de configuration. Une exception sera levée si une chaîne de connexion portant le nom spécifié est introuvable.

## Base de données/Model First avec la chaîne de connexion dans le fichier de app.config/web.config

Les modèles créés avec le Concepteur EF diffèrent des Code First dans la mesure votre modèle existe déjà et n'est pas généré à partir du code lorsque l'application s'exécute. Le modèle existe en général comme un fichier EDMX dans votre projet.

Le concepteur ajouterez une chaîne de connexion EF à votre fichier app.config ou web.config. Cette chaîne de connexion est spéciale car elle contient des informations sur la façon de trouver les informations dans votre fichier EDMX. Exemple :

```
<configuration>
  <connectionStrings>
    <add name="Northwind_Entities"
      connectionString="metadata=res://*/Northwind.csdl|
                        res://*/Northwind.ssdl|
                        res://*/Northwind.msl;
      provider=System.Data.SqlClient;
      provider connection string=
        "Data Source=.\sqlexpress;
          Initial Catalog=Northwind;
          Integrated Security=True;
          MultipleActiveResultSets=True";
      providerName="System.Data.EntityClient"/>
  </connectionStrings>
</configuration>
```

Le Concepteur EF génère également un code qui indique à DbContext pour utiliser cette connexion en passant le nom de chaîne de connexion au constructeur DbContext. Exemple :

```
public class NorthwindContext : DbContext
{
    public NorthwindContext()
        : base("name=Northwind_Entities")
    {
    }
}
```

DbContext sait qu'il peut pour charger le modèle existant (au lieu de Code First pour être calculé à partir du code), car la chaîne de connexion est une chaîne de connexion EF contenant les détails du modèle à utiliser.

## Autres options de constructeur DbContext

La classe DbContext contient d'autres constructeurs et les modèles d'utilisation qui permettent des scénarios plus avancés. Certaines d'entre elles sont :

- Vous pouvez utiliser la classe DbModelBuilder pour générer un modèle Code First sans instancier une instance de DbContext. Le résultat de ce est un objet DbModel. Vous pouvez ensuite passer cet objet DbModel à un des constructeurs DbContext lorsque vous êtes prêt à créer votre instance de DbContext.
- Vous pouvez passer une chaîne de connexion complète de DbContext au lieu de simplement le nom de chaîne de base de données ou de connexion. Par défaut, cette chaîne de connexion est utilisée avec le fournisseur System.Data.SqlClient ; Cela peut être modifié en définissant une implémentation différente de IConnectionFactory sur le contexte. Database.DefaultConnectionFactory.
- Vous pouvez utiliser un objet DbConnection existant en le passant à un constructeur DbContext. Si l'objet de connexion est une instance de EntityConnection, puis le modèle spécifié dans la connexion sera utilisé plutôt que calculer un modèle avec Code First. Si l'objet est une instance d'un autre type — par exemple, SqlConnection, le contexte de l'utilisera pour le mode Code First.

- Vous pouvez passer un ObjectContext existant à un constructeur DbContext pour créer un DbContext encapsulant le contexte existant. Cela peut être utilisé pour les applications existantes qui utilisent ObjectContext, mais qui souhaitent tirer parti de DbContext dans certaines parties de l'application.

# Résolution des dépendances

13/09/2018 • 15 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

À compter de EF6, Entity Framework contient un mécanisme à usage général pour obtenir les implémentations des services dont il a besoin. Autrement dit, quand EF utilise une instance de certaines interfaces ou les classes de base il demandera une implémentation concrète de la classe d'interface ou de base à utiliser. Cela s'effectue via l'utilisation de l'interface `IDbDependencyResolver` :

```
public interface IDbDependencyResolver
{
    object GetService(Type type, object key);
}
```

La méthode `GetService` est généralement appelée par Entity Framework et est gérée par une implémentation de `IDbDependencyResolver` fourni par Entity Framework ou par l'application. Lorsqu'elle est appelée, l'argument de type est le type de classe interface ou de base du service demandé, et l'objet de clé est null ou un objet qui fournit des informations contextuelles sur le service demandé.

Sauf mention contraire, n'importe quel objet retourné doit être thread-safe, car il peut être utilisé comme un singleton. Dans de nombreux cas, que l'objet retourné est une fabrique dans ce cas, l'usine elle-même doit être thread-safe, mais l'objet retourné par la fabrique est inutile être thread-safe, car une nouvelle instance est demandée à partir de la fabrique pour chaque utilisation.

Cet article ne contient-elle pas tous les détails d'implémentation `IDbDependencyResolver`, mais au lieu de cela agit comme une référence pour les types de service (autrement dit, l'interface et la base de types de classe) pour lequel EF appelle `GetService` et la sémantique de l'objet clé pour chacun d'eux appels.

## System.Data.Entity.IDatabaseInitializer < TContext >

**Version introduite:** EF6.0.0

**Objet retourné:** un initialiseur de base de données pour le type de contexte donné

**Clé:** non utilisé ; sera null

## Func < System.Data.Entity.Migrations.Sql.SqlMigrationGenerator >

**Version introduite:** EF6.0.0

**Objet retourné:** une fabrique pour créer un générateur SQL qui peut être utilisé pour les Migrations et les autres actions qui entraînent une base de données doit être créée, telle que la création de base de données avec des initialiseurs de base de données.

**Clé:** une chaîne contenant le nom invariant du fournisseur ADO.NET en spécifiant le type de base de données pour laquelle générer la requête SQL. Par exemple, le Générateur SQL de SQL Server est retourné pour la clé « `System.Data.SqlClient` ».

**NOTE**

Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## System.Data.Entity.Core.Common.DbProviderServices

**Version introduite:** EF6.0.0

**Objet retourné:** fournisseur de l'Entity Framework à utiliser pour un nom invariant de fournisseur donné

**Clé:** une chaîne contenant le nom invariant du fournisseur ADO.NET en spécifiant le type de base de données pour laquelle un fournisseur est nécessaire. Par exemple, le fournisseur SQL Server est retourné pour la clé « System.Data.SqlClient ».

**NOTE**

Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## System.Data.Entity.Infrastructure.IDbConnectionFactory

**Version introduite:** EF6.0.0

**Objet retourné:** la fabrique de connexions qui sera utilisée lors de l'Entity Framework crée une connexion de base de données par convention. Autrement dit, lorsque aucune connexion ou une chaîne de connexion correspond à EF, et aucune chaîne de connexion ne peut être trouvé dans le fichier app.config ou web.config, ce service est utilisé pour créer une connexion par convention. Modification de la fabrique de connexion peut autoriser EF d'utiliser un autre type de base de données (par exemple, SQL Server Compact Edition) par défaut.

**Clé:** non utilisé ; sera null

**NOTE**

Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## System.Data.Entity.Infrastructure.IManifestTokenService

**Version introduite:** EF6.0.0

**Objet retourné:** un service qui peut générer un jeton du manifeste du fournisseur à partir d'une connexion. Ce service est généralement utilisé de deux manières. Tout d'abord, il peut être utilisé afin d'éviter un Code First, connexion à la base de données lors de la création d'un modèle. En second lieu, il peut être utilisé pour forcer le Code First pour générer un modèle pour une version de base de données spécifique, par exemple, pour forcer un modèle pour SQL Server 2005, même si SQL Server 2008 est parfois utilisé.

**Durée de vie:** Singleton--le même objet peut être utilisé plusieurs fois simultanément par différents threads

**Clé:** non utilisé ; sera null

## System.Data.Entity.Infrastructure.IDbProviderFactoryService

**Version introduite:** EF6.0.0

**Objet retourné:** un service qui peut obtenir une fabrique de fournisseurs à partir d'une connexion donnée. Sur le .NET 4.5, le fournisseur est publiquement accessible à partir de la connexion. Sur .NET 4 l'implémentation par

défaut de ce service utilise quelques méthodes heuristiques pour trouver le fournisseur correspondant. Si ces cas de défaillance une nouvelle implémentation de ce service peuvent être inscrits pour fournir une résolution appropriées.

**Clé:** non utilisé ; sera null

## Func < DbContext, System.Data.Entity.Infrastructure.IDbModelCacheKey>

**Version introduite:** EF6.0.0

**Objet retourné:** une fabrique qui génère une clé de cache de modèle pour un contexte donné. Par défaut, Entity Framework met en cache un modèle par type de DbContext par le fournisseur. Une implémentation différente de ce service peut être utilisée pour ajouter d'autres informations, telles que nom de schéma, à la clé de cache.

**Clé:** non utilisé ; sera null

## System.Data.Entity.Spatial.DbSpatialServices

**Version introduite:** EF6.0.0

**Objet retourné:** fournisseur spatial EF une qui ajoute la prise en charge pour le fournisseur EF de base pour les types geography et geometry spatiaux.

**Clé:** DbSptialServices est demandé de deux manières. Tout d'abord, spécifique au fournisseur de services spatiaux sont demandés à l'aide d'un objet DbProviderInfo (qui contient l'invariant jeton de nom et le manifest) comme clé. En second lieu, DbSpatialServices peuvent être demandées sans clé. Cela permet de résoudre le « spatial mondial » qui est utilisé lors de la création des types de DbGeography ou DbGeometry autonomes.

### NOTE

Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## Func < System.Data.Entity.Infrastructure.IDbExecutionStrategy>

**Version introduite:** EF6.0.0

**Objet retourné:** une fabrique pour créer un service qui permet à un fournisseur implémenter les nouvelles tentatives ou un autre comportement lorsque les requêtes et les commandes sont exécutées sur la base de données. Si aucune implémentation n'est fournie, puis EF sera simplement exécuter les commandes et propager toutes les exceptions levées. Pour SQL Server ce service est utilisé pour fournir une stratégie de nouvelle tentative, ce qui est particulièrement utile lors de l'exécution par rapport à des serveurs de base de données basée sur le cloud, tels que SQL Azure.

**Clé:** ExecutionStrategyKey un objet qui contient le nom invariant du fournisseur et éventuellement un nom de serveur pour lequel la stratégie d'exécution est utilisée.

### NOTE

Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## Func < DbConnection, chaîne, System.Data.Entity.Migrations.History.HistoryContext>

**Version introduite:** EF6.0.0

**Objet retourné:** une fabrique qui permet à un fournisseur configurer le mappage de la HistoryContext à la `_MigrationHistory` table utilisée par des Migrations Entity Framework. Le HistoryContext est un Code premier DbContext et peut être configuré à l'aide de l'API fluent normale pour modifier des éléments tels que le nom de la table et les spécifications de mappage de colonne.

**Clé:** non utilisé ; sera null

**NOTE**

Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## System.Data.Common.DbProviderFactory

**Version introduite:** EF6.0.0

**Objet retourné:** fournisseur de l'ADO.NET à utiliser pour un nom invariant de fournisseur donné.

**Clé:** une chaîne contenant le nom invariant du fournisseur ADO.NET

**NOTE**

Ce service n'est généralement pas modifié directement dans la mesure où l'implémentation par défaut utilise l'inscription du fournisseur ADO.NET normale. Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## System.Data.Entity.Infrastructure.IProviderInvariantName

**Version introduite:** EF6.0.0

**Objet retourné:** un service qui est utilisé pour déterminer un nom invariant du fournisseur pour un type donné de DbProviderFactory. L'implémentation par défaut de ce service utilise l'inscription du fournisseur ADO.NET. Cela signifie que si le fournisseur ADO.NET n'est pas inscrit de façon normale, car DbProviderFactory est résolue par Entity Framework, puis il sera également nécessaire pour résoudre ce service.

**Clé:** instance de la DbProviderFactory pour lequel un nom invariant est requis.

**NOTE**

Pour plus d'informations sur des services de fournisseur dans EF6, consultez le [modèle de fournisseur EF6](#) section.

## System.Data.Entity.Core.Mapping.ViewGeneration.IViewAssemblyCache

**Version introduite:** EF6.0.0

**Objet retourné:** un cache des assemblies qui contiennent des vues pré générées. Un remplacement est généralement utilisé pour informer EF les assemblies qui contiennent des vues pré générées sans effectuer n'importe quel découverte.

**Clé:** non utilisé ; sera null

## System.Data.Entity.Infrastructure.Pluralization.IPluralizationService

**Version introduite:** EF6.0.0

**Objet retourné:** un service utilisé par EF au pluriel et au singulier les noms. Par défaut, un service de pluralisation anglais est utilisé.

**Clé:** non utilisé ; sera null

## System.Data.Entity.Infrastructure.Interception.IDbInterceptor

**Version introduite:** EF6.0.0

**Objets retournés:** des intercepteurs qui doivent être inscrits au démarrage de l'application. Notez que ces objets sont demandées à l'aide de l'appel GetServices et tous les intercepteurs retournés par n'importe quel programme de résolution de dépendance seront inscrits.

**Clé:** non utilisé ; sera null.

## Func < System.Data.Entity.DbContext, Action < chaîne>, System.Data.Entity.Infrastructure.Interception.DatabaseLogFormatter>

**Version introduite:** EF6.0.0

**Objet retourné:** une fabrique qui sera utilisée pour créer le formateur de journal de base de données qui sera utilisée lorsque le contexte. Propriété de Database.Log est définie sur le contexte donné.

**Clé:** non utilisé ; sera null.

## Func < System.Data.Entity.DbContext>

**Version introduite:** EF6.1.0

**Objet retourné:** une fabrique qui sera utilisée pour créer des instances de contexte pour les Migrations lorsque le contexte n'a pas un constructeur sans paramètre accessible.

**Clé:** objet de Type pour le type de DbContext dérivé pour laquelle une fabrique est nécessaire.

## Func < System.Data.Entity.Core.Metadata.Edm.IMetadataAnnotationSerializer >

**Version introduite:** EF6.1.0

**Objet retourné:** une fabrique qui sera utilisée pour créer des sérialiseurs pour la sérialisation de fortement typée des annotations personnalisées telles que s'ils peuvent être sérialisés ou déserialisés en XML pour une utilisation dans les Migrations Code First.

**Clé:** le nom de l'annotation est sérialisée ou déserialisée.

## Func < System.Data.Entity.Infrastructure.TransactionHandler>

**Version introduite:** EF6.1.0

**Objet retourné:** une fabrique qui sera utilisée pour créer des gestionnaires pour les transactions afin qu'un traitement spécial peut être appliqué aux situations telles que la gestion des échecs de validation.

**Clé:** ExecutionStrategyKey un objet qui contient le nom invariant du fournisseur et éventuellement un nom de serveur pour lequel le Gestionnaire de transaction sera utilisé.

# Gestion des connexions

13/09/2018 • 8 minutes to read

Cette page décrit le comportement d'Entity Framework en ce qui concerne le passage de connexions pour le contexte et les fonctionnalités de la **Database.Connection.Open()** API.

## Connexions passant au contexte

### Comportement de EF5 et versions antérieures

Il existe deux constructeurs qui acceptent les connexions :

```
public DbContext(DbConnection existingConnection, bool contextOwnsConnection)
public DbContext(DbConnection existingConnection, DbCompiledModel model, bool contextOwnsConnection)
```

Il est possible de les utiliser, mais vous devez contourner quelques limitations :

1. Si vous passez une connexion ouverte à une de ces puis la première fois que l'infrastructure essaie d'utiliser qu'une exception `InvalidOperationException` est levée indiquant que le programme ne peut pas rouvrir une connexion déjà ouverte.
2. L'indicateur `contextOwnsConnection` est interprété comme si la connexion à la banque sous-jacente doit être supprimée lorsque le contexte est supprimé ou non. Toutefois, indépendamment de ce paramètre, le magasin de connexion est toujours fermé lorsque le contexte est supprimé. Donc si vous avez plusieurs `DbContext` avec la même connexion quelle que soit la contexte est supprimé tout d'abord fermera la connexion (même si vous avez mélangé une connexion ADO.NET existante avec un `DbContext`, `DbContext` sera toujours fermer la connexion lorsqu'il est supprimé) .

Il est possible de contourner la limitation première ci-dessus en passant d'une connexion fermée et seulement l'exécution de code qui ouvrirait une fois que tous les contextes ont été créés :

```

using System.Collections.Generic;
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.EntityClient;
using System.Linq;

namespace ConnectionManagementExamples
{
    class ConnectionManagementExampleEF5
    {
        public static void TwoDbContextsOneConnection()
        {
            using (var context1 = new BloggingContext())
            {
                var conn =
                    ((EntityConnection)
                        ((IObjectContextAdapter)context1).ObjectContext.Connection)
                        .StoreConnection;

                using (var context2 = new BloggingContext(conn, contextOwnsConnection: false))
                {
                    context2.Database.ExecuteSqlCommand(
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'");

                    var query = context1.Posts.Where(p => p.Blog.Rating > 5);
                    foreach (var post in query)
                    {
                        post.Title += "[Cool Blog]";
                    }
                    context1.SaveChanges();
                }
            }
        }
    }
}

```

La deuxième limite signifie simplement que vous devez éviter de supprimer un de vos objets DbContext jusqu'à ce que vous êtes prêt pour la connexion à fermer.

### **Comportement dans EF6 et versions ultérieures**

Dans EF6 et les versions futures DbContext a les deux constructeurs mêmes mais ne nécessite plus que la connexion passée au constructeur fermée lorsqu'elle est reçue. Par conséquent, vous pouvez désormais :

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace ConnectionManagementExamples
{
    class ConnectionManagementExample
    {
        public static void PassingAnOpenConnection()
        {
            using (var conn = new SqlConnection("{connectionString}"))
            {
                conn.Open();

                var sqlCommand = new SqlCommand();
                sqlCommand.Connection = conn;
                sqlCommand.CommandText =
                    @"UPDATE Blogs SET Rating = 5" +
                    " WHERE Name LIKE '%Entity Framework%'";
                sqlCommand.ExecuteNonQuery();

                using (var context = new BloggingContext(conn, contextOwnsConnection: false))
                {
                    var query = context.Posts.Where(p => p.Blog.Rating > 5);
                    foreach (var post in query)
                    {
                        post.Title += "[Cool Blog]";
                    }
                    context.SaveChanges();
                }

                var sqlCommand2 = new SqlCommand();
                sqlCommand2.Connection = conn;
                sqlCommand2.CommandText =
                    @"UPDATE Blogs SET Rating = 7" +
                    " WHERE Name LIKE '%Entity Framework Rocks%'";
                sqlCommand2.ExecuteNonQuery();
            }
        }
    }
}

```

L'indicateur `contextOwnsConnection` maintenant contrôle également ou non la connexion est fermée et supprimée lorsque la classe `DbContext` est supprimé. Donc dans l'exemple ci-dessus la connexion n'est pas fermée lorsque le contexte est supprimé (ligne 32) comme il aurait été dans les versions précédentes d'EF, mais lors de la connexion elle-même soit libérée (ligne 40).

Bien sûr, il est toujours possible pour la classe `DbContext` prendre le contrôle de la connexion (simplement ensemble `contextOwnsConnection` sur `true` ou utilisez un des autres constructeurs) si vous le souhaitez.

#### **NOTE**

Il existe certaines considérations supplémentaires lors de l'utilisation de transactions avec ce nouveau modèle. Pour plus d'informations, consultez [utilisation de Transactions](#).

## Database.Connection.Open()

### Comportement de EF5 et versions antérieures

Dans EF5 et versions antérieures, il existe un bogue telles que la `ObjectContext.Connection.State` n'a pas mis à

jour pour refléter l'état réel de la connexion à la banque sous-jacente. Par exemple, si vous avez exécuté le code suivant vous pouvez afficher l'état **fermé** même si en fait sous-jacent connexion à la banque est **Open**.

```
((IObjectContextAdapter)context).ObjectContext.Connection.State
```

Séparément, si vous ouvrez la connexion de base de données en appelant Database.Connection.Open() il sera ouverte jusqu'à ce que la prochaine fois que vous exécutez une requête ou appeler tout ce qui requiert une connexion de base de données (par exemple, SaveChanges()) mais après que sous-jacent stocke connexion va être fermée. Le contexte sera puis ouvert à nouveau et ré-fermer la connexion chaque fois qu'une autre opération de base de données est requise :

```
using System;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.EntityClient;

namespace ConnectionManagementExamples
{
    public class DatabaseOpenConnectionBehaviorEF5
    {
        public static void DatabaseOpenConnectionBehavior()
        {
            using (var context = new BloggingContext())
            {
                // At this point the underlying store connection is closed

                context.Database.Connection.Open();

                // Now the underlying store connection is open
                // (though ObjectContext.Connection.State will report closed)

                var blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);

                // The underlying store connection is still open

                context.SaveChanges();

                // After SaveChanges() the underlying store connection is closed
                // Each SaveChanges() / query etc now opens and immediately closes
                // the underlying store connection

                blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);
                context.SaveChanges();
            }
        }
    }
}
```

## Comportement dans EF6 et versions ultérieures

Pour EF6 et les versions futures nous avons adopté l'approche que si le code appelant choisit d'ouvrir la connexion par le contexte d'appel. Database.Connection.Open(), il a une bonne raison de le faire et le framework suppose qu'il souhaite contrôler ouverture et fermeture de la connexion et n'est plus automatiquement ferme la connexion.

### NOTE

Cela peut potentiellement entraîner aux connexions qui sont ouverts pour un certain temps, par conséquent, utilisez avec précaution.

Nous avons également mis à jour le code afin que `ObjectContext.Connection.State` maintenant effectue le suivi de l'état de la connexion sous-jacente correctement.

```
using System;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Core.EntityClient;
using System.Data.Entity.Infrastructure;

namespace ConnectionManagementExamples
{
    internal class DatabaseOpenConnectionBehaviorEF6
    {
        public static void DatabaseOpenConnectionBehavior()
        {
            using (var context = new BloggingContext())
            {
                // At this point the underlying store connection is closed

                context.Database.Connection.Open();

                // Now the underlying store connection is open and the
                // ObjectContext.Connection.State correctly reports open too

                var blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);
                context.SaveChanges();

                // The underlying store connection remains open for the next operation

                blog = new Blog { /* Blog's properties */ };
                context.Blogs.Add(blog);
                context.SaveChanges();

                // The underlying store connection is still open
            } // The context is disposed - so now the underlying store connection is closed
        }
    }
}
```

# Résilience des connexions et logique de nouvelle tentative

05/12/2019 • 11 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Les applications se connectant à un serveur de base de données ont toujours été vulnérables aux interruptions de connexion en raison de défaillances du serveur principal et de l'instabilité du réseau. Toutefois, dans un environnement LAN qui fonctionne sur des serveurs de base de données dédiés, ces erreurs sont assez rares, car une logique supplémentaire pour gérer ces échecs n'est souvent pas nécessaire. Avec la montée en charge des serveurs de base de données Cloud tels que Windows Azure SQL Database et des connexions sur des réseaux moins fiables, il est désormais plus courant de rencontrer des interruptions de connexion. Cela peut être dû à des techniques défensives utilisées par les bases de données Cloud pour garantir l'équité du service, telles que la limitation de la connexion, ou l'instabilité du réseau provoquant des délais d'attente intermittents et d'autres erreurs temporaires.

La résilience de connexion fait référence à la possibilité pour EF de retenter automatiquement les commandes qui échouent en raison de ces interruptions de connexion.

## Stratégies d'exécution

La nouvelle tentative de connexion est prise en charge par une implémentation de l'interface `IDbExecutionStrategy`. Les implémentations de `IDbExecutionStrategy` sont chargées d'accepter une opération et, si une exception se produit, de déterminer si une nouvelle tentative est appropriée et de retenter si elle est. Il existe quatre stratégies d'exécution fournies avec EF :

1. **DefaultExecutionStrategy**: cette stratégie d'exécution ne réessaie aucune opération, il s'agit de la valeur par défaut pour les bases de données autres que SQL Server.
2. **DefaultSqlExecutionStrategy**, : il s'agit d'une stratégie d'exécution interne qui est utilisée par défaut. Cette stratégie ne réessaie pas du tout, mais elle encapsule toutes les exceptions qui peuvent être temporaires pour informer les utilisateurs qu'ils peuvent souhaiter activer la résilience des connexions.
3. **DbExecutionStrategy**: cette classe est appropriée comme classe de base pour d'autres stratégies d'exécution, y compris celles personnalisées. Il implémente une stratégie de nouvelle tentative exponentielle, où la nouvelle tentative initiale a lieu sans délai et le délai augmente de façon exponentielle jusqu'à ce que le nombre maximal de tentatives soit atteint. Cette classe possède une méthode `ShouldRetryOn` abstraite qui peut être implémentée dans des stratégies d'exécution dérivées pour contrôler les exceptions qui doivent être retentées.
4. **SqlAzureExecutionStrategy**: cette stratégie d'exécution hérite de `DbExecutionStrategy` et réessaie sur les exceptions connues comme pouvant être transitoires lorsque vous utilisez Azure SQL Database.

## NOTE

Les stratégies d'exécution 2 et 4 sont incluses dans le fournisseur SQL Server fourni avec EF, qui se trouve dans l'assembly `EntityFramework.SqlServer` et qui sont conçus pour fonctionner avec SQL Server.

## Activation d'une stratégie d'exécution

Le moyen le plus simple pour indiquer à EF d'utiliser une stratégie d'exécution consiste à utiliser la méthode SetExecutionStrategy de la classe [DbConfiguration](#) :

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetExecutionStrategy("System.Data.SqlClient", () => new SqlAzureExecutionStrategy());
    }
}
```

Ce code indique à EF d'utiliser le [SqlAzureExecutionStrategy](#) lors de la connexion à SQL Server.

## Configuration de la stratégie d'exécution

Le constructeur de [SqlAzureExecutionStrategy](#) peut accepter deux paramètres, MaxRetryCount et MaxDelay. Le nombre de MaxRetry est le nombre maximal de tentatives de la stratégie. Le MaxDelay est un intervalle de temps qui représente le délai maximal entre les nouvelles tentatives que la stratégie d'exécution doit utiliser.

Pour définir le nombre maximal de nouvelles tentatives sur 1 et le délai maximal à 30 secondes, vous devez exécuter la commande suivante :

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetExecutionStrategy(
            "System.Data.SqlClient",
            () => new SqlAzureExecutionStrategy(1, TimeSpan.FromSeconds(30)));
    }
}
```

Le [SqlAzureExecutionStrategy](#) réessaie instantanément la première fois qu'un échec temporaire se produit, mais il retardera entre chaque nouvelle tentative jusqu'à ce que le nombre maximal de nouvelles tentatives soit dépassé ou que le délai total atteigne le délai maximal.

Les stratégies d'exécution renouvellent uniquement un nombre limité d'exceptions qui sont généralement temporaires, vous devrez toujours gérer d'autres erreurs et interceppter l'exception [RetryLimitExceeded](#) pour le cas où une erreur n'est pas temporaire ou prend trop de temps pour être résolue automatiquement.

Il existe des limitations connues en cas d'utilisation d'une stratégie de nouvelle tentative d'exécution :

## Les requêtes de streaming ne sont pas prises en charge

Par défaut, EF6 et versions ultérieures effectuent la mise en mémoire tampon des résultats de la requête au lieu de les diffuser. Si vous souhaitez que les résultats soient diffusés en continu, vous pouvez utiliser la méthode [AsStreaming](#) pour modifier une requête LINQ to Entities en streaming.

```
using (var db = new BloggingContext())
{
    var query = (from b in db.Blogs
                 orderby b.Url
                 select b).AsStreaming();
}
```

La diffusion en continu n'est pas prise en charge lorsqu'une stratégie d'exécution de nouvelle tentative est inscrite. Cette limitation existe, car la connexion peut supprimer de façon partielle les résultats retournés. Dans ce cas, EF doit réexécuter l'intégralité de la requête, mais n'a pas de moyen fiable de savoir quels résultats ont déjà été retournés (les données ont peut-être été modifiées depuis l'envoi de la requête initiale, les résultats peuvent revenir dans un ordre différent, les résultats peuvent ne pas avoir d'identificateur unique , etc.).

## Les transactions initiées par l'utilisateur ne sont pas prises en charge

Lorsque vous avez configuré une stratégie d'exécution qui se traduit par de nouvelles tentatives, il existe des restrictions concernant l'utilisation des transactions.

Par défaut, EF effectue toutes les mises à jour de la base de données dans une transaction. Vous n'avez rien à faire pour activer cela, EF le fait toujours automatiquement.

Par exemple, dans le code suivant, SaveChanges est exécuté automatiquement dans une transaction. Si SaveChanges devait échouer après l'insertion de l'un des nouveaux sites, la transaction est restaurée et aucune modification n'est appliquée à la base de données. Le contexte est également laissé dans un État qui permet d'appeler à nouveau SaveChanges pour réessayer d'appliquer les modifications.

```
using (var db = new BloggingContext())
{
    db.Blogs.Add(new Site { Url = "http://msdn.com/data/ef" });
    db.Blogs.Add(new Site { Url = "http://blogs.msdn.com/adonet" });
    db.SaveChanges();
}
```

Lorsque vous n'utilisez pas une stratégie de nouvelle tentative d'exécution, vous pouvez encapsuler plusieurs opérations dans une transaction unique. Par exemple, le code suivant encapsule deux appels de SaveChanges dans une transaction unique. Si une partie de l'une ou l'autre des opérations échoue, aucune des modifications n'est appliquée.

```
using (var db = new BloggingContext())
{
    using (var trn = db.Database.BeginTransaction())
    {
        db.Blogs.Add(new Site { Url = "http://msdn.com/data/ef" });
        db.Blogs.Add(new Site { Url = "http://blogs.msdn.com/adonet" });
        db.SaveChanges();

        db.Blogs.Add(new Site { Url = "http://twitter.com/efmagicunicorns" });
        db.SaveChanges();

        trn.Commit();
    }
}
```

Cela n'est pas pris en charge lors de l'utilisation d'une stratégie de nouvelle tentative d'exécution, car EF n'est pas conscient des opérations précédentes et de la manière de les réessayer. Par exemple, si le deuxième SaveChanges a échoué, EF n'a plus les informations requises pour réessayer le premier appel SaveChanges.

### Solution : appeler manuellement la stratégie d'exécution

La solution consiste à utiliser manuellement la stratégie d'exécution et à lui attribuer l'ensemble de la logique à exécuter, afin qu'elle puisse réessayer tout en cas d'échec de l'une des opérations. Lorsqu'une stratégie d'exécution dérivée de DbExecutionStrategy est en cours d'exécution, elle interrompt la stratégie d'exécution implicite utilisée dans SaveChanges.

Notez que tous les contextes doivent être construits dans le bloc de code pour être retentés. Cela garantit que

nous démarrons avec un état propre pour chaque nouvelle tentative.

```
var executionStrategy = new SqlAzureExecutionStrategy();

executionStrategy.Execute(
    () =>
{
    using (var db = new BloggingContext())
    {
        using (var trn = db.Database.BeginTransaction())
        {
            db.Blogs.Add(new Blog { Url = "http://msdn.com/data/ef" });
            db.Blogs.Add(new Blog { Url = "http://blogs.msdn.com/adonet" });
            db.SaveChanges();

            db.Blogs.Add(new Blog { Url = "http://twitter.com/efmagicunicorns" });
            db.SaveChanges();

            trn.Commit();
        }
    }
});
```

# Gestion des échecs de validation de transaction

27/09/2018 • 6 minutes to read

## NOTE

**EF6.1 et versions ultérieures uniquement** -les fonctionnalités, API, etc. abordés dans cette page ont été introduits dans Entity Framework 6.1. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Dans le cadre de 6.1, nous vous présentons une nouvelle fonctionnalité de résilience de connexion pour Entity Framework : la possibilité de détecter et de récupérer automatiquement lorsque des échecs de connexion temporaires affectent l'accusé de réception de la transaction est validée. Des informations complètes sur le scénario sont mieux décrits dans le billet de blog [connectivité de base de données SQL et le problème de l'idempotence](#). En résumé, le scénario est que lorsqu'une exception est levée pendant une validation de transaction il existe deux causes possibles :

1. Échec de la validation de transaction sur le serveur
2. La validation de transaction a réussi sur le serveur, mais un problème de connectivité a empêché la notification de réussite d'atteindre le client

Quand le premier cas produit, l'application ou l'utilisateur pouvez retenter l'opération, mais lorsque le deuxième cas se produit nouvelles tentatives doivent être évitées et que l'application peut récupérer automatiquement. Le défi qui consiste sans la capacité à détecter quelle était la raison réelle, une exception a été signalée lors de la validation, l'application ne peut pas choisir le bon plan d'action. La nouvelle fonctionnalité dans EF 6.1 permet EF devrait vérifier de nouveau avec la base de données si la transaction a réussi et prendre en toute transparence de la ligne de conduite appropriée.

## À l'aide de la fonctionnalité

Afin d'activer la fonctionnalité, vous devez inclure un appel à [SetTransactionHandler](#) dans le constructeur de votre **DbConfiguration**. Si vous n'êtes pas familiarisé avec **DbConfiguration**, consultez [Configuration en fonction du Code](#). Cette fonctionnalité peut être utilisée en association avec les nouvelles tentatives automatiques qu'ont été introduits dans EF6, ce qui facilite la situation dans laquelle la transaction en fait la validation a échoué sur le serveur en raison d'une défaillance passagère :

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.SqlServer;

public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetTransactionHandler(SqlProviderServices.ProviderInvariantName, () => new CommitFailureHandler());
        SetExecutionStrategy(SqlProviderServices.ProviderInvariantName, () => new SqlAzureExecutionStrategy());
    }
}
```

## Mode de suivi des transactions

Lorsque la fonctionnalité est activée, EF ajoute automatiquement une nouvelle table à la base de données appelée [Transactions](#). Une nouvelle ligne est insérée dans cette table chaque fois qu'une transaction est créée par Entity

Framework et l'existence de cette ligne est vérification si l'échec de la transaction se produit lors de la validation.

Bien que EF fera de son mieux pour nettoyer les lignes de la table lorsqu'ils ne sont plus nécessaires, la table peut atteindre si l'application se termine prématurément et c'est pourquoi que vous devrez peut-être vider la table manuellement dans certains cas.

## Comment gérer les échecs de validation avec les Versions précédentes

Avant d'EF 6.1 s'est pas mécanisme permettant de gérer les échecs de validation dans le produit d'EF. Il existe plusieurs façons de gérer cette situation peut être appliquée aux versions précédentes d'EF6 :

- Option 1 : ne rien faire

La probabilité d'un échec de connexion lors de la validation de transaction est faible, elle peut être acceptable pour votre application de simplement échouer si cette condition se produit réellement.

- Option 2 : utiliser la base de données pour réinitialiser l'état

1. Ignorer le DbContext actuel
2. Créer un nouveau DbContext et restaurer l'état de votre application à partir de la base de données
3. Informer l'utilisateur que la dernière opération ne peut-être pas terminée avec succès

- Option 3 - suivre manuellement la transaction

1. Ajouter une table non suivies à la base de données utilisé pour suivre l'état des transactions.
2. Insérer une ligne dans la table au début de chaque transaction.
3. Si la connexion échoue lors de la validation, vérifier la présence de la ligne correspondante dans la base de données.
  - Si la ligne est présente, se poursuivent normalement, car la transaction a été validée avec succès
  - Si la ligne est absente, utilisez une stratégie d'exécution pour retenter l'opération actuelle.
4. Si la validation est réussie, supprimez la ligne correspondante pour éviter la croissance de la table.

[Ce billet de blog](#) contient des exemples de code pour accomplir cela sur SQL Azure.

# Liaison de liaison avec WinForms

23/11/2019 • 25 minutes to read

Cette procédure pas à pas montre comment lier des types POCO à des contrôles Windows Forms (WinForms) dans un formulaire maître/détail. L'application utilise Entity Framework pour remplir les objets avec les données de la base de données, effectuer le suivi des modifications et conserver les données dans la base de données.

Le modèle définit deux types qui participent à une relation un-à-plusieurs : la catégorie (principal\maître) et le produit (détails\dépendants). Les outils Visual Studio sont ensuite utilisés pour lier les types définis dans le modèle aux contrôles WinForms. L'infrastructure de liaison de données WinForms active la navigation entre les objets connexes : la sélection de lignes dans le mode maître entraîne la mise à jour de la vue détaillée avec les données enfants correspondantes.

Les captures d'écran et les listes de codes de cette procédure pas à pas sont extraites de Visual Studio 2013 mais vous pouvez effectuer cette procédure pas à pas avec Visual Studio 2012 ou Visual Studio 2010.

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous devez installer Visual Studio 2013, Visual Studio 2012 ou Visual Studio 2010.

Si vous utilisez Visual Studio 2010, vous devez également installer NuGet. Pour plus d'informations, consultez [installation de NuGet](#).

## Création de l'application

- Ouvrez Visual Studio
- **Fichier> nouveau>...**
- Sélectionnez **fenêtres** dans le volet gauche et **Windows FormsApplication** dans le volet droit
- Entrez **WinFormswithEFSample** comme nom
- Sélectionnez **OK**.

## Installer le package NuGet Entity Framework

- Dans Explorateur de solutions, cliquez avec le bouton droit sur le projet **WinFormswithEFSample**
- Sélectionnez **gérer les packages NuGet...**
- Dans la boîte de dialogue gérer les packages NuGet, sélectionnez l'onglet **en ligne** et choisissez le package **EntityFramework** .
- Cliquez sur **installer**

### NOTE

En plus de l'assembly EntityFramework, une référence à System. ComponentModel. DataAnnotations est également ajoutée. Si le projet a une référence à System. Data. Entity, il sera supprimé lors de l'installation du package EntityFramework. L'assembly System. Data. Entity n'est plus utilisé pour les applications Entity Framework 6.

## Implémentation de **IListSource** pour les collections

Les propriétés de collection doivent implémenter l'interface **IListSource** pour activer la liaison de données bidirectionnelle avec le tri lors de l'utilisation de Windows Forms. Pour ce faire, nous allons étendre

ObservableCollection pour ajouter la fonctionnalité IListSource.

- Ajoutez une classe **ObservableListSource** au projet:
  - Cliquez avec le bouton droit sur le nom du projet
  - Sélectionnez **Ajouter-> nouvel élément**
  - Sélectionnez **classe** et entrez **ObservableListSource** pour le nom de la classe
- Remplacez le code généré par défaut par le code suivant :

*Cette classe active la liaison de données bidirectionnelle et le tri. La classe dérive de ObservableCollection<T> et ajoute une implémentation explicite de IListSource. La méthode GetList () de IListSource est implémentée pour retourner une implémentation de IBindingList qui reste synchronisée avec ObservableCollection. L'implémentation IBindingList générée par ToBindingList prend en charge le tri. La méthode d'extension ToBindingList est définie dans l'assembly EntityFramework.*

```
using System.Collections;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Diagnostics.CodeAnalysis;
using System.Data.Entity;

namespace WinFormswithEFSample
{
    public class ObservableListSource<T> : ObservableCollection<T>, IListSource
        where T : class
    {
        private IBindingList _bindingList;

        bool IListSource.ContainsListCollection { get { return false; } }

        IList IListSource.GetList()
        {
            return _bindingList ?? (_bindingList = this.ToBindingList());
        }
    }
}
```

## Définir un modèle

Dans cette procédure pas à pas, vous pouvez choisir d'implémenter un modèle à l'aide de Code First ou du concepteur EF. Suivez l'une des deux sections suivantes.

### Option 1 : définir un modèle à l'aide de Code First

Cette section montre comment créer un modèle et la base de données qui lui est associée à l'aide de Code First. Passez à la section suivante (**option 2 : définir un modèle à l'aide de Database First**) si vous préférez utiliser Database First pour rétroconcevoir votre modèle à partir d'une base de données à l'aide du concepteur EF

Lors de l'utilisation de Code First développement, vous commencez généralement par écrire des classes .NET Framework qui définissent votre modèle conceptuel (domaine).

- Ajouter une nouvelle classe de **produit** au projet
- Remplacez le code généré par défaut par le code suivant :

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormswithEFSample
{
    public class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }

        public int CategoryId { get; set; }
        public virtual Category Category { get; set; }
    }
}

```

- Ajoutez une classe **Category** au projet.
- Remplacez le code généré par défaut par le code suivant :

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace WinFormswithEFSample
{
    public class Category
    {
        private readonly ObservableCollection<Product> _products =
            new ObservableCollection<Product>();

        public int CategoryId { get; set; }
        public string Name { get; set; }
        public virtual ObservableCollection<Product> Products { get { return _products; } }
    }
}

```

En plus de définir des entités, vous devez définir une classe qui dérive de **DbContext** et expose les propriétés **DbSet< TEntity >**. Les propriétés **DbSet** permettent au contexte de savoir quels types vous souhaitez inclure dans le modèle. Les types **DbContext** et **DbSet** sont définis dans l'assembly EntityFramework.

Une instance du type dérivé DbContext gère les objets d'entité au moment de l'exécution, ce qui comprend le remplissage des objets avec les données d'une base de données, le suivi des modifications et la persistance des données dans la base de données.

- Ajoutez une nouvelle classe **ProductContext** au projet.
- Remplacez le code généré par défaut par le code suivant :

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Text;

namespace WinFormswithEFSample
{
    public class ProductContext : DbContext
    {
        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
    }
}

```

Compilez le projet.

## Option 2 : définir un modèle à l'aide de Database First

Cette section montre comment utiliser Database First pour rétroconcevoir votre modèle à partir d'une base de données à l'aide du concepteur EF. Si vous avez terminé la section précédente (**option 1 : définir un modèle à l'aide de code First**) , ignorez cette section et passez directement à la section **chargement différé** .

### Créer une base de données existante

En général, lorsque vous ciblez une base de données existante, elle est déjà créée, mais pour cette procédure pas à pas, nous devons créer une base de données à laquelle accéder.

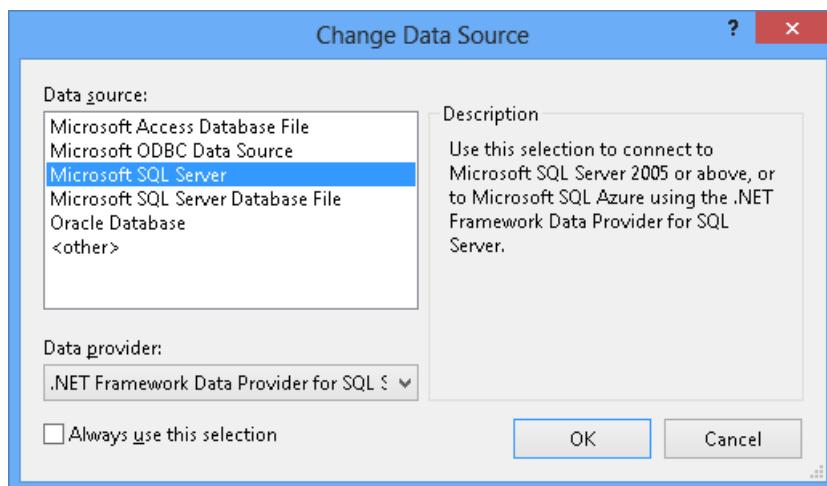
Le serveur de base de données installé avec Visual Studio diffère selon la version de Visual Studio que vous avez installée :

- Si vous utilisez Visual Studio 2010, vous allez créer une base de données SQL Express.
- Si vous utilisez Visual Studio 2012, vous allez créer une base de données de [base de données locale](#).

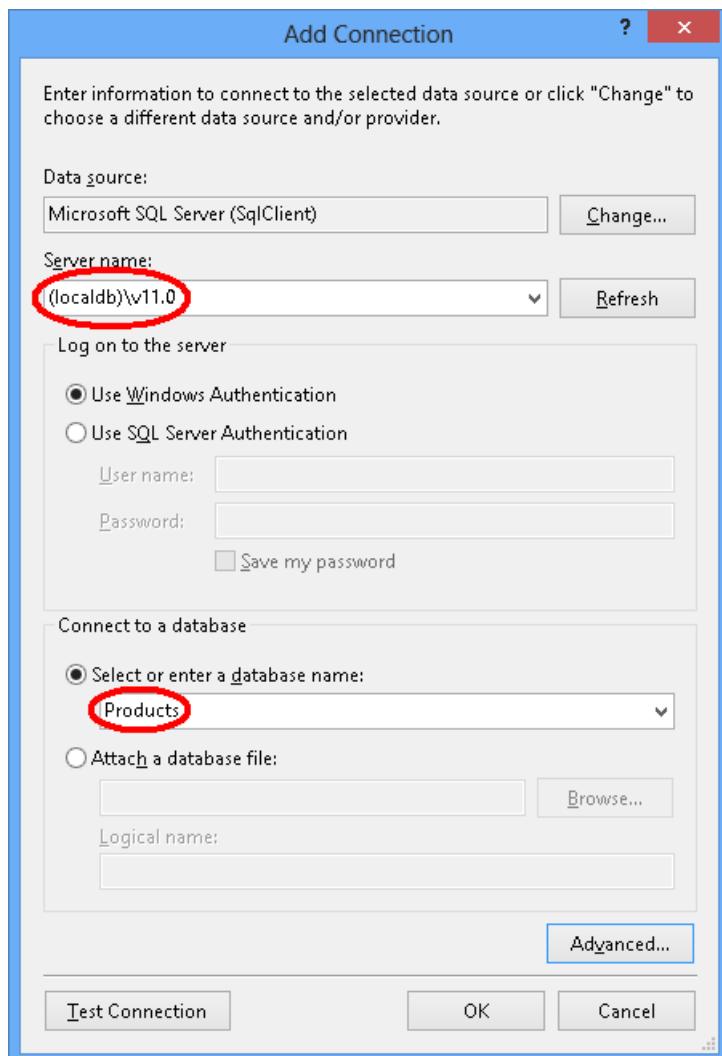
Commençons par générer la base de données.

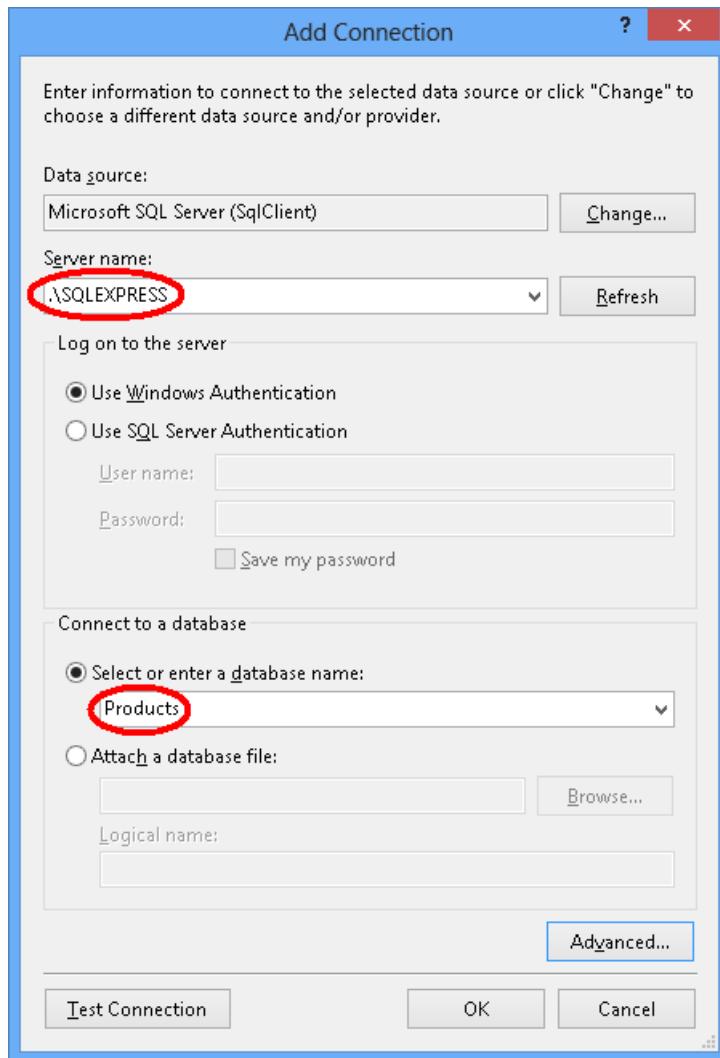
### • Vue-> Explorateur de serveurs

- Cliquez avec le bouton droit sur **connexions de données-> ajouter une connexion...**
- Si vous n'êtes pas connecté à une base de données à partir de Explorateur de serveurs avant de devoir sélectionner Microsoft SQL Server comme source de données



- Connectez-vous à la base de données locale ou SQL Express, en fonction de celle que vous avez installée, puis entrez **Products** comme nom de la base de données.





- Sélectionnez **OK**. vous serez invité à créer une nouvelle base de données, sélectionnez **Oui** .



- La nouvelle base de données s'affiche alors dans Explorateur de serveurs, cliquez dessus avec le bouton droit et sélectionnez **nouvelle requête** .
- Copiez le code SQL suivant dans la nouvelle requête, cliquez avec le bouton droit sur la requête et sélectionnez **exécuter** .

```

CREATE TABLE [dbo].[Categories] (
    [CategoryId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    CONSTRAINT [PK_dbo.Categories] PRIMARY KEY ([CategoryId])
)

CREATE TABLE [dbo].[Products] (
    [ProductId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    [CategoryId] [int] NOT NULL,
    CONSTRAINT [PK_dbo.Products] PRIMARY KEY ([ProductId])
)

CREATE INDEX [IX_CategoryId] ON [dbo].[Products]([CategoryId])

ALTER TABLE [dbo].[Products] ADD CONSTRAINT [FK_dbo.Products_dbo.Categories_CategoryId] FOREIGN KEY
([CategoryId]) REFERENCES [dbo].[Categories] ([CategoryId]) ON DELETE CASCADE

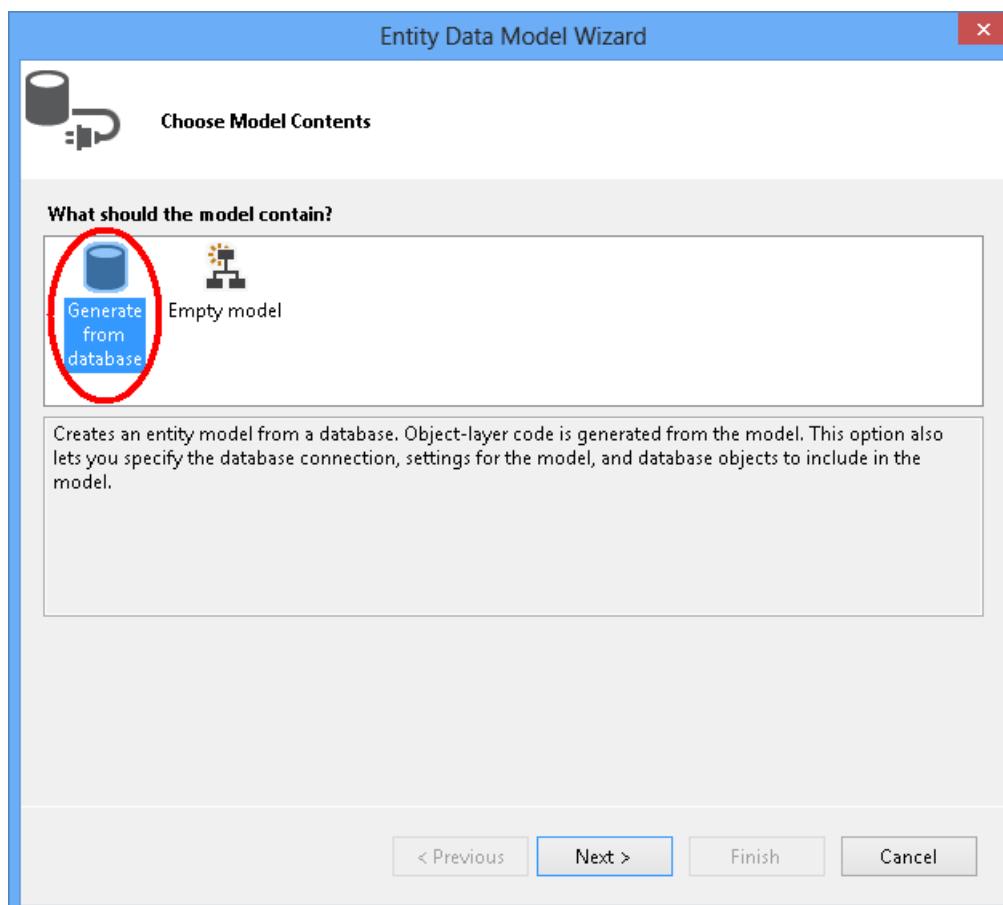
```

#### Modèle d'ingénierie à rebours

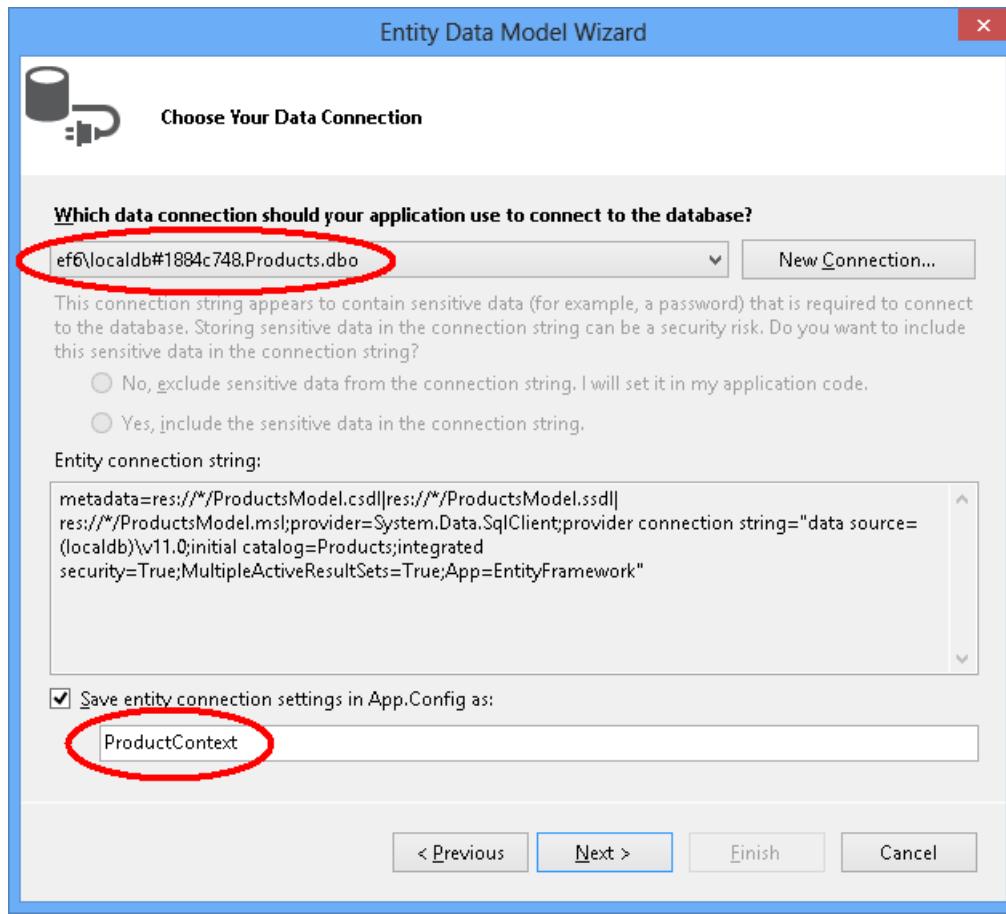
Nous allons utiliser Entity Framework Designer, inclus dans le cadre de Visual Studio, pour créer notre modèle.

- **Projet-> ajouter un nouvel élément...**

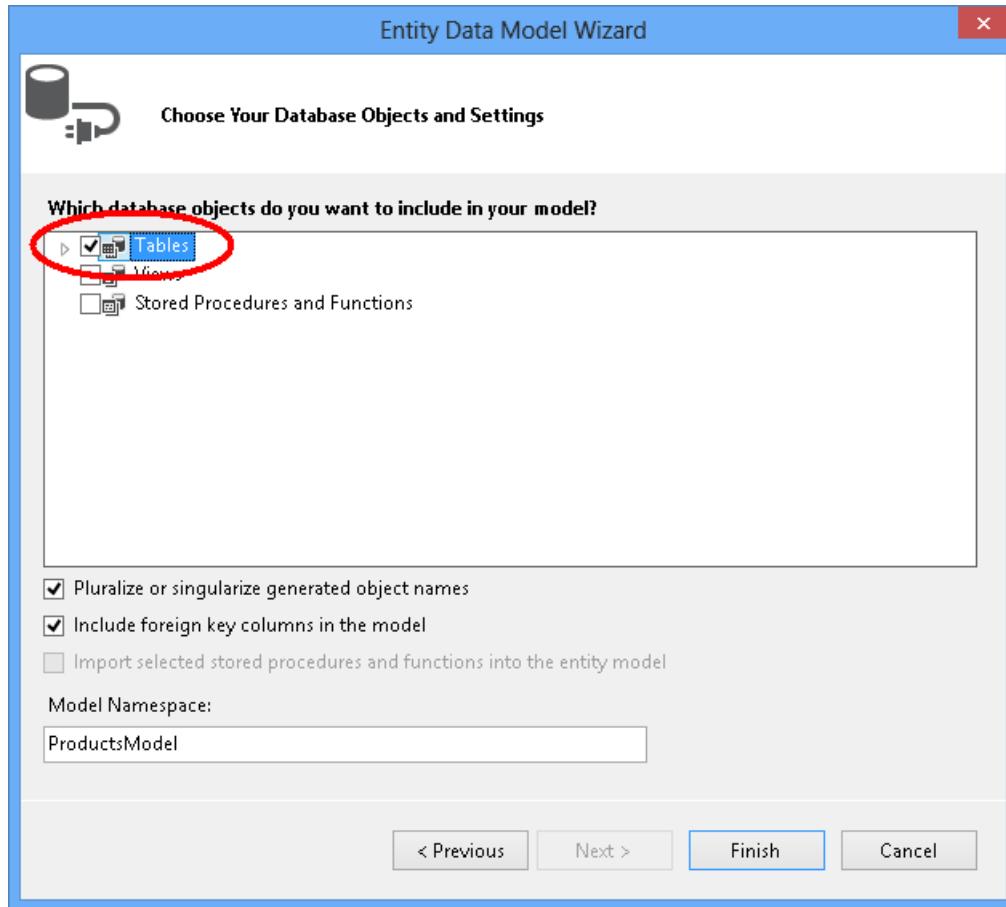
- Sélectionnez **données** dans le menu de gauche, puis **ADO.NET Entity Data Model**
- Entrez **ProductModel** comme nom et cliquez sur **OK**.
- Cela lance l'**assistant Entity Data Model**
- Sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.



- Sélectionnez la connexion à la base de données que vous avez créée dans la première section, entrez **ProductContext** comme nom de la chaîne de connexion, puis cliquez sur **suivant**.



- Cochez la case en regard de « tables » pour importer toutes les tables, puis cliquez sur « Terminer ».



Une fois le processus d'ingénierie à rebours terminé, le nouveau modèle est ajouté à votre projet et vous est ouvert pour que vous l'affichez dans le Entity Framework Designer. Un fichier app.config a également été ajouté à votre projet avec les détails de connexion de la base de données.

## Étapes supplémentaires dans Visual Studio 2010

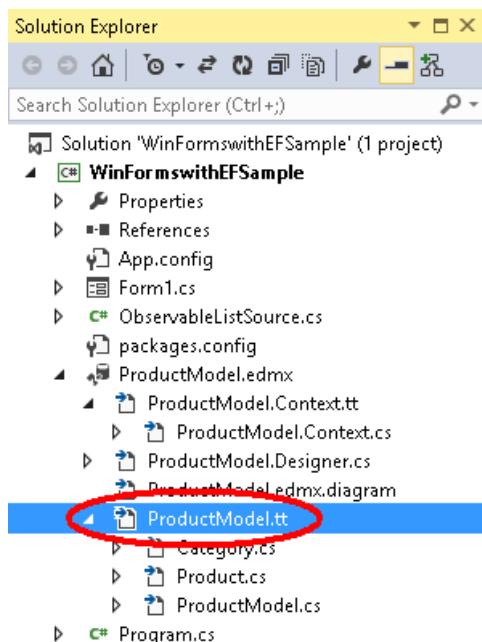
Si vous travaillez dans Visual Studio 2010, vous devrez mettre à jour le concepteur EF pour utiliser la génération de code EF6.

- Cliquez avec le bouton droit sur une zone vide de votre modèle dans le concepteur EF, puis sélectionnez **Ajouter un élément de génération de code...**
- Sélectionnez **modèles en ligne** dans le menu de gauche et recherchez **DbContext**
- Sélectionnez le **Générateur de DbContext EF 6. x pour C#**, entrez **ProductsModel** comme nom et cliquez sur Ajouter.

### Mise à jour de la génération de code pour la liaison de données

EF génère du code à partir de votre modèle à l'aide de modèles T4. Les modèles fournis avec Visual Studio ou téléchargés à partir de la Galerie Visual Studio sont destinés à un usage général. Cela signifie que les entités générées à partir de ces modèles ont des propriétés `ICollection<T>` simples. Toutefois, lors de la liaison de données, il est souhaitable d'avoir des propriétés de collection qui implémentent `IListSource`. C'est la raison pour laquelle nous avons créé la classe `ObservableListSource` ci-dessus et nous allons maintenant modifier les modèles pour utiliser cette classe.

- Ouvrir le **Explorateur de solutions** et rechercher le fichier **ProductModel.edmx**
- Rechercher le fichier **ProductModel.tt** qui sera imbriqué dans le fichier ProductModel.edmx



- Double-cliquez sur le fichier `ProductModel.tt` pour l'ouvrir dans l'éditeur Visual Studio.
- Recherchez et remplacez les deux occurrences de «`ICollection`» par «`ObservableListSource`». Celles-ci se trouvent sur des lignes d'environ 296 et 484.
- Recherchez et remplacez la première occurrence de «`HashSet`» par «`ObservableListSource`». Cette occurrence se trouve à environ la ligne 50. **Ne remplacez pas** la deuxième occurrence de `HashSet` trouvée plus loin dans le code.
- Enregistrez le fichier `ProductModel.tt`. Cela devrait entraîner la régénération du code pour les entités. Si le code ne se régénère pas automatiquement, cliquez avec le bouton droit sur `ProductModel.tt` et choisissez « exécuter un outil personnalisé ».

Si vous ouvrez maintenant le fichier `Category.cs` (qui est imbriqué sous `ProductModel.tt`), vous devez voir que la collection `Products` a le type `ObservableListSource<Product>`.

Compilez le projet.

## Chargement différé

La propriété **Products** de la classe **Category** et la propriété **Category** de la classe **Product** sont des propriétés de navigation. Dans Entity Framework, les propriétés de navigation offrent un moyen de naviguer dans une relation entre deux types d'entités.

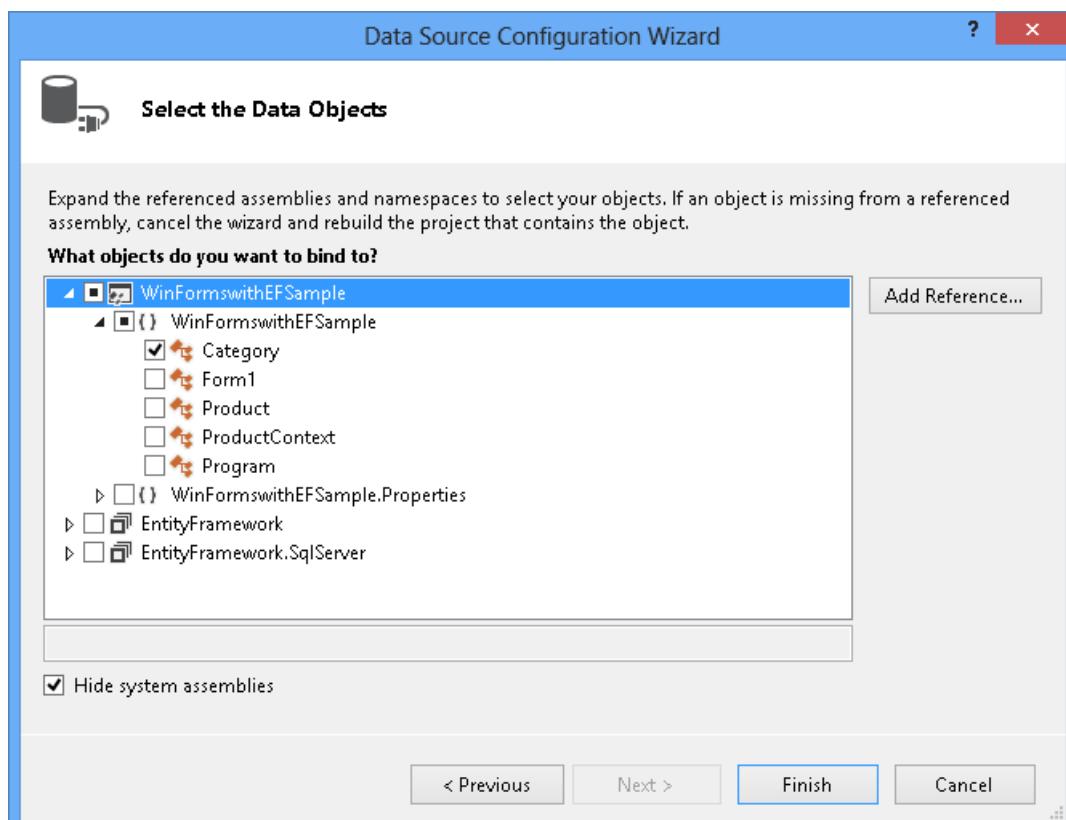
EF vous donne la possibilité de charger automatiquement les entités associées à partir de la base de données la première fois que vous accédez à la propriété de navigation. Avec ce type de chargement (appelé chargement différé), sachez que la première fois que vous accédez à chaque propriété de navigation, une requête distincte est exécutée sur la base de données si le contenu n'est pas déjà dans le contexte.

Lorsque vous utilisez des types d'entités POCO, EF réalise un chargement différé en créant des instances de types de proxy dérivés pendant l'exécution, puis en substituant les propriétés virtuelles dans vos classes pour ajouter le raccordement de chargement. Pour bénéficier du chargement différé d'objets connexes, vous devez déclarer les accesseurs get de propriété de navigation comme **public** et **virtuel (substituable** dans Visual Basic) et vous ne devez pas être **sealed (NotOverridable** dans Visual Basic). Lors de l'utilisation de Database First propriétés de navigation sont automatiquement configurées pour permettre le chargement différé. Dans la section Code First, nous avons choisi de rendre les propriétés de navigation virtuelles pour la même raison

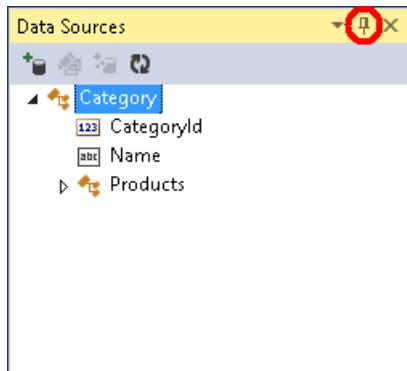
## Lier un objet à des contrôles

Ajoutez les classes définies dans le modèle en tant que sources de données pour cette application WinForms.

- Dans le menu principal, sélectionnez **projet-> ajouter une nouvelle source de données...** (dans Visual Studio 2010, vous devez sélectionner **données-> ajouter une nouvelle source de données...** )
- Dans la fenêtre choisir un type de source de données, sélectionnez **objet**, puis cliquez sur **suivant .**
- Dans la boîte de dialogue Sélectionner les objets de données, dérouler les **WinFormswithEFSample** deux fois et sélectionner une **catégorie** il n'est pas nécessaire de sélectionner la source de données du produit, car nous y accéderons via la propriété du produit sur la source de données de catégorie.



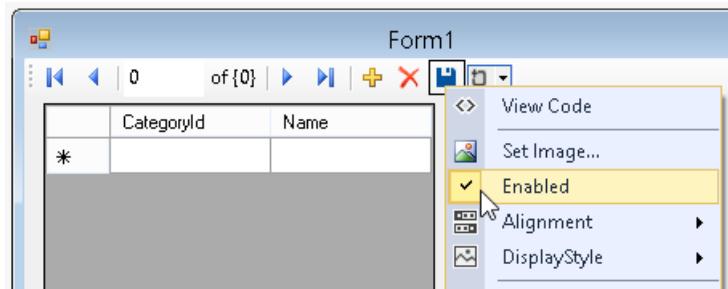
- Cliquez sur **Terminer**. Si la fenêtre sources de données ne s'affiche pas, sélectionnez **Afficher-> autres sources de données Windows->**
- Appuyez sur l'icône d'épingle pour que la fenêtre sources de données ne soit pas masquée automatiquement. Vous devrez peut-être cliquer sur le bouton Actualiser si la fenêtre était déjà visible.



- Dans Explorateur de solutions, double-cliquez sur le fichier **Form1.cs** pour ouvrir le formulaire principal dans le concepteur.
- Sélectionnez la source de données de **catégorie** et faites-la glisser sur le formulaire. Par défaut, un nouveau DataGridView (**categoryDataGridView**) et des contrôles de barre d'outils de navigation sont ajoutés au concepteur. Ces contrôles sont liés aux composants BindingSource (**categoryBindingSource**) et du navigateur de liaison (**categoryBindingNavigator**) qui sont également créés.
- Modifiez les colonnes sur le **categoryDataGridView**. Nous souhaitons définir la colonne **CategoryID** en lecture seule. La valeur de la propriété **CategoryID** est générée par la base de données après l'enregistrement des données.
  - Cliquez avec le bouton droit sur le contrôle DataGridView et sélectionnez **Modifier les colonnes...**
  - Sélectionnez la colonne **CategoryId** et affectez la valeur **true** à **ReadOnly**.
  - Appuyez sur **OK**
- Sélectionnez produits dans la source de données catégorie et faites-le glisser sur le formulaire. Les **productDataGridView** et **productBindingSource** sont ajoutés au formulaire.
- Modifiez les colonnes sur le **productDataGridView**. Nous souhaitons masquer les colonnes **CategoryId** et **Category** et définir **ProductId** en lecture seule. La valeur de la propriété **ProductId** est générée par la base de données après l'enregistrement des données.
  - Cliquez avec le bouton droit sur le contrôle DataGridView et sélectionnez **modifier les colonnes...**
  - Sélectionnez la colonne **ProductId** et affectez la valeur **true** à **ReadOnly**.
  - Sélectionnez la colonne **CategoryID** et appuyez sur le bouton **supprimer**. Faites de même avec la colonne **Category**.
  - Appuyez sur **OK**.

Jusqu'à présent, nous avons associé nos contrôles DataGridView à des composants BindingSource dans le concepteur. Dans la section suivante, nous allons ajouter du code au code-behind pour définir **categoryBindingSource**. **DataSource** sur la collection d'entités actuellement suivies par **DbContext**. Lorsque nous avons fait glisser les produits de la catégorie sous la catégorie, le WinForms s'est engagé à configurer la propriété **productsBindingSource**. **DataSource** sur **categoryBindingSource** et la propriété **productsBindingSource**. **DataMember** sur **Products**. En raison de cette liaison, seuls les produits appartenant à la catégorie actuellement sélectionnée seront affichés dans le **productDataGridView**.

- Activez le bouton **Enregistrer** dans la barre d'outils de navigation en cliquant sur le bouton droit de la souris et en sélectionnant **activé**.



- Ajoutez le gestionnaire d'événements pour le bouton enregistrer en double-cliquant sur le bouton. Cette opération ajoute le gestionnaire d'événements et vous permet d'afficher le code-behind du formulaire. Le code du gestionnaire d'événements **categoryBindingNavigatorSaveItem\_Click** est ajouté dans la section suivante.

## Ajouter le code qui gère l'interaction des données

Nous allons maintenant ajouter le code permettant d'utiliser le ProductContext pour accéder aux données. Mettez à jour le code de la fenêtre principale du formulaire comme indiqué ci-dessous.

Le code déclare une instance de longue durée de ProductContext. L'objet ProductContext est utilisé pour interroger et enregistrer des données dans la base de données. La méthode Dispose () sur l'instance ProductContext est ensuite appelée à partir de la méthode OnClosing substituée. Les commentaires de code fournissent des détails sur ce que fait le code.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.Entity;

namespace WinFormswithEFSample
{
    public partial class Form1 : Form
    {
        ProductContext _context;
        public Form1()
        {
            InitializeComponent();
        }

        protected override void OnLoad(EventArgs e)
        {
            base.OnLoad(e);
            _context = new ProductContext();

            // Call the Load method to get the data for the given DbSet
            // from the database.
            // The data is materialized as entities. The entities are managed by
            // the DbContext instance.
            _context.Categories.Load();

            // Bind the categoryBindingSource.DataSource to
            // all the Unchanged, Modified and Added Category objects that
            // are currently tracked by the DbContext.
            // Note that we need to call ToBindingList() on the
            // ObservableCollection< TEntity > returned by
            // the DbSet.Local property to get the BindingList< T >
            // in order to facilitate two-way binding in WinForms.
        }
    }
}

```

```

        this.categoryBindingSource.DataSource =
            _context.Categories.Local.ToBindingList();
    }

    private void categoryBindingNavigatorSaveItem_Click(object sender, EventArgs e)
    {
        this.Validate();

        // Currently, the Entity Framework doesn't mark the entities
        // that are removed from a navigation property (in our example the Products)
        // as deleted in the context.
        // The following code uses LINQ to Objects against the Local collection
        // to find all products and marks any that do not have
        // a Category reference as deleted.
        // The ToList call is required because otherwise
        // the collection will be modified
        // by the Remove call while it is being enumerated.
        // In most other situations you can do LINQ to Objects directly
        // against the Local property without using ToList first.
        foreach (var product in _context.Products.Local.ToList())
        {
            if (product.Category == null)
            {
                _context.Products.Remove(product);
            }
        }

        // Save the changes to the database.
        this._context.SaveChanges();

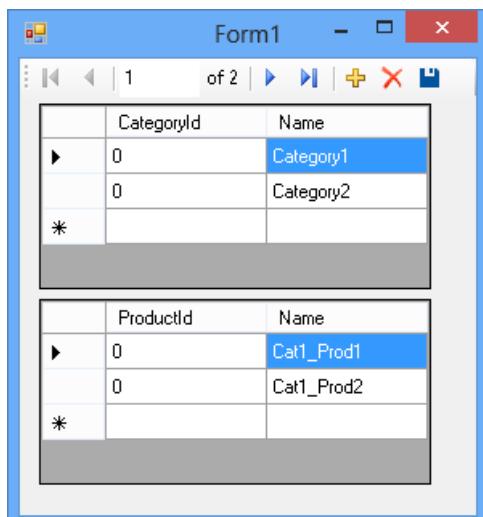
        // Refresh the controls to show the values
        // that were generated by the database.
        this.categoryDataGridView.Refresh();
        this.productsDataGridView.Refresh();
    }

    protected override void OnClosing(CancelEventArgs e)
    {
        base.OnClosing(e);
        this._context.Dispose();
    }
}
}

```

## Tester l'application Windows Forms

- Compilez et exéutez l'application, et vous pouvez tester la fonctionnalité.



- Une fois les clés générées par le magasin enregistrées, elles s'affichent à l'écran.

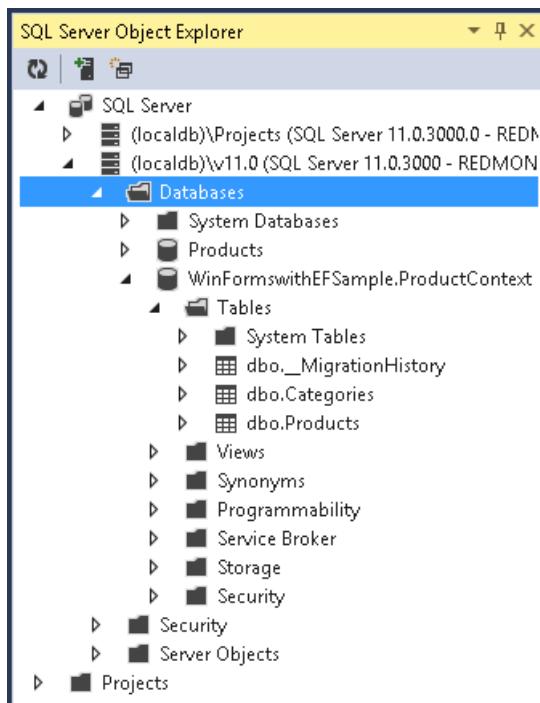
Form1

|   | CategoryId | Name      |
|---|------------|-----------|
| ▶ | 1          | Category1 |
|   | 2          | Category2 |
| * |            |           |

|   | ProductId | Name       |
|---|-----------|------------|
| ▶ | 1         | Cat1_Prod1 |
|   | 2         | Cat1_Prod2 |
| * |           |            |

- Si vous avez utilisé Code First, vous verrez également qu'une base de données **WinFormswithEFSample**. **ProductContext** est créée pour vous.



# Liaison de données avec WPF

04/10/2018 • 26 minutes to read

Cette procédure pas à pas montre comment lier les types POCO à des contrôles WPF dans un formulaire « maître / détail ». L'application utilise les API Entity Framework pour remplir des objets avec des données à partir de la base de données, le suivi des modifications et conserver les données dans la base de données.

Le modèle définit deux types impliqués dans une relation un-à-plusieurs : **catégorie** (principal\master) et **produit** (dépendants\détail). Ensuite, les outils de Visual Studio sont utilisés pour lier les types définis dans le modèle pour les contrôles WPF. L'infrastructure de liaison de données WPF permet la navigation entre les objets connexes : sélection de lignes dans la vue principale, la vue de détail mettre à jour avec les données enfants correspondantes.

Les captures d'écran et les listings de code dans cette procédure pas à pas sont effectuées à partir de Visual Studio 2013, mais vous pouvez effectuer cette procédure pas à pas avec Visual Studio 2012 ou Visual Studio 2010.

## Utilisez l'Option 'Object' pour la création de Sources de données WPF

Avec une version précédente d'Entity Framework, nous avons utilisé est recommandé d'utiliser le **base de données** option lorsque vous créez une nouvelle Source de données basé sur un modèle créé avec le Concepteur EF. C'est parce que le concepteur génère un contexte dérivé ObjectContext et les classes d'entité dérivé à partir d'EntityObject. À l'aide de l'option de base de données vous permet d'écrire le meilleur code permettant d'interagir avec cette surface d'API.

Les concepteurs d'EF pour Visual Studio 2012 et Visual Studio 2013 générer un contexte qui dérive de DbContext ainsi que de simples classes d'entité POCO. Avec Visual Studio 2010, nous vous recommandons de passer en un modèle de génération de code qui utilise le DbContext comme décrit plus loin dans cette procédure pas à pas.

Lors de l'utilisation de la surface de cette API, vous devez utiliser le **objet** option lors de la création d'une nouvelle Source de données, comme illustré dans cette procédure pas à pas.

Si nécessaire, vous pouvez revenir à la génération de code en fonction de ObjectContext pour les modèles créés avec le Concepteur EF.

## Conditions préalables

Vous devez disposer de Visual Studio 2013, Visual Studio 2012 ou Visual Studio 2010 est installé pour terminer cette procédure pas à pas.

Si vous utilisez Visual Studio 2010, vous devez également installer le package NuGet. Pour plus d'informations, consultez [installation NuGet](#).

## Création de l'application

- Ouvrir Visual Studio
- **Fichier -> nouveau -> projet...**
- Sélectionnez **Windows** dans le volet gauche et **WPFAApplication** dans le volet droit
- Entrez **WPFwithEFSample** comme nom
- Sélectionnez **OK**.

## Installez le package NuGet Entity Framework

- Dans l'Explorateur de solutions, cliquez sur le **WinFormswithEFSample** projet
- Sélectionnez **gérer les Packages NuGet...**
- Dans la boîte de dialogue Gérer les Packages NuGet, sélectionnez le **Online** onglet et sélectionnez le **EntityFramework** package
- Cliquez sur **installer**

#### NOTE

En plus de l'assembly EntityFramework une référence à System.ComponentModel.DataAnnotations est également ajoutée. Si le projet a une référence à System.Data.Entity, puis elle va être supprimée lorsque le package EntityFramework est installé. L'assembly System.Data.Entity est n'est plus utilisé pour les applications Entity Framework 6.

## Définir un modèle

Dans cette procédure pas à pas, que vous pouvez choisir d'implémenter un modèle à l'aide de Code First ou le Concepteur EF. Effectuez l'une des deux sections suivantes.

### Option 1 : Définir un modèle à l'aide de Code First

Cette section montre comment créer un modèle et sa base de données associée à l'aide de Code First. Passez à la section suivante (**Option 2 : définir un modèle à l'aide de la première base de données**) si vous préférez utiliser Database First pour inverser concevoir votre modèle à partir d'une base de données à l'aide du Concepteur EF

Lors de l'utilisation du développement Code First vous commencez généralement par écriture de classes .NET Framework qui définissent votre modèle conceptuel (domaine).

- Ajoutez une nouvelle classe à la **WPFwithEFSample** :
  - Avec le bouton droit sur le nom du projet
  - Sélectionnez **ajouter**, puis **nouvel élément**
  - Sélectionnez **classe** et entrez **produit** pour le nom de classe
- Remplacez le **produit** définition avec le code suivant de la classe :

```

namespace WPFwithEFSample
{
    public class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }

        public int CategoryId { get; set; }
        public virtual Category Category { get; set; }
    }
}

```

- Add a **\*\*Category\*\*** class with the following definition:

```

using System.Collections.ObjectModel;

namespace WPFwithEFSample
{
    public class Category
    {
        public Category()
        {
            this.Products = new ObservableCollection<Product>();
        }

        public int CategoryId { get; set; }
        public string Name { get; set; }

        public virtual ObservableCollection<Product> Products { get; private set; }
    }
}

```

Le **produits** propriété sur le **catégorie** classe et **catégorie** propriété sur le **produit** classe sont des propriétés de navigation. Dans Entity Framework, les propriétés de navigation permettent de naviguer d'une relation entre deux types d'entités.

Outre la définition des entités, vous devez définir une classe qui dérive de DbContext et expose DbSet< TEntity > propriétés. Le DbSet< TEntity > propriétés permettent le contexte de connaître les types que vous souhaitez inclure dans le modèle.

Une instance du type DbContext dérivée gère les objets d'entité pendant l'exécution, ce qui inclut le remplissage des objets avec des données à partir d'une base de données, modifier le suivi et la persistance des données à la base de données.

- Ajouter un nouveau **ProductContext** classe au projet avec la définition suivante :

```

using System.Data.Entity;

namespace WPFwithEFSample
{
    public class ProductContext : DbContext
    {
        public DbSet<Category> Categories { get; set; }
        public DbSet<Product> Products { get; set; }
    }
}

```

Compilez le projet.

### Option 2 : Définir un modèle à l'aide de la première base de données

Cette section montre comment utiliser la première base de données à l'ingénierie inverse de votre modèle à partir d'une base de données à l'aide du Concepteur EF. Si vous avez terminé la section précédente (**Option 1 : définir**

un modèle à l'aide de **Code First**), puis ignorez cette section et passer directement à la **le chargement différé** section.

#### Créer une base de données existante

En général, lorsque vous ciblez une base de données existante, qu'il est déjà créé, mais pour cette procédure pas à pas, nous devons créer une base de données pour accéder à.

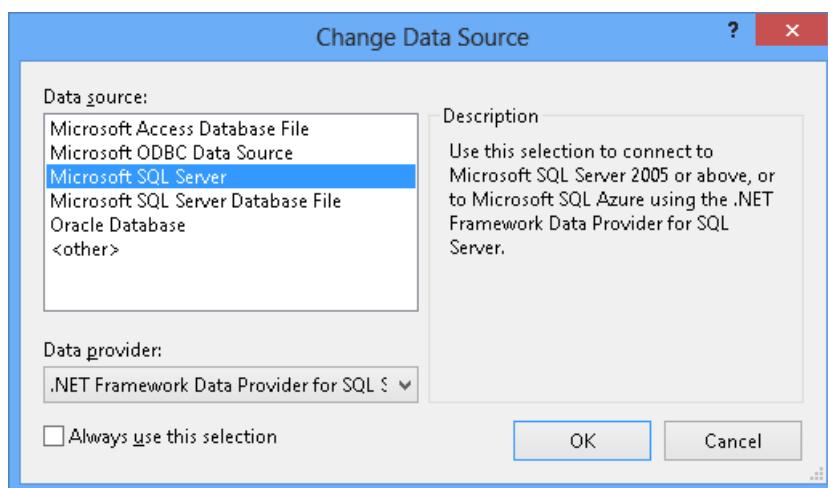
Le serveur de base de données qui est installé avec Visual Studio est différent selon la version de Visual Studio que vous avez installé :

- Si vous utilisez Visual Studio 2010, vous créerez une base de données SQL Express.
- Si vous utilisez Visual Studio 2012, vous créerez un **LocalDB** base de données.

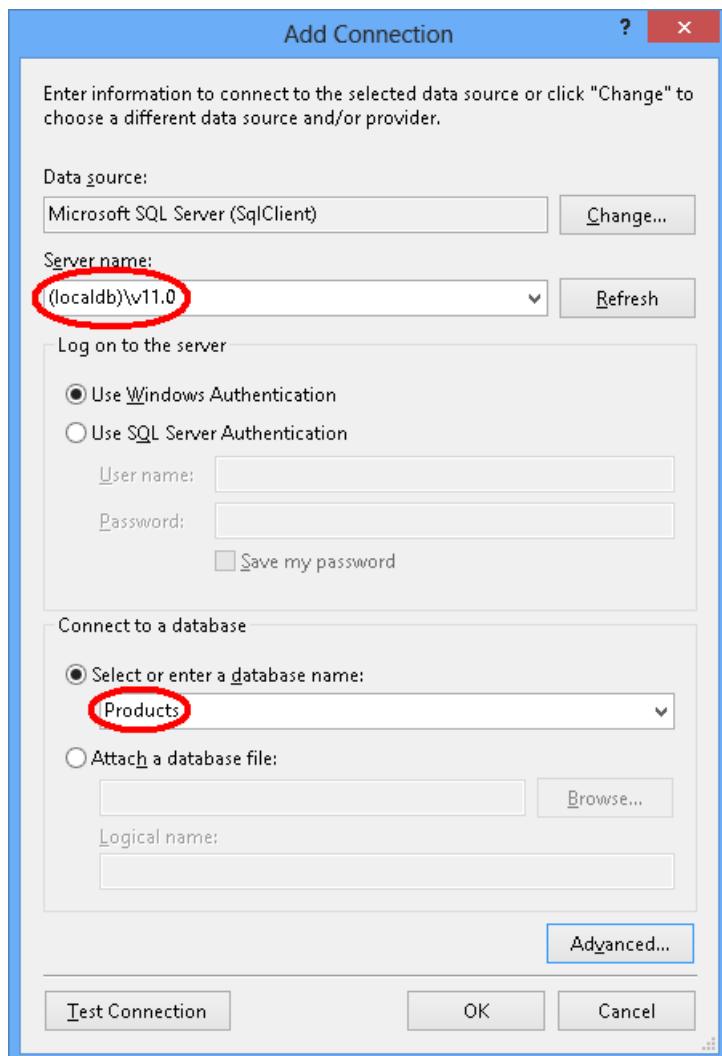
Procérons à générer la base de données.

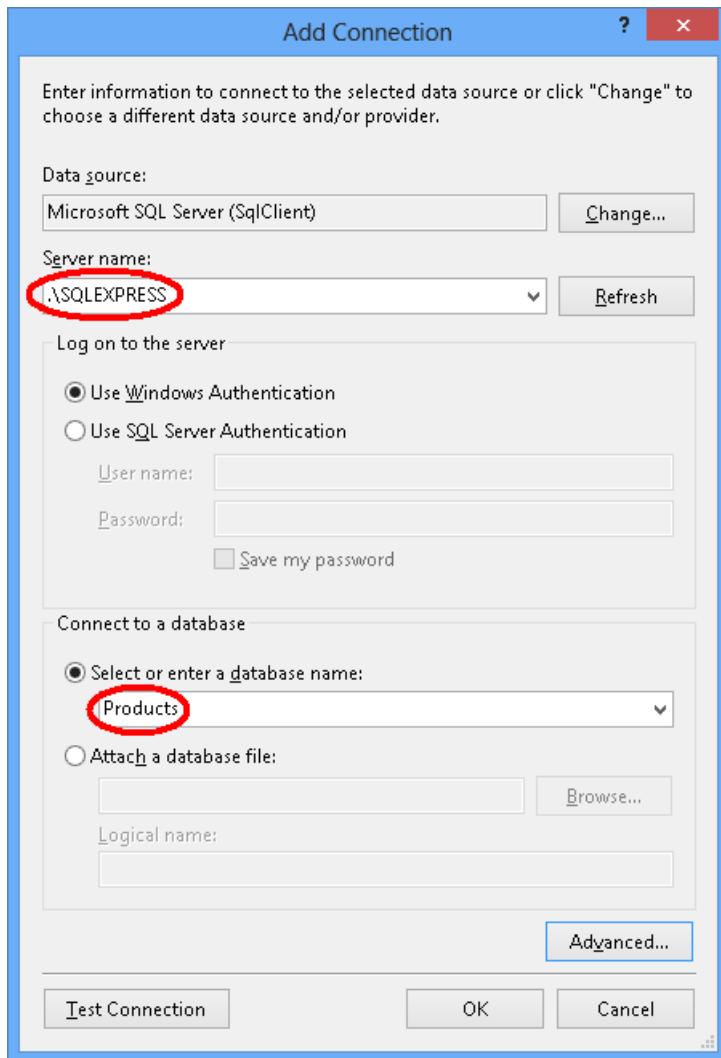
- **Vue -> Explorateur de serveurs**

- Cliquez avec le bouton droit sur **des connexions de données -> ajouter une connexion...**
- Si vous n'avez pas connecté à une base de données à partir de l'Explorateur de serveurs avant que vous devez sélectionner Microsoft SQL Server comme source de données

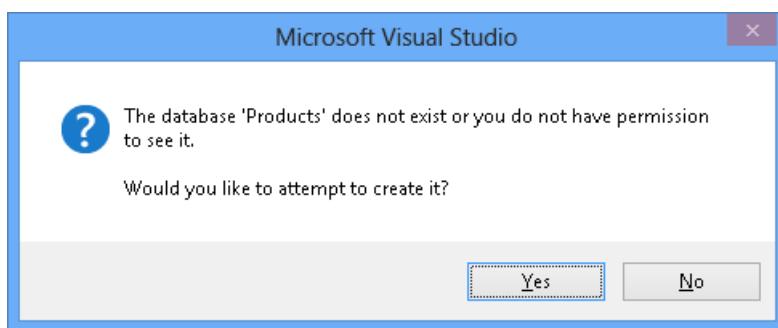


- Se connecter à la base de données locale ou de SQL Express, en fonction de celles que vous avez installé, puis entrez **produits** en tant que le nom de la base de données





- Sélectionnez **OK** et vous demandera si vous souhaitez créer une base de données, sélectionnez **Oui**



- La nouvelle base de données s'affiche maintenant dans l'Explorateur de serveurs, avec le bouton droit dessus et sélectionnez **nouvelle requête**
- Copiez le code SQL suivant dans la nouvelle requête, puis avec le bouton droit sur la requête, puis sélectionnez **Execute**

```

CREATE TABLE [dbo].[Categories] (
    [CategoryId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    CONSTRAINT [PK_dbo.Categories] PRIMARY KEY ([CategoryId])
)

CREATE TABLE [dbo].[Products] (
    [ProductId] [int] NOT NULL IDENTITY,
    [Name] [nvarchar](max),
    [CategoryId] [int] NOT NULL,
    CONSTRAINT [PK_dbo.Products] PRIMARY KEY ([ProductId])
)

CREATE INDEX [IX_CategoryId] ON [dbo].[Products]([CategoryId])

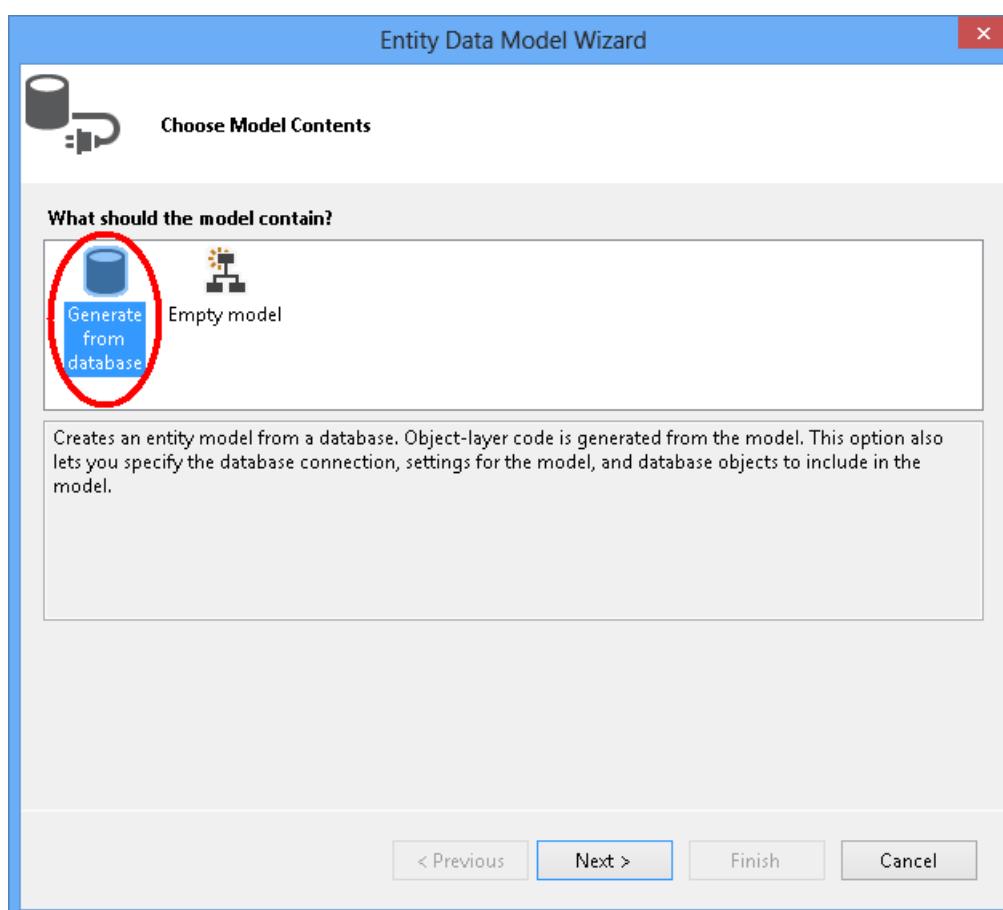
ALTER TABLE [dbo].[Products] ADD CONSTRAINT [FK_dbo.Products_dbo.Categories_CategoryId] FOREIGN KEY
([CategoryId]) REFERENCES [dbo].[Categories] ([CategoryId]) ON DELETE CASCADE

```

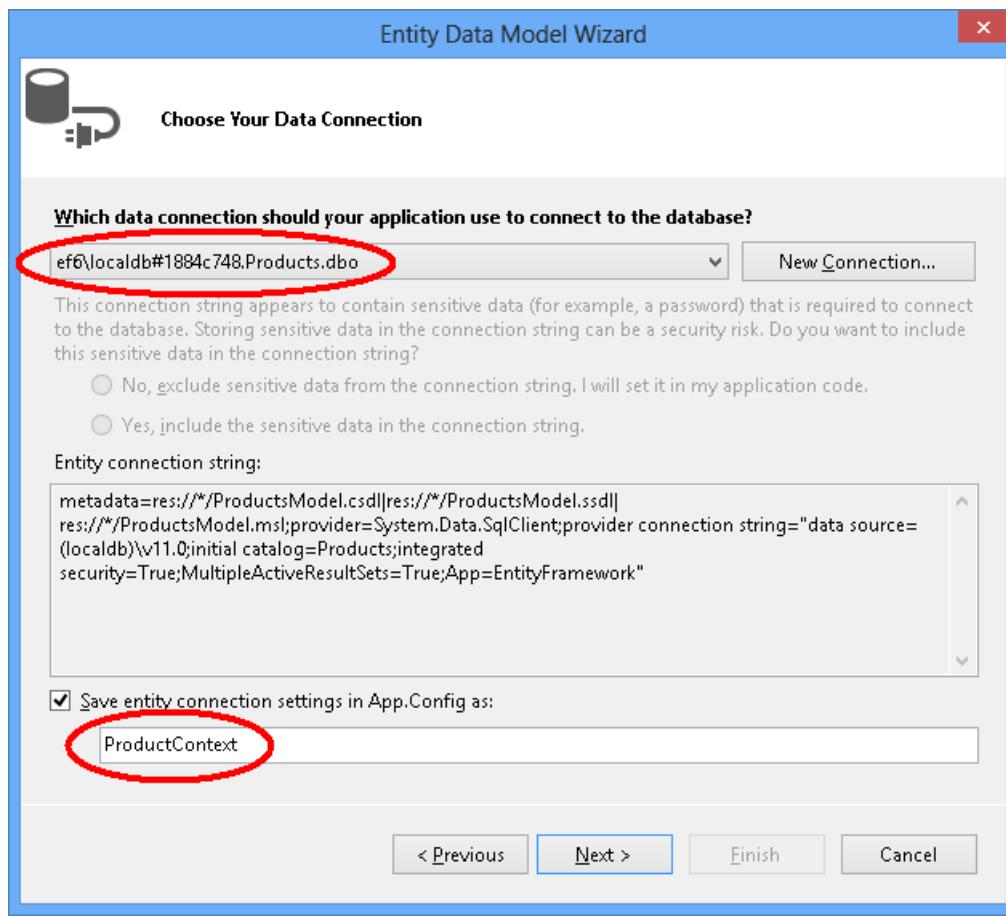
#### Modèle d'ingénierie à rebours

Nous allons utiliser Entity Framework Designer, qui est inclus dans le cadre de Visual Studio, pour créer notre modèle.

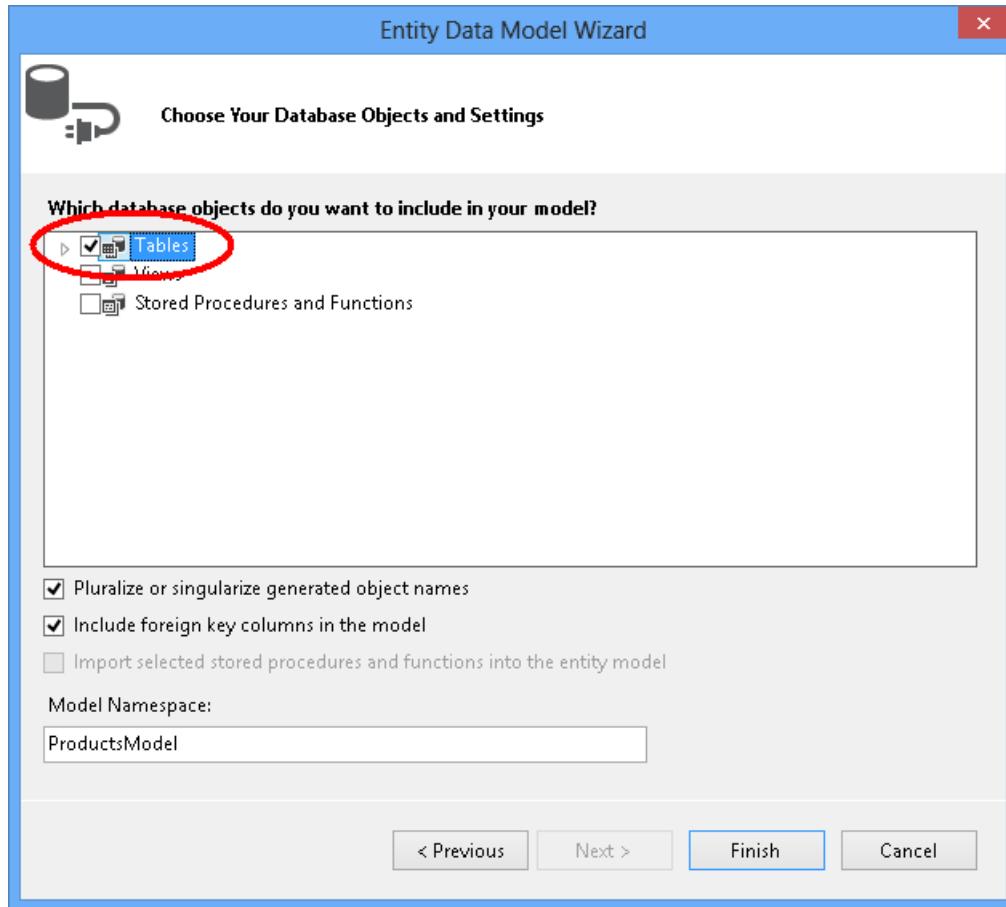
- **Projet -> ajouter un nouvel élément...**
- Sélectionnez **données** dans le menu de gauche, puis **ADO.NET Entity Data Model**
- Entrez **ProductModel** comme nom et cliquez sur **OK**
- Cette opération lance le **Assistant Entity Data Model**
- Sélectionnez **générer à partir de la base de données** et cliquez sur **suivant**



- Sélectionnez la connexion à la base de données que vous avez créé dans la première section, entrez **ProductContext** comme nom de la chaîne de connexion et cliquez sur **suivant**



- Cliquez sur la case à cocher en regard de « Tables » pour importer toutes les tables, cliquez sur « Terminer »



Une fois que le processus d'ingénierie à rebours est terminé le nouveau modèle est ajouté à votre projet et ouvert pour l'afficher dans le Concepteur d'Entity Framework. Un fichier App.config a également été ajouté à votre projet avec les détails de connexion pour la base de données.

## Étapes supplémentaires dans Visual Studio 2010

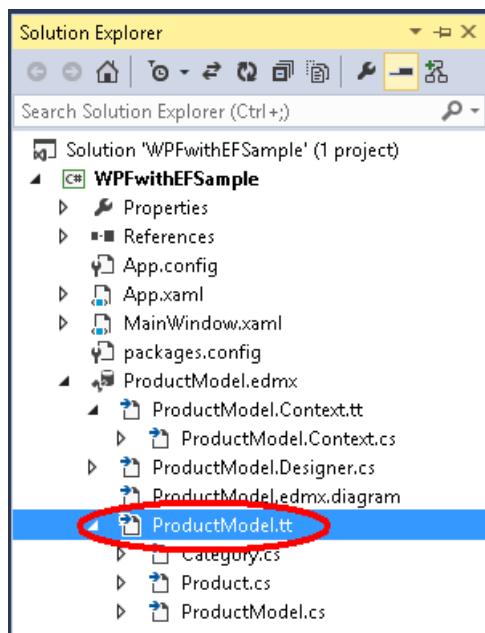
Si vous travaillez dans Visual Studio 2010 vous devrez mettre à jour le Concepteur EF pour utiliser la génération de code EF6.

- Avec le bouton droit sur un endroit vide de votre modèle dans le Concepteur EF et sélectionnez **ajouter un élément de génération de Code...**
- Sélectionnez **modèles en ligne** dans le menu de gauche et recherchez **DbContext**
- Sélectionnez le **EF 6.x générateur DbContext pour C#**, entrez **ProductsModel** comme nom et cliquez sur Ajouter

## Mise à jour de la génération de code pour la liaison de données

Entity Framework génère du code à partir de votre modèle à l'aide de modèles T4. Les modèles fournis avec Visual Studio ou téléchargé à partir de la galerie Visual Studio sont destinés à usage général. Cela signifie que les entités générées à partir de ces modèles ont `ICollection<T>` propriétés. Toutefois, lors de données de la liaison à l'aide de WPF il est souhaitable d'utiliser **ObservableCollection** pour les propriétés de collection afin que WPF peut effectuer le suivi de modifications apportées aux collections. Dans cette optique, nous allons pour modifier les modèles pour utiliser ObservableCollection.

- Ouvrez le **Explorateur de solutions** et recherchez **ProductModel.edmx** fichier
- Rechercher la **ProductModel.tt** fichier qui doit être imbriqué sous le fichier ProductModel.edmx



- Double-cliquez sur le fichier ProductModel.tt pour l'ouvrir dans l'éditeur Visual Studio
- Rechercher et remplacer les deux occurrences de «**ICollection**» avec »**ObservableCollection**». Il s'agit trouve environ à lignes 296 et 484.
- Rechercher et remplacer la première occurrence de «**HashSet**» avec »**ObservableCollection**». Cet événement se trouve approximativement à la ligne 50. **Ne le faites pas** remplacer la deuxième occurrence de HashSet figure plus loin dans le code.
- Rechercher et remplacer la seule occurrence de «**System.Collections.Generic**» avec »**System.Collections.ObjectModel**». Cela se trouve approximativement à la ligne 424.
- Enregistrez le fichier ProductModel.tt. Cela doit provoquer le code pour les entités d'être régénérée. Si le code ne régénère pas automatiquement, puis avec le bouton droit sur ProductModel.tt et choisissez « Exécuter un outil personnalisé ».

Si vous ouvrez le fichier Category.cs (qui est imbriqué sous ProductModel.tt), vous devez voir que la collection de

produits a le type **ObservableCollection<produit>**.

Compilez le projet.

## Chargement différé

Le **produits** propriété sur le **catégorie** classe et **catégorie** propriété sur le **produit** classe sont des propriétés de navigation. Dans Entity Framework, les propriétés de navigation permettent de naviguer d'une relation entre deux types d'entités.

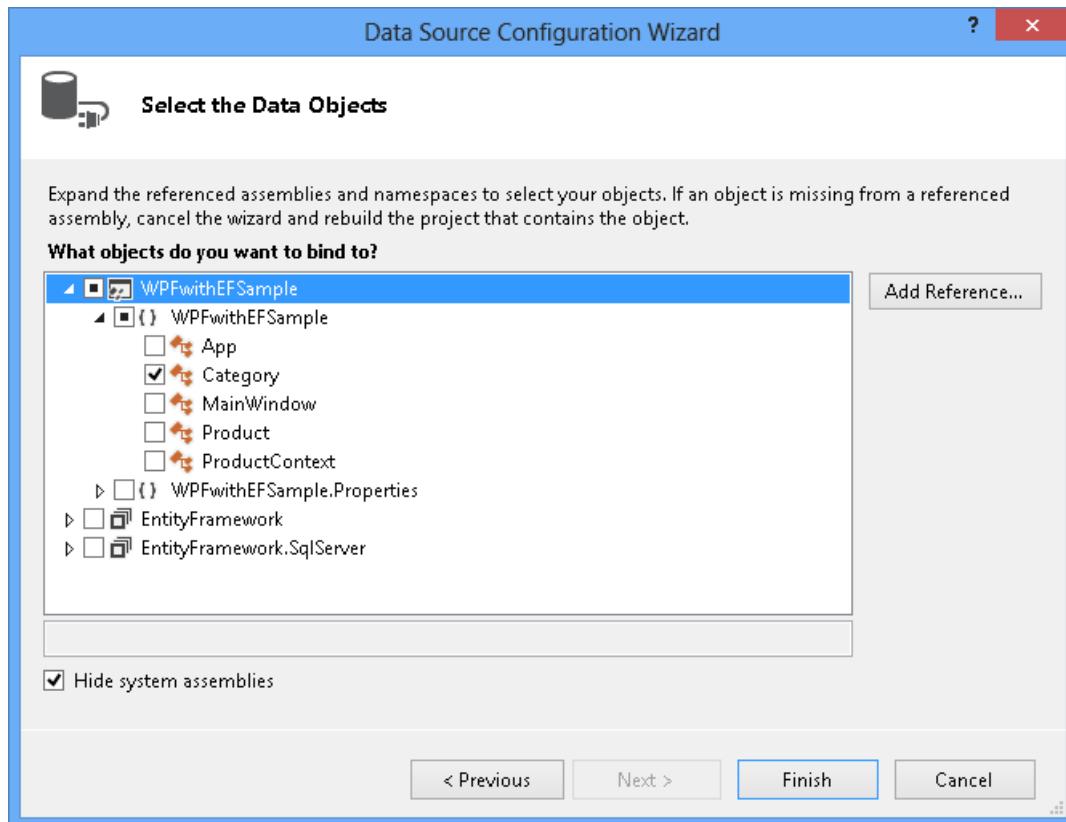
Entity Framework vous offre une option de chargement des entités connexes à partir de la base de données automatiquement la première fois que vous accédez à la propriété de navigation. Avec ce type de chargement (appelé chargement différé), n'oubliez pas que la première fois que vous accédez à chaque propriété de navigation une requête distincte sera exécutée la base de données si le contenu n'est pas déjà dans le contexte.

Lorsque vous utilisez des types d'entités POCO, EF réalise le chargement différé par la création d'instances de types de proxy dérivée pendant l'exécution, puis en remplaçant les propriétés virtuelles dans vos classes pour ajouter le raccordement de chargement. Pour obtenir le chargement différé d'objets connexes, vous devez déclarer les accesseurs Get de propriété en tant que navigation **public** et **virtuels (Overridable** en Visual Basic), et vous classe ne doit pas être **sealed (NotOverridable** en Visual Basic). Lors de la base de données à l'aide des propriétés de navigation premier sont automatiquement effectuées virtuelles pour activer le chargement différé. Dans la section de Code First que nous avons choisi créer les propriétés de navigation virtuelle pour la même raison

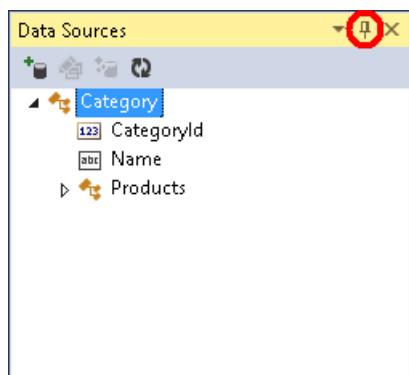
## Lier des objets aux contrôles

Ajoutez les classes qui sont définies dans le modèle en tant que sources de données pour cette application WPF.

- Double-cliquez sur **MainWindow.xaml** dans l'Explorateur de solutions pour ouvrir le formulaire principal
- Dans le menu principal, sélectionnez **projet -> ajouter une nouvelle Source de données...** (dans Visual Studio 2010, vous devez sélectionner **données -> ajouter nouvelle Source de données...**)
- Dans la zone Choisir un Typewindow de Source de données, sélectionnez **objet** et cliquez sur **suivant**
- Dans le, sélectionnez la boîte de dialogue des objets de données, dérouler les **WPFwithEFSample** deux fois, puis sélectionnez **catégorie**  
*Il est inutile de sélectionner le **produit** de source de données, car nous le verrons par le biais le **produit** de propriété sur le **catégorie** source de données*



- Cliquez sur **terminer**.
- La fenêtre Sources de données est ouverte en regard de la fenêtre de MainWindow.xaml *si la fenêtre Sources de données ne s'affichent pas, sélectionnez vue -> autres Windows -> des Sources de données*
- Appuyez sur l'icône d'épingle, afin de la fenêtre Sources de données ne sont pas automatique masquer. Vous devrez peut-être appuyer sur le bouton de rafraîchissement si la fenêtre a été déjà visible.



- Sélectionnez le **catégorie** source de données et faites-la glisser sur le formulaire.

Les éléments suivants s'est produite lorsque nous avons fait glisser cette source :

- Le **categoryViewSource** ressource et le **categoryDataGrid** contrôle ont été ajoutées pour XAML
- La propriété DataContext sur l'élément de grille parent a été définie sur « {StaticResource categoryViewSource} ». Le **categoryViewSource** ressources servant de source de liaison externe\élément de grille parent. Les éléments internes de la grille puis héritent la valeur DataContext grille (propriété ItemsSource de la categoryDataGrid est définie sur « {Binding} »)

```

<Window.Resources>
    <CollectionViewSource x:Key="categoryViewSource"
        d:DesignSource="{d:DesignInstance {x:Type local:Category}, CreateList=True}"/>
</Window.Resources>
<Grid DataContext="{StaticResource categoryViewSource}">
    <DataGrid x:Name="categoryDataGrid" AutoGenerateColumns="False" EnableRowVirtualization="True"
        ItemsSource="{Binding}" Margin="13,13,43,191"
        RowDetailsVisibilityMode="VisibleWhenSelected">
        <DataGrid.Columns>
            <DataGridTextColumn x:Name="categoryIdColumn" Binding="{Binding CategoryId}"
                Header="Category Id" Width="SizeToHeader"/>
            <DataGridTextColumn x:Name="nameColumn" Binding="{Binding Name}"
                Header="Name" Width="SizeToHeader"/>
        </DataGrid.Columns>
    </DataGrid>
</Grid>

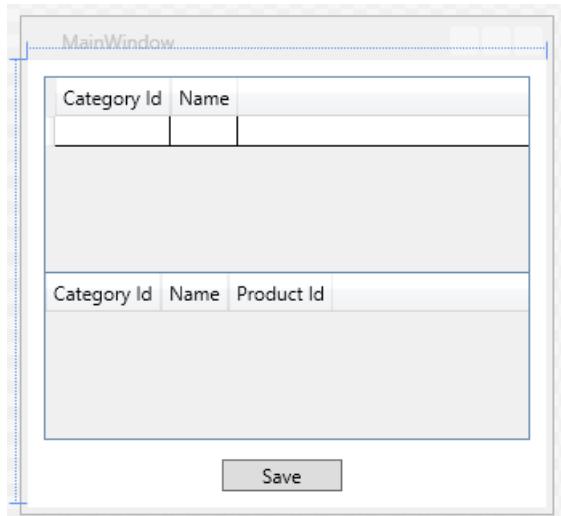
```

## Ajout d'une grille de détails

Maintenant que nous avons une grille pour afficher les catégories de nous allons ajouter une grille de détails pour afficher les produits associés.

- Sélectionnez le **produits** propriété sous la **catégorie** source de données et faites-la glisser sur le formulaire.
  - Le **categoryProductsViewSource** ressource et **productDataGrid** grille sont ajoutés à XAML
  - Le chemin de liaison pour cette ressource est défini pour les produits
  - Infrastructure de liaison de données WPF permet de s'assurer que seuls les produits liés à la catégorie sélectionnée apparaissent dans **productDataGrid**
- Dans la boîte à outils, faites glisser **bouton** une session sur le formulaire. Définir le **nom** propriété **buttonSave** et **contenu** propriété **enregistrer**.

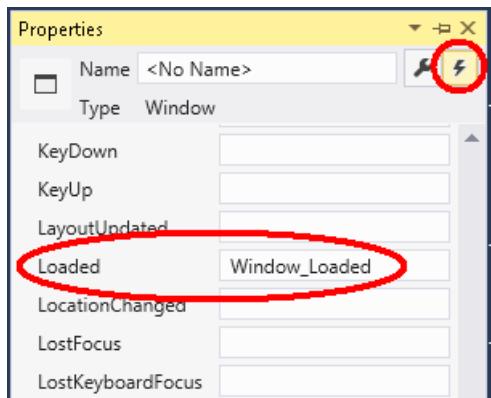
Le formulaire doit ressembler à ceci :



## Ajoutez le Code qui gère l'Interaction de données

Il est temps d'ajouter des gestionnaires d'événements à la fenêtre principale.

- Dans la fenêtre XAML, cliquez sur le <**fenêtre**> élément, cette opération sélectionne la fenêtre principale
- Dans le **propriétés** fenêtre choisissez **événements** en haut à droite, puis double-cliquez sur la zone de texte à droite de la **Loaded** étiquette



- Également ajouter le **cliquez sur** événement pour le **enregistrer** bouton en double-cliquant sur le bouton Enregistrer dans le concepteur.

Cela vous amène à du code-behind pour le formulaire, nous allons maintenant modifier le code pour utiliser le ProductContext pour accéder aux données. Mettre à jour le code pour le MainWindow comme indiqué ci-dessous.

Le code déclare une instance d'exécution longue de **ProductContext**. Le **ProductContext** objet est utilisé pour interroger et enregistrer les données dans la base de données. Le **Dispose()** sur le **ProductContext** instance est ensuite appelée à partir de l'élément substitué **OnClosing** (méthode). Les commentaires du code fournissent des informations sur ce fait le code.

```

using System.Data.Entity;
using System.Linq;
using System.Windows;

namespace WPFwithEFSample
{
    public partial class MainWindow : Window
    {
        private ProductContext _context = new ProductContext();
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            System.Windows.Data.CollectionViewSource categoryViewSource =
                ((System.Windows.Data.CollectionViewSource)(this.FindResource("categoryViewSource")));

            // Load is an extension method on IQueryable,
            // defined in the System.Data.Entity namespace.
            // This method enumerates the results of the query,
            // similar to ToList but without creating a list.
            // When used with Linq to Entities this method
            // creates entity objects and adds them to the context.
            _context.Categories.Load();

            // After the data is loaded call the DbSet<T>.Local property
            // to use the DbSet<T> as a binding source.
            categoryViewSource.Source = _context.Categories.Local;
        }

        private void buttonSave_Click(object sender, RoutedEventArgs e)
        {
            // When you delete an object from the related entities collection
            // (in this case Products), the Entity Framework doesn't mark
            // these child entities as deleted.
            // Instead, it removes the relationship between the parent and the child
            // by setting the parent reference to null.
            // So we manually have to delete the products
            // that have a Category reference set to null.
        }
    }
}

```

```

// The following code uses LINQ to Objects
// against the Local collection of Products.
// The ToList call is required because otherwise the collection will be modified
// by the Remove call while it is being enumerated.
// In most other situations you can use LINQ to Objects directly
// against the Local property without using ToList first.
foreach (var product in _context.Products.Local.ToList())
{
    if (product.Category == null)
    {
        _context.Products.Remove(product);
    }
}

_context.SaveChanges();
// Refresh the grids so the database generated values show up.
this.categoryDataGrid.Items.Refresh();
this.productsDataGrid.Items.Refresh();
}

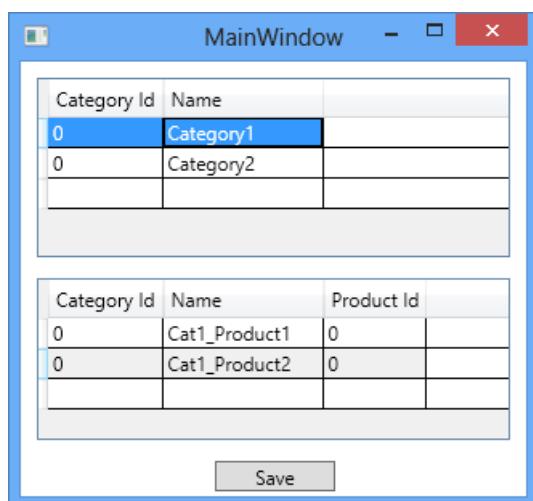
protected override void OnClosing(System.ComponentModel.CancelEventArgs e)
{
    base.OnClosing(e);
    this._context.Dispose();
}
}

}

```

## Tester l'Application WPF

- Compilez et exécutez l'application. Si vous avez utilisé un Code First, vous verrez qu'un **WPFwithEFSample.ProductContext** base de données est créée pour vous.
- Entrez un nom de catégorie dans les noms de produit et de la grille supérieure dans la grille inférieure *n'entrez pas quoi que ce soit dans les colonnes ID, car la clé primaire est générée par la base de données*



- Appuyez sur la **enregistrer** bouton pour enregistrer les données dans la base de données

Après l'appel à du DbContext **SaveChanges()**, les ID sont remplies avec les valeurs de la base de données générée. Étant donné que nous avons appelé **Refresh()** après **SaveChanges()** le **DataGrid** contrôles sont mis à jour avec les nouvelles valeurs également.

The screenshot shows a Windows application window titled "MainWindow". Inside the window, there are two data grids and a "Save" button.

The first data grid has columns "Category Id" and "Name". It contains three rows:

| Category Id | Name      |
|-------------|-----------|
| 3           | Category1 |
| 4           | Category2 |
|             |           |

The second data grid has columns "Category Id", "Name", and "Product Id". It contains three rows:

| Category Id | Name          | Product Id |
|-------------|---------------|------------|
| 4           | Cat2_Product1 | 5          |
| 4           | Cat2_Product2 | 6          |
|             |               |            |

Below the data grids is a "Save" button.

## Ressources supplémentaires

Pour en savoir plus sur la liaison de données pour les collections à l'aide de WPF, consultez [cette rubrique](#) dans la documentation WPF.

# Utilisation d'entités déconnectées

11/10/2019 • 4 minutes to read

Dans une application Entity Framework, une classe de contexte détecte les changements des entités suivies. La méthode `SaveChanges` conserve les changements détectés par le contexte dans la base de données. Quand vous utilisez des applications multicouches, les objets d'entité sont généralement modifiés quand ils sont déconnectés du contexte, et vous devez choisir à la fois comment suivre les changements et comment les signaler au contexte. Cette rubrique décrit les différentes options disponibles quand vous utilisez Entity Framework avec des entités déconnectées.

## Frameworks de service web

Les technologies de services web prennent généralement en charge des modèles pouvant servir à conserver les changements sur des objets déconnectés individuels. Par exemple, l'API Web ASP.NET vous permet de codifier des actions de contrôleur pouvant inclure des appels à EF pour conserver les changements d'un objet dans une base de données. En fait, les outils d'API Web dans Visual Studio facilitent la génération automatique d'un contrôleur API Web à partir de votre modèle Entity Framework 6. Pour plus d'informations, consultez [Utilisation de l'API Web avec Entity Framework 6](#).

Historiquement, plusieurs autres technologies de services web ont offert une intégration à Entity Framework, par exemple, [WCF Data Services](#) et [Services RIA](#).

## API EF de bas niveau

Si vous ne voulez pas utiliser de solution multicouche existante ou que vous voulez personnaliser une action de contrôleur dans les services d'API Web, Entity Framework fournit des API qui vous permettent d'appliquer les changements d'une couche déconnectée. Pour plus d'informations, consultez [Add, Attach et état d'entité](#)

## Entités de suivi automatique

Suivre les changements sur des graphiques d'entité arbitraires déconnectés du contexte EF n'est pas chose simple. Le problème a été partiellement résolu par le modèle de génération de code des entités de suivi automatique. Ce modèle génère des classes d'entité qui contiennent une logique pour suivre les changements d'une couche déconnectée, comme l'état dans les entités elles-mêmes. Un ensemble de méthodes d'extension est également généré pour appliquer ces changements à un contexte.

Ce modèle peut être utilisé avec des modèles créés à l'aide d'EF Designer, mais ne peut pas être utilisé avec des modèles Code First. Pour plus d'informations, consultez [Entités de suivi automatique](#).

### IMPORTANT

Nous ne recommandons plus d'utiliser le modèle des entités de suivi automatique. Il reste disponible uniquement pour prendre en charge les applications existantes. Si votre application doit utiliser des graphiques d'entités déconnectés, envisagez d'autres alternatives comme les [Entités traçables](#), qui présentent une technologie similaire aux entités de suivi automatique, mais développée de manière plus active par la communauté, ou bien l'écriture de code personnalisé à l'aide des API de suivi de changements de bas niveau.

# Entités de suivi automatique

11/10/2019 • 11 minutes to read

## IMPORTANT

Nous ne recommandons plus d'utiliser le modèle des entités de suivi automatique. Il reste disponible uniquement pour prendre en charge les applications existantes. Si votre application doit utiliser des graphiques d'entités déconnectés, envisagez d'autres alternatives comme les [Entités traçables](#), qui présentent une technologie similaire aux entités de suivi automatique, mais développée de manière plus active par la communauté, ou bien l'écriture de code personnalisé à l'aide des API de suivi de changements de bas niveau.

Dans une application Entity Framework, c'est un contexte qui est responsable du suivi des changements dans vos objets. Vous utilisez la méthode `SaveChanges` pour conserver les changements dans la base de données. Quand vous utilisez des applications multicouches, les objets d'entité sont généralement déconnectés du contexte et vous devez choisir à la fois comment suivre les changements et comment les signaler au contexte. Les entités de suivi automatique peuvent vous aider à suivre les changements dans n'importe quelle couche, puis de replacer ces changements dans un contexte pour les enregistrer.

Utilisez les entités de suivi automatique uniquement si le contexte n'est pas disponible sur la couche où ont lieu les changements du graphique d'objet. Si le contexte est disponible, il s'occupe de suivre les changements, vous n'avez donc pas besoin d'utiliser les entités de suivi automatique.

Cet élément de modèle génère deux fichiers .tt (modèle de texte) :

- Le fichier **<nom du modèle>.tt** génère les types d'entité et une classe d'assistance qui contient la logique de suivi des changements utilisée par les entités de suivi automatique et les méthodes d'extension qui permettent de définir l'état sur les entités de suivi automatique.
- Le fichier **<nom du modèle>.Context.tt** génère un contexte dérivé et une classe d'extension qui contient des méthodes **ApplyChanges** pour les classes **ObjectContext** et **ObjectSet**. Ces méthodes examinent les informations de suivi des modifications contenues dans le graphique des entités de suivi automatique pour déduire l'ensemble des opérations à effectuer pour enregistrer les modifications dans la base de données.

## Bien démarrer

Pour commencer, visitez la page [Procédure pas à pas des entités de suivi automatique](#).

## Points fonctionnels à prendre en considération lors de l'utilisation d'entités de suivi automatique

## IMPORTANT

Nous ne recommandons plus d'utiliser le modèle des entités de suivi automatique. Il reste disponible uniquement pour prendre en charge les applications existantes. Si votre application doit utiliser des graphiques d'entités déconnectés, envisagez d'autres alternatives comme les [Entités traçables](#), qui présentent une technologie similaire aux entités de suivi automatique, mais développée de manière plus active par la communauté, ou bien l'écriture de code personnalisé à l'aide des API de suivi de changements de bas niveau.

Tenez compte des éléments suivants lors de l'utilisation d'entités de suivi automatique :

- Assurez-vous que votre projet client possède une référence à l'assembly contenant les types d'entités. Si

vous ajoutez uniquement la référence de service au projet client, celui-ci utilisera les types de proxy WCF et non les types réels d'entité de suivi automatique. Cela signifie que vous n'aurez pas accès aux fonctionnalités de notification automatisée qui gèrent le suivi des entités sur client. Si vous ne souhaitez pas inclure les types d'entités, vous devrez définir manuellement les informations de suivi des modifications sur le client pour les modifications à renvoyer au service.

- Les appels aux opérations de service doivent être sans état et créer une nouvelle instance du contexte de l'objet. Nous vous recommandons aussi de créer un contexte d'objet dans un bloc **using**.
- Quand vous envoyez le graphique modifié sur le client au service et que vous voulez continuer à utiliser le même graphique sur le client, vous devez effectuer une itération manuelle au sein du graphique et appeler la méthode **AcceptChanges** sur chaque objet pour réinitialiser le suivi des changements.

Si des objets dans votre graphique contiennent des propriétés avec des valeurs générées par la base de données (par exemple, des valeurs d'identité ou de concurrence), l'Entity Framework remplace les valeurs de ces propriétés par les valeurs générées par la base de données après l'appel de la méthode **SaveChanges**. Vous pouvez implémenter votre opération de service pour retourner des objets enregistrés ou une liste de valeurs de propriété générées pour les objets au client. Le client devra ensuite remplacer les instances d'objet ou valeurs de propriété d'objet par les objets ou valeurs de propriété retournés à partir de l'opération de service.

- La fusion de graphiques à partir de plusieurs demandes de service peut introduire des objets avec des valeurs de clés dupliquées dans le graphique résultant. L'Entity Framework ne supprime pas les objets avec des clés en double quand vous appelez la méthode **ApplyChanges**, mais lève une exception. Pour éviter les graphiques avec des valeurs de clé en double, suivez l'un des modèles décrits dans le blog suivant : [Self-Tracking Entities: ApplyChanges and duplicate entities](#).
- Lorsque vous modifiez la relation entre les objets en définissant la propriété de clé étrangère, la propriété de navigation de référence a la valeur Null et n'est pas synchronisée sur l'entité principale appropriée sur le client. Dès que le graphique est attaché au contexte d'objet (par exemple, une fois que vous avez appelé la méthode **ApplyChanges**), les propriétés de clé étrangère et les propriétés de navigation sont synchronisées.

Le fait que la propriété de navigation de référence ne soit pas synchronisée avec l'objet principal approprié peut poser un problème si vous avez spécifié la suppression en cascade dans la relation de clé étrangère. Si vous supprimez l'objet principal, la suppression ne sera pas propagée aux objets dépendants. Si vous avez spécifié des suppressions en cascade, utilisez les propriétés de navigation pour modifier les relations au lieu de définir la propriété de clé étrangère.

- Les entités de suivi automatique ne sont pas activées pour effectuer un chargement différé.
- La sérialisation binaire et la sérialisation en objets de gestion d'état ASP.NET ne sont pas prises en charge par les entités de suivi automatique. Toutefois, vous pouvez personnaliser le modèle pour ajouter la prise en charge de la sérialisation binaire. Pour plus d'informations, consultez [Utilisation de la sérialisation binaire et de ViewState avec les entités de suivi automatique](#).

## Considérations relatives à la sécurité

Les considérations de sécurité suivantes doivent être prises en compte quand vous utilisez les entités de suivi automatique :

- Un service ne doit pas compter sur les requêtes pour récupérer ou mettre à jour les données d'un client non approuvé ou via un canal non approuvé. Un client doit être authentifié : un canal sécurisé doit être utilisé ou bien une enveloppe de message. Les requêtes de clients pour mettre à jour ou récupérer des données doivent

être validées pour vérifier qu'elles sont conformes aux modifications attendues et légitimes pour le scénario donné.

- Évitez d'utiliser des informations sensibles comme clés d'entité (par exemple, des numéros de sécurité sociale). Cela limite l'éventualité de sérialiser par inadvertance des informations sensibles dans les graphiques d'entité de suivi automatique sur un client qui n'est pas d'un niveau de confiance suffisant. Avec les associations indépendantes, la clé d'origine d'une entité associée à celle qui est sérialisée peut être également envoyée au client.
- Pour éviter la propagation de messages d'exception contenant des données sensibles vers la couche cliente, les appels à **ApplyChanges** et **SaveChanges** sur la couche serveur doivent être wrappés dans le code de gestion des exceptions.

# Procédure pas à pas des entités de suivi automatique

23/11/2019 • 21 minutes to read

## IMPORTANT

Nous ne recommandons plus d'utiliser le modèle des entités de suivi automatique. Il reste disponible uniquement pour prendre en charge les applications existantes. Si votre application doit utiliser des graphiques d'entités déconnectés, envisagez d'autres alternatives comme les [Entités traçables](#), qui présentent une technologie similaire aux entités de suivi automatique, mais développée de manière plus active par la communauté, ou bien l'écriture de code personnalisé à l'aide des API de suivi de changements de bas niveau.

Cette procédure pas à pas illustre le scénario dans lequel un service Windows Communication Foundation (WCF) expose une opération qui retourne un graphique d'entité. Ensuite, une application cliente manipule ce graphique et soumet les modifications à une opération de service qui valide et enregistre les mises à jour dans une base de données à l'aide de Entity Framework.

Avant d'effectuer cette procédure pas à pas, veillez à lire la page [entités de suivi automatique](#).

Cette procédure pas à pas effectue les actions suivantes :

- Crée une base de données à laquelle accéder.
- Crée une bibliothèque de classes qui contient le modèle.
- Bascule vers le modèle générateur d'entité de suivi automatique.
- Déplace les classes d'entité vers un projet distinct.
- Crée un service WCF qui expose des opérations pour interroger et enregistrer des entités.
- Crée des applications clientes (console et WPF) qui consomment le service.

Nous allons utiliser Database First dans cette procédure pas à pas, mais les mêmes techniques s'appliquent également à Model First.

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous avez besoin d'une version récente de Visual Studio.

## Créer une base de données

Le serveur de base de données installé avec Visual Studio diffère selon la version de Visual Studio que vous avez installée :

- Si vous utilisez Visual Studio 2012, vous allez créer une base de données de base de données locale.
- Si vous utilisez Visual Studio 2010, vous allez créer une base de données SQL Express.

Commençons par générer la base de données.

- Ouvrez Visual Studio
- **Vue-> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **connexions de données-> ajouter une connexion...**
- Si vous n'êtes pas connecté à une base de données à partir de Explorateur de serveurs avant de devoir sélectionner **Microsoft SQL Server** comme source de données
- Connectez-vous à la base de données locale ou SQL Express, en fonction de celle que vous avez installée

- Entrez **STESample** comme nom de la base de données
- Sélectionnez **OK**. vous serez invité à créer une nouvelle base de données, sélectionnez **Oui**.
- La nouvelle base de données s'affiche à présent dans Explorateur de serveurs
- Si vous utilisez Visual Studio 2012
  - Dans Explorateur de serveurs, cliquez avec le bouton droit sur la base de données, puis sélectionnez **nouvelle requête**.
  - Copiez le code SQL suivant dans la nouvelle requête, cliquez avec le bouton droit sur la requête et sélectionnez **exécuter**.
- Si vous utilisez Visual Studio 2010
  - Sélectionnez **les données-> l'éditeur Transact SQL-> nouvelle connexion à la requête...**
  - Entrez **.\\SQLEXPRESS** comme nom de serveur, puis cliquez sur **OK**.
  - Sélectionnez la base de données **STESample** dans la liste déroulante en haut de l'éditeur de requête.
  - Copiez le code SQL suivant dans la nouvelle requête, cliquez avec le bouton droit sur la requête et sélectionnez **Exécuter SQL**.

```

CREATE TABLE [dbo].[Blogs] (
    [BlogId] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url] NVARCHAR (200) NULL,
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [dbo].[Posts] (
    [PostId] INT IDENTITY (1, 1) NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [dbo].[Blogs] ([BlogId])
ON DELETE CASCADE
);

SET IDENTITY_INSERT [dbo].[Blogs] ON
INSERT INTO [dbo].[Blogs] ([BlogId], [Name], [Url]) VALUES (1, N'ADO.NET Blog', N'blogs.msdn.com/adonet')
SET IDENTITY_INSERT [dbo].[Blogs] OFF
INSERT INTO [dbo].[Posts] ([Title], [Content], [BlogId]) VALUES (N'Intro to EF', N'Interesting stuff...', 1)
INSERT INTO [dbo].[Posts] ([Title], [Content], [BlogId]) VALUES (N'What is New', N'More interesting stuff...', 1)
  
```

## Créer le modèle

Tout d'abord, nous avons besoin d'un projet dans lequel placer le modèle.

- **Fichier-> nouveau>...**
- Sélectionnez **Visual C#** dans le volet gauche, puis **bibliothèque de classes**
- Entrez **STESample** comme nom et cliquez sur **OK**.

Nous allons maintenant créer un modèle simple dans le concepteur EF pour accéder à la base de données :

- **Projet-> ajouter un nouvel élément...**
- Dans le volet gauche, sélectionnez **données**, puis **ADO.NET Entity Data Model**
- Entrez **BloggingModel** comme nom et cliquez sur **OK**.
- Sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.
- Entrez les informations de connexion pour la base de données que vous avez créée dans la section précédente.
- Entrez **BloggingContext** comme nom de la chaîne de connexion, puis cliquez sur **suivant**.

- Cochez la case en regard de **tables**, puis cliquez sur **Terminer**.

## Basculer vers la génération de code STE

Maintenant, nous devons désactiver la génération de code et l'échange par défaut pour les entités de suivi automatique.

### Si vous utilisez Visual Studio 2012

- Développez **BloggingModel.edmx** dans **Explorateur de solutions** et supprimez le **BloggingModel.TT** et le **BloggingModel.Context.TT** *la génération de code par défaut est désactivée*.
- Cliquez avec le bouton droit sur une zone vide de l'aire du concepteur EF, puis sélectionnez **Ajouter un élément de génération de code...**
- Sélectionnez **en ligne** dans le volet gauche et recherchez le **Générateur Ste**
- Sélectionnez le **Générateur Ste pour le modèle C#**, entrez **STETemplate** comme nom et cliquez sur **Ajouter**.
- Les fichiers **STETemplate.TT** et **STETemplate.Context.TT** sont ajoutés imbriqués sous le fichier **BloggingModel.edmx**

### Si vous utilisez Visual Studio 2010

- Cliquez avec le bouton droit sur une zone vide de l'aire du concepteur EF, puis sélectionnez **Ajouter un élément de génération de code...**
- Sélectionnez **code** dans le volet gauche, puis **Générateur d'entité de suivi automatique ADO.net**
- Entrez **STETemplate** comme nom et cliquez sur **Ajouter**.
- Les fichiers **STETemplate.TT** et **STETemplate.Context.TT** sont ajoutés directement à votre projet

## Déplacer les types d'entités dans un projet distinct

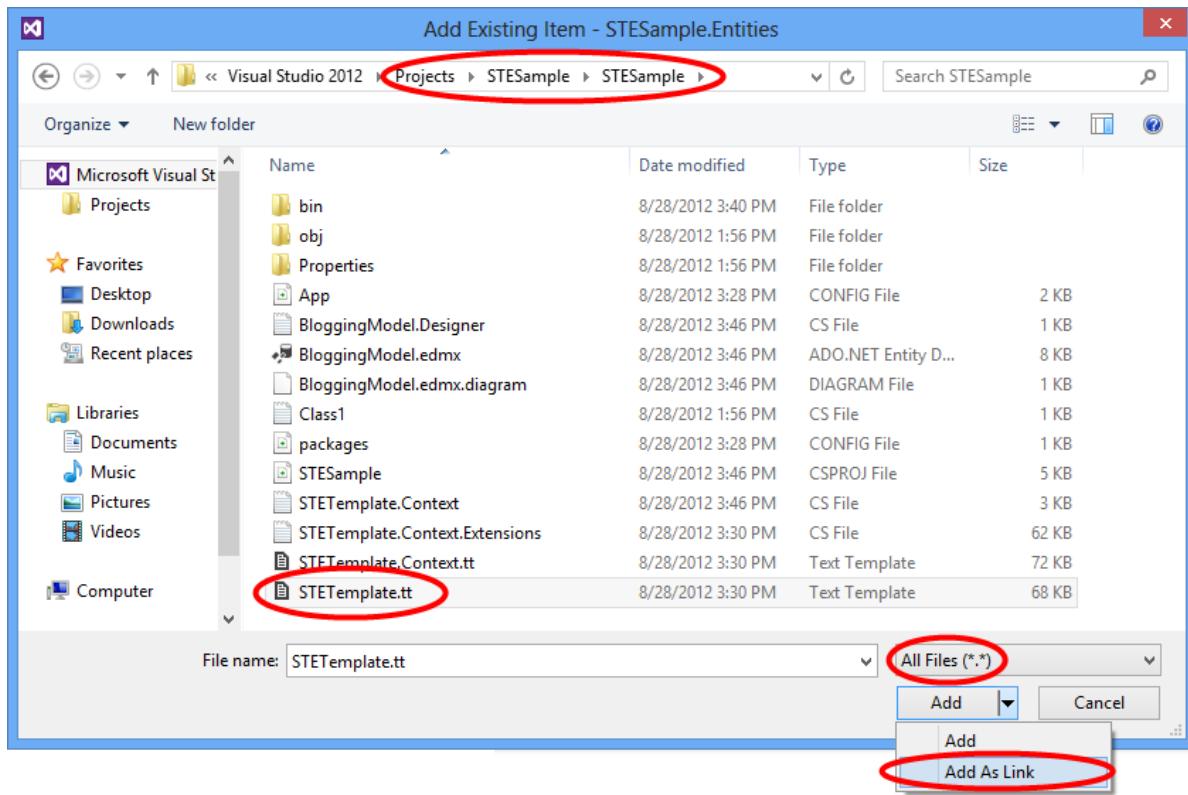
Pour utiliser les entités de suivi automatique, notre application cliente a besoin d'accéder aux classes d'entité générées à partir de notre modèle. Étant donné que nous ne souhaitons pas exposer l'ensemble du modèle à l'application cliente, nous allons déplacer les classes d'entité dans un projet distinct.

La première étape consiste à arrêter la génération de classes d'entité dans le projet existant :

- Cliquez avec le bouton droit sur **STETemplate.TT** dans **Explorateur de solutions** et sélectionnez **Propriétés**
- Dans la fenêtre **Propriétés**, effacez **TextTemplatingFileGenerator** de la propriété **CustomTool**
- Développez **STETemplate.TT** dans **Explorateur de solutions** et supprimez tous les fichiers imbriqués sous celui-ci.

Ensuite, nous allons ajouter un nouveau projet et générer les classes d'entité qu'il contient.

- > **de fichier-ajouter un projet de>...**
- Sélectionnez **Visual C#** dans le volet gauche, puis **bibliothèque de classes**
- Entrez **STESample. Entities** comme nom et cliquez sur **OK**.
- **Projet-> ajouter un élément existant...**
- Accédez au dossier du projet **STESample**
- Sélectionnez cette option pour afficher **tous les fichiers (\*.\*)**
- Sélectionnez le fichier **STETemplate.TT**
- Cliquez sur la flèche déroulante à côté du bouton **Ajouter** et sélectionnez **Ajouter en tant que lien**



Nous allons également nous assurer que les classes d'entité sont générées dans le même espace de noms que le contexte. Cela réduit simplement le nombre d'instructions using que nous devons ajouter dans notre application.

- Cliquez avec le bouton droit sur le **STETemplate.TT** lié dans **Explorateur de solutions** et sélectionnez **Propriétés**.
- Dans la fenêtre **Propriétés**, définissez espace de noms de l' **outil personnalisé** sur **STESample**

Le code généré par le modèle STE aura besoin d'une référence à **System. Runtime. Serialization** pour pouvoir être compilé. Cette bibliothèque est nécessaire pour les attributs **DataContract** et **DataMember** WCF utilisés sur les types d'entités sérialisables.

- Cliquez avec le bouton droit sur le projet **STESample. Entities** dans **Explorateur de solutions** puis sélectionnez **Ajouter une référence...**
  - Dans Visual Studio 2012, activez la case à cocher en regard de **System. Runtime. Serialization** , puis cliquez sur **OK** .
  - Dans Visual Studio 2010, sélectionnez **System. Runtime. Serialization** , puis cliquez sur **OK** .

Enfin, le projet avec notre contexte doit avoir une référence aux types d'entité.

- Cliquez avec le bouton droit sur le projet **STESample** dans **Explorateur de solutions** puis sélectionnez **Ajouter une référence...**
  - Dans Visual Studio 2012-sélectionnez **solution** dans le volet gauche, cochez la case en regard de **STESample. Entities** , puis cliquez sur **OK** .
  - Dans Visual Studio 2010-sélectionnez l'onglet**projets** , sélectionnez **STESample. Entities** , puis cliquez sur **OK** .

#### NOTE

Une autre option pour déplacer les types d'entités vers un projet distinct consiste à déplacer le fichier de modèle au lieu de le lier à partir de son emplacement par défaut. Si vous procédez ainsi, vous devrez mettre à jour la variable **FichierEntrée** dans le modèle pour fournir le chemin d'accès relatif au fichier edmx (dans cet exemple, il s'agit de ..\BloggingModel. edmx).

# Créer un service WCF

Maintenant, il est temps d'ajouter un service WCF pour exposer nos données, nous allons commencer par créer le projet.

- > de fichier-ajouter un projet de...
  - Sélectionnez **Visual C#** dans le volet gauche, puis **application de service WCF**.
  - Entrez **STESample. service** comme nom et cliquez sur **OK**.
  - Ajouter une référence à l'assembly **System. Data. Entity**
  - Ajouter une référence aux projets **STESample** et **STESample. Entities**

Nous devons copier la chaîne de connexion EF dans ce projet afin qu'elle soit trouvée lors de l'exécution.

- Ouvrez le fichier **app. config** du projet \*\*STESample \*\*et copiez l'élément **connectionStrings**
- Collez l'élément **connectionStrings** en tant qu'élément enfant de l'élément **configuration** du fichier **Web. config** dans le projet **STESample. service**.

Il est maintenant temps de mettre en œuvre le service réel.

- Ouvrez **IService1.cs** et remplacez le contenu par le code suivant:

```
using System.Collections.Generic;
using System.ServiceModel;

namespace STESample.Service
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        List<Blog> GetBlogs();

        [OperationContract]
        void UpdateBlog(Blog blog);
    }
}
```

- Ouvrez **Service1. svc** et remplacez le contenu par le code suivant:

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;

namespace STESample.Service
{
    public class Service1 : IService1
    {
        /// <summary>
        /// Gets all the Blogs and related Posts.
        /// </summary>
        public List<Blog> GetBlogs()
        {
            using (BloggingContext context = new BloggingContext())
            {
                return context.Blogs.Include("Posts").ToList();
            }
        }

        /// <summary>
        /// Updates Blog and its related Posts.
        /// </summary>
        public void UpdateBlog(Blog blog)
        {
            using (BloggingContext context = new BloggingContext())
            {
                try
                {
                    // TODO: Perform validation on the updated order before applying the changes.

                    // The ApplyChanges method examines the change tracking information
                    // contained in the graph of self-tracking entities to infer the set of operations
                    // that need to be performed to reflect the changes in the database.
                    context.Blogs.ApplyChanges(blog);
                    context.SaveChanges();

                }
                catch (UpdateException)
                {
                    // To avoid propagating exception messages that contain sensitive data to the client
                    tier
                    // calls to ApplyChanges and SaveChanges should be wrapped in exception handling code.
                    throw new InvalidOperationException("Failed to update. Try your request again.");
                }
            }
        }
    }
}

```

## Utiliser le service à partir d'une application console

Nous allons créer une application console qui utilise notre service.

- **Fichier> nouveau>...**
- Sélectionnez **Visual C#** dans le volet gauche, puis **application console** .
- Entrez **STESample. ConsoleTest** comme nom et cliquez sur **OK** .
- Ajouter une référence au projet **STESample. Entities**

Nous avons besoin d'une référence de service à notre service WCF

- Cliquez avec le bouton droit sur le projet **STESample. ConsoleTest** dans **Explorateur de solutions** puis sélectionnez **Ajouter une référence de service...**

- Cliquez sur **découvrir**
- Entrez **BloggingService** comme espace de noms, puis cliquez sur **OK**.

Nous pouvons maintenant écrire du code pour consommer le service.

- Ouvrez **Program.cs** et remplacez le contenu par le code suivant.

```
using STESample.ConsoleTest.BloggingService;
using System;
using System.Linq;

namespace STESample.ConsoleTest
{
    class Program
    {
        static void Main(string[] args)
        {
            // Print out the data before we change anything
            Console.WriteLine("Initial Data:");
            DisplayBlogsAndPosts();

            // Add a new Blog and some Posts
            AddBlogAndPost();
            Console.WriteLine("After Adding:");
            DisplayBlogsAndPosts();

            // Modify the Blog and one of its Posts
            UpdateBlogAndPost();
            Console.WriteLine("After Update:");
            DisplayBlogsAndPosts();

            // Delete the Blog and its Posts
            DeleteBlogAndPost();
            Console.WriteLine("After Delete:");
            DisplayBlogsAndPosts();

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }

        static void DisplayBlogsAndPosts()
        {
            using (var service = new Service1Client())
            {
                // Get all Blogs (and Posts) from the service
                // and print them to the console
                var blogs = service.GetBlogs();
                foreach (var blog in blogs)
                {
                    Console.WriteLine(blog.Name);
                    foreach (var post in blog.Posts)
                    {
                        Console.WriteLine(" - {0}", post.Title);
                    }
                }
            }

            Console.WriteLine();
            Console.WriteLine();
        }

        static void AddBlogAndPost()
        {
            using (var service = new Service1Client())
            {
                // Create a new Blog with a couple of Posts
                var newBlog = new Blog
```

```

    {
        Name = "The New Blog",
        Posts =
        {
            new Post { Title = "Welcome to the new blog"},
            new Post { Title = "What's new on the new blog"}
        }
    };

    // Save the changes using the service
    service.UpdateBlog(newBlog);
}

static void UpdateBlogAndPost()
{
    using (var service = new Service1Client())
    {
        // Get all the Blogs
        var blogs = service.GetBlogs();

        // Use LINQ to Objects to find The New Blog
        var blog = blogs.First(b => b.Name == "The New Blog");

        // Update the Blogs name
        blog.Name = "The Not-So-New Blog";

        // Update one of the related posts
        blog.Posts.First().Content = "Some interesting content...";

        // Save the changes using the service
        service.UpdateBlog(blog);
    }
}

static void DeleteBlogAndPost()
{
    using (var service = new Service1Client())
    {
        // Get all the Blogs
        var blogs = service.GetBlogs();

        // Use LINQ to Objects to find The Not-So-New Blog
        var blog = blogs.First(b => b.Name == "The Not-So-New Blog");

        // Mark all related Posts for deletion
        // We need to call ToList because each Post will be removed from the
        // Posts collection when we call MarkAsDeleted
        foreach (var post in blog.Posts.ToList())
        {
            post.MarkAsDeleted();
        }

        // Mark the Blog for deletion
        blog.MarkAsDeleted();

        // Save the changes using the service
        service.UpdateBlog(blog);
    }
}
}

```

Vous pouvez à présent exécuter l'application pour la voir en action.

- Cliquez avec le bouton droit sur le projet **STESample. ConsoleTest** dans **Explorateur de solutions** puis sélectionnez **Déboguer-> démarrer une nouvelle instance**

La sortie suivante s'affiche lors de l'exécution de l'application.

```
Initial Data:  
ADO.NET Blog  
- Intro to EF  
- What is New  
  
After Adding:  
ADO.NET Blog  
- Intro to EF  
- What is New  
The New Blog  
- Welcome to the new blog  
- What's new on the new blog  
  
After Update:  
ADO.NET Blog  
- Intro to EF  
- What is New  
The Not-So-New Blog  
- Welcome to the new blog  
- What's new on the new blog  
  
After Delete:  
ADO.NET Blog  
- Intro to EF  
- What is New  
  
Press any key to exit...
```

## Utiliser le service à partir d'une application WPF

Nous allons créer une application WPF qui utilise notre service.

- **Fichier>nouveau>...**
- Sélectionnez **Visual C#** dans le volet gauche, puis **application WPF** .
- Entrez **STESample. WPFTest** comme nom et cliquez sur **OK** .
- Ajouter une référence au projet **STESample. Entities**

Nous avons besoin d'une référence de service à notre service WCF

- Cliquez avec le bouton droit sur le projet **STESample. WPFTest** dans **Explorateur de solutions** puis sélectionnez **Ajouter une référence de service...**
- Cliquez sur **découvrir**
- Entrez **BloggingService** comme espace de noms, puis cliquez sur **OK** .

Nous pouvons maintenant écrire du code pour consommer le service.

- Ouvrez **MainWindow. Xaml** et remplacez le contenu par le code suivant.

```

<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:STESample="clr-namespace:STESample;assembly=STESample.Entities"
    mc:Ignorable="d" x:Class="STESample.WPFTest.MainWindow"
    Title="MainWindow" Height="350" Width="525" Loaded="Window_Loaded">

    <Window.Resources>
        <CollectionViewSource
            x:Key="blogViewSource"
            d:DesignSource="{d:DesignInstance {x:Type STESample:Blog}, CreateList=True}"/>
        <CollectionViewSource
            x:Key="blogPostsViewSource"
            Source="{Binding Posts, Source={StaticResource blogViewSource}}"/>
    </Window.Resources>

    <Grid DataContext="{StaticResource blogViewSource}">
        <DataGrid AutoGenerateColumns="False" EnableRowVirtualization="True"
                  ItemsSource="{Binding}" Margin="10,10,10,179">
            <DataGrid.Columns>
                <DataGridTextColumn Binding="{Binding BlogId}" Header="Id" Width="Auto" IsReadOnly="True" />
                <DataGridTextColumn Binding="{Binding Name}" Header="Name" Width="Auto"/>
                <DataGridTextColumn Binding="{Binding Url}" Header="Url" Width="Auto"/>
            </DataGrid.Columns>
        </DataGrid>
        <DataGrid AutoGenerateColumns="False" EnableRowVirtualization="True"
                  ItemsSource="{Binding Source={StaticResource blogPostsViewSource}}"
                  Margin="10,145,10,38">
            <DataGrid.Columns>
                <DataGridTextColumn Binding="{Binding PostId}" Header="Id" Width="Auto" IsReadOnly="True"/>
                <DataGridTextColumn Binding="{Binding Title}" Header="Title" Width="Auto"/>
                <DataGridTextColumn Binding="{Binding Content}" Header="Content" Width="Auto"/>
            </DataGrid.Columns>
        </DataGrid>
        <Button Width="68" Height="23" HorizontalAlignment="Right" VerticalAlignment="Bottom"
               Margin="0,0,10,10" Click="buttonSave_Click">Save</Button>
    </Grid>
</Window>

```

- Ouvrez le code-behind pour MainWindow (**MainWindow.xaml.cs**) et remplacez le contenu par le code suivant :

```

using STESample.WPFTest.BloggingService;
using System.Collections.Generic;
using System.Linq;
using System.Windows;
using System.Windows.Data;

namespace STESample.WPFTest
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            using (var service = new Service1Client())
            {
                // Find the view source for Blogs and populate it with all Blogs (and related Posts)
                // from the Service. The default editing functionality of WPF will allow the objects
                // to be manipulated on the screen.
                var blogsViewSource = (CollectionViewSource)this.FindResource("blogViewSource");
                blogsViewSource.Source = service.GetBlogs().ToList();
            }
        }

        private void buttonSave_Click(object sender, RoutedEventArgs e)
        {
            using (var service = new Service1Client())
            {
                // Get the blogs that are bound to the screen
                var blogsViewSource = (CollectionViewSource)this.FindResource("blogViewSource");
                var blogs = (List<Blog>)blogsViewSource.Source;

                // Save all Blogs and related Posts
                foreach (var blog in blogs)
                {
                    service.UpdateBlog(blog);
                }

                // Re-query for data to get database-generated keys etc.
                blogsViewSource.Source = service.GetBlogs().ToList();
            }
        }
    }
}

```

Vous pouvez à présent exécuter l'application pour la voir en action.

- Cliquez avec le bouton droit sur le projet **STESample. WPFTest** dans **Explorateur de solutions** puis sélectionnez **Déboguer-> démarrer une nouvelle instance**
- Vous pouvez manipuler les données à l'aide de l'écran et les enregistrer par le biais du service à l'aide du bouton **Enregistrer**.

| Blog Id | Name         | Url                   |
|---------|--------------|-----------------------|
| 1       | ADO.NET Blog | blogs.msdn.com/adonet |
| 3       | My Blog      |                       |
|         |              |                       |

| Post Id | Title   | Content                  |
|---------|---------|--------------------------|
| 5       | Welcome | This is my first post... |
|         |         |                          |

# Journalisation et interception des opérations de base de données

23/11/2019 • 24 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

À partir de Entity Framework 6, chaque fois que Entity Framework envoie une commande à la base de données, cette commande peut être interceptée par le code de l'application. C'est le plus souvent utilisé pour la journalisation SQL, mais elle peut également être utilisée pour modifier ou abandonner la commande.

En particulier, EF comprend les éléments suivants :

- Une propriété de journal pour le contexte similaire à `DataContext.log` dans LINQ to SQL
- Mécanisme de personnalisation du contenu et de la mise en forme de la sortie envoyée au Journal
- Blocs de construction de bas niveau pour l'interception donnant plus de contrôle/flexibilité

## Propriété du journal de contexte

La propriété `DbContext.Database.log` peut avoir pour valeur un délégué pour toute méthode qui prend une chaîne. Le plus souvent, il est utilisé avec `n'importe quel` `TextWriter` en le définissant sur la méthode « `Write` » de ce `TextWriter`. Toutes les SQL générées par le contexte actuel seront enregistrées dans ce writer. Par exemple, le code suivant consignera SQL sur la console :

```
using (var context = new BlogContext())
{
    context.Database.Log = Console.WriteLine;

    // Your code here...
}
```

Notez ce `context.Database.log` est défini sur `Console.WriteLine`. C'est tout ce qui est nécessaire pour enregistrer SQL sur la console.

Nous allons ajouter un code simple de requête/insertion/mise à jour pour pouvoir voir une sortie :

```
using (var context = new BlogContext())
{
    context.Database.Log = Console.WriteLine;

    var blog = context.Blogs.First(b => b.Title == "One Unicorn");

    blog.Posts.First().Title = "Green Eggs and Ham";

    blog.Posts.Add(new Post { Title = "I do not like them!" });

    context.SaveChangesAsync().Wait();
}
```

La sortie suivante est générée :

```
SELECT TOP (1)
    [Extent1].[Id] AS [Id],
    [Extent1].[Title] AS [Title]
FROM [dbo].[Blogs] AS [Extent1]
WHERE (N'One Unicorn' = [Extent1].[Title]) AND ([Extent1].[Title] IS NOT NULL)
-- Executing at 10/8/2013 10:55:41 AM -07:00
-- Completed in 4 ms with result: SqlDataReader

SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Title] AS [Title],
    [Extent1].[BlogId] AS [BlogId]
FROM [dbo].[Posts] AS [Extent1]
WHERE [Extent1].[BlogId] = @EntityKeyValue1
-- EntityKeyValue1: '1' (Type = Int32)
-- Executing at 10/8/2013 10:55:41 AM -07:00
-- Completed in 2 ms with result: SqlDataReader

UPDATE [dbo].[Posts]
SET [Title] = @0
WHERE ([Id] = @1)
-- @0: 'Green Eggs and Ham' (Type = String, Size = -1)
-- @1: '1' (Type = Int32)
-- Executing asynchronously at 10/8/2013 10:55:41 AM -07:00
-- Completed in 12 ms with result: 1

INSERT [dbo].[Posts]([Title], [BlogId])
VALUES (@0, @1)
SELECT [Id]
FROM [dbo].[Posts]
WHERE @@ROWCOUNT > 0 AND [Id] = scope_identity()
-- @0: 'I do not like them!' (Type = String, Size = -1)
-- @1: '1' (Type = Int32)
-- Executing asynchronously at 10/8/2013 10:55:41 AM -07:00
-- Completed in 2 ms with result: SqlDataReader
```

(Notez qu'il s'agit de la sortie en supposant que toutes les initialisations de base de données ont déjà eu lieu. Si l'initialisation de la base de données ne s'est pas encore produite, il y aurait beaucoup plus de sortie qui indique que toutes les migrations de travail effectuées dans le cadre de la recherche ou de la création d'une nouvelle base de données.)

## Qu'est-ce qui est enregistré ?

Lorsque la propriété log est définie, tous les éléments suivants sont enregistrés :

- SQL pour tous les différents types de commandes. Exemple :
  - Requêtes, y compris les requêtes LINQ normales, les requêtes eSQL et les requêtes brutes à partir de méthodes telles que `SqlQuery`
  - Insertions, mises à jour et suppressions générées dans le cadre de `SaveChanges`
  - Requêtes de chargement des relations, telles que celles générées par le chargement différé
- Paramètres
- Indique si la commande est exécutée de façon asynchrone
- Horodateur indiquant le début de l'exécution de la commande
- Si la commande a été exécutée avec succès, a échoué en levant une exception ou, pour `Async`, a été annulée
- Indication de la valeur de résultat
- Durée approximative nécessaire à l'exécution de la commande. Notez qu'il s'agit de l'heure à partir de l'envoi de la commande pour récupérer l'objet de résultat. Elle n'inclut pas le temps nécessaire pour lire les résultats.

En examinant l'exemple de sortie ci-dessus, chacune des quatre commandes journalisées est la suivante :

- Requête résultant de l'appel au contexte. Blogs.
  - Notez que la méthode `ToString` pour obtenir le SQL n'aurait pas fonctionné pour cette requête puisque « First » ne fournit pas de `IQueryable` sur lequel `ToString` pourrait être appelé
- Requête résultant du chargement différé du blog. Publié
  - Notez les détails des paramètres pour la valeur de clé pour laquelle le chargement différé se produit
  - Seules les propriétés du paramètre définies à des valeurs non définies par défaut sont journalisées. Par exemple, la propriété `Size` s'affiche uniquement si elle est différente de zéro.
- Deux commandes résultant de `SaveChangesAsync`; une pour la mise à jour pour modifier un titre de publication, l'autre pour une insertion pour ajouter une nouvelle publication
  - Notez les détails des paramètres pour les propriétés FK et title
  - Notez que ces commandes sont exécutées de façon asynchrone

## Journalisation dans différents emplacements

Comme indiqué ci-dessus, la journalisation dans la console est très facile. Il est également facile de se connecter à la mémoire, au fichier, etc. en utilisant différents genres de [TextWriter](#).

Si vous êtes familiarisé avec LINQ to SQL vous pouvez remarquer que dans LINQ to SQL la propriété log est définie sur l'objet `TextWriter` réel (par exemple, `console.out`) alors que dans EF, la propriété log est définie sur une méthode qui accepte une chaîne (par exemple, `, Console.WriteLine` ou `console.out.WriteLine`). Cela consiste à découpler EF de `TextWriter` en acceptant tout délégué pouvant agir en tant que récepteur pour les chaînes. Par exemple, imaginez que vous disposez déjà d'une infrastructure de journalisation et qu'elle définit une méthode de journalisation comme :

```
public class MyLogger
{
    public void Log(string component, string message)
    {
        Console.WriteLine("Component: {0} Message: {1} ", component, message);
    }
}
```

Cela peut être raccordé à la propriété du journal EF comme suit :

```
var logger = new MyLogger();
context.Database.Log = s => logger.Log("EFApp", s);
```

## Journalisation des résultats

L'enregistreur d'événements par défaut enregistre le texte de la commande (SQL), les paramètres et la ligne « en cours d'exécution » avec un horodateur avant que la commande ne soit envoyée à la base de données. Une ligne « terminée » contenant le temps écoulé est journalisée après l'exécution de la commande.

Notez que, pour les commandes Async, la ligne « terminé » n'est journalisée que lorsque la tâche asynchrone se termine, échoue ou est annulée.

La ligne « terminé » contient des informations différentes selon le type de commande et si l'exécution a réussi ou non.

### Exécution réussie

Pour les commandes qui se terminent correctement, la sortie est « terminée en x ms avec le résultat : », suivie d'une indication du résultat. Pour les commandes qui retournent un lecteur de données, l'indication du résultat est

le type de `DbDataReader` retourné. Pour les commandes qui retournent une valeur entière, comme la commande de mise à jour affichée au-dessus du résultat affiché, est cet entier.

## Échec de l'exécution

Pour les commandes qui échouent en levant une exception, la sortie contient le message de l'exception. Par exemple, l'utilisation de `SqlQuery` pour effectuer des requêtes sur une table qui existe entraîne une sortie de journal similaire à ce qui suit :

```
SELECT * from ThisTableIsMissing
-- Executing at 5/13/2013 10:19:05 AM
-- Failed in 1 ms with error: Invalid object name 'ThisTableIsMissing'.
```

## Exécution annulée

Pour les commandes `Async` où la tâche est annulée, le résultat peut être un échec avec une exception, car c'est ce que fait souvent le fournisseur ADO.NET sous-jacent lorsqu'une tentative d'annulation est effectuée. Si cela ne se produit pas et que la tâche est annulée correctement, la sortie se présente comme suit :

```
update Blogs set Title = 'No' where Id = -1
-- Executing asynchronously at 5/13/2013 10:21:10 AM
-- Canceled in 1 ms
```

## Modification du contenu et de la mise en forme du journal

En coulisses, la propriété `Database.Log` utilise un objet `DatabaseLogFormatter`. Cet objet lie efficacement une implémentation de `IDbCommandInterceptor` (voir ci-dessous) à un délégué qui accepte des chaînes et un `DbContext`. Cela signifie que les méthodes sur `DatabaseLogFormatter` sont appelées avant et après l'exécution de commandes par EF. Ces méthodes `DatabaseLogFormatter` rassemblent et mettent en forme la sortie de journal et l'envoient au délégué.

### Personnalisation de `DatabaseLogFormatter`

La modification de ce qui est enregistré et de sa mise en forme peut être obtenue en créant une nouvelle classe qui dérive de `DatabaseLogFormatter` et substitue les méthodes selon le cas. Les méthodes les plus courantes de remplacement sont les suivantes :

- `LogCommand` : remplacez cette valeur pour modifier la façon dont les commandes sont journalisées avant leur exécution. Par défaut, `LogCommand` appelle `LogParameter` pour chaque paramètre ; vous pouvez choisir d'effectuer la même opération dans votre remplacement ou gérer les paramètres différemment.
- `LogResult` : remplacez cette valeur pour modifier le mode de journalisation du résultat de l'exécution d'une commande.
- `LogParameter` : remplacez cette valeur pour modifier la mise en forme et le contenu de la journalisation des paramètres.

Par exemple, supposons que nous voulions enregistrer une seule ligne avant que chaque commande ne soit envoyée à la base de données. Pour ce faire, vous pouvez utiliser deux remplacements :

- Remplacer `LogCommand` pour mettre en forme et écrire la ligne unique de SQL
- Remplacez `LogResult` pour ne rien faire.

Le code doit ressembler à ceci :

```

public class OneLineFormatter : DatabaseLogFormatter
{
    public OneLineFormatter(DbContext context, Action<string> writeAction)
        : base(context, writeAction)
    {
    }

    public override void LogCommand<TResult>(
        DbCommand command, DbCommandInterceptionContext<TResult> interceptionContext)
    {
        Write(string.Format(
            "Context '{0}' is executing command '{1}'{2}",
            Context.GetType().Name,
            command.CommandText.Replace(Environment.NewLine, ""),
            Environment.NewLine));
    }

    public override void LogResult<TResult>(
        DbCommand command, DbCommandInterceptionContext<TResult> interceptionContext)
    {
    }
}

```

Pour consigner la sortie, appelez simplement la méthode Write qui enverra la sortie au délégué d'écriture configuré.

(Notez que ce code effectue une suppression simpliste des sauts de ligne comme un exemple. Elle ne fonctionnera probablement pas correctement pour l'affichage de SQL complexe.)

### Définition de DatabaseLogFormatter

Une fois qu'une nouvelle classe DatabaseLogFormatter a été créée, elle doit être inscrite auprès d'EF. Cette opération s'effectue à l'aide de la configuration basée sur le code. En résumé, cela implique la création d'une nouvelle classe qui dérive de DbConfiguration dans le même assembly que votre classe DbContext, puis l'appel de SetDatabaseLogFormatter dans le constructeur de cette nouvelle classe. Exemple :

```

public class MyDbConfiguration : DbConfiguration
{
    public MyDbConfiguration()
    {
        SetDatabaseLogFormatter(
            (context, writeAction) => new OneLineFormatter(context, writeAction));
    }
}

```

### Utilisation du nouveau DatabaseLogFormatter

Ce nouveau DatabaseLogFormatter sera désormais utilisé à chaque fois que Database.log est défini. Par conséquent, l'exécution du code de la partie 1 génère désormais la sortie suivante :

```

Context 'BlogContext' is executing command 'SELECT TOP (1) [Extent1].[Id] AS [Id], [Extent1].[Title] AS [Title]FROM [dbo].[Blogs] AS [Extent1]WHERE (N'One Unicorn' = [Extent1].[Title]) AND ([Extent1].[Title] IS NOT NULL)'
Context 'BlogContext' is executing command 'SELECT [Extent1].[Id] AS [Id], [Extent1].[Title] AS [Title], [Extent1].[BlogId] AS [BlogId]FROM [dbo].[Posts] AS [Extent1]WHERE [Extent1].[BlogId] = @EntityKeyValue1'
Context 'BlogContext' is executing command 'update [dbo].[Posts]set [Title] = @0where ([Id] = @1)'
Context 'BlogContext' is executing command 'insert [dbo].[Posts]([Title], [BlogId])values (@0, @1)select [Id]from [dbo].[Posts]where @@rowcount > 0 and [Id] = scope_identity()'

```

## Blocs de construction d'interception

Jusqu'à présent, nous avons vu comment utiliser DbContext.Database.Log pour journaliser le SQL généré par EF. Mais ce code est en fait une façade relativement légère sur certains blocs de construction de bas niveau pour une interception plus générale.

## Interfaces d'interception

Le code d'interception est construit autour du concept d'interfaces d'interception. Ces interfaces héritent de IDbInterceptor et définissent les méthodes appelées quand EF effectue une action. L'objectif est d'avoir une interface par type d'objet qui est intercepté. Par exemple, l'interface IDbCommandInterceptor définit des méthodes qui sont appelées avant que EF effectue un appel à ExecuteNonQuery, ExecuteScalar, ExecuteReader et les méthodes associées. De même, l'interface définit les méthodes qui sont appelées lorsque chacune de ces opérations se termine. La classe DatabaseLogFormatter que nous avons examinée ci-dessus implémente cette interface pour enregistrer des commandes.

## Contexte d'interception

En examinant les méthodes définies sur l'une des interfaces d'intercepteur, il est évident que chaque appel reçoit un objet de type DbInterceptionContext ou un type dérivé de celui-ci, tel que DbCommandInterceptionContext<>. Cet objet contient des informations contextuelles sur l'action effectuée par EF. Par exemple, si l'action est effectuée pour le compte d'un DbContext, le DbContext est inclus dans le DbInterceptionContext. De même, pour les commandes qui sont exécutées de façon asynchrone, l'indicateur IsAsync est défini sur DbCommandInterceptionContext.

## Gestion des résultats

La classe DbCommandInterceptionContext<> contient des propriétés appelées result, OriginalResult, exception et OriginalException. Ces propriétés ont la valeur null/zéro pour les appels aux méthodes d'interception qui sont appelées avant l'exécution de l'opération, c'est-à-dire pour le... Exécution des méthodes. Si l'opération est exécutée et réussit, result et OriginalResult sont définis sur le résultat de l'opération. Ces valeurs peuvent ensuite être observées dans les méthodes d'interception qui sont appelées après l'exécution de l'opération, c'est-à-dire sur le... Méthodes exécutées. De même, si l'opération lève, les propriétés exception et OriginalException sont définies.

## Suppression de l'exécution

Si un intercepteur définit la propriété Result avant l'exécution de la commande (dans l'un des... Exécution des méthodes) ensuite, EF ne tente pas d'exécuter la commande, mais utilise simplement le jeu de résultats. En d'autres termes, l'intercepteur peut supprimer l'exécution de la commande, mais l'EF continue comme si la commande avait été exécutée.

Un exemple d'utilisation de cette méthode est le traitement par lot de commandes qui a traditionnellement été fait avec un fournisseur d'encapsulation. L'intercepteur stocke la commande en vue d'une exécution ultérieure sous la forme d'un lot, mais prétend « prétendre » à EF que la commande a été exécutée normalement. Notez qu'il faut plus que cela pour implémenter le traitement par lot, mais il s'agit d'un exemple de la façon dont la modification du résultat de l'interception peut être utilisée.

L'exécution peut également être supprimée en définissant la propriété d'exception dans l'un des... Exécution des méthodes. EF se poursuit alors comme si l'exécution de l'opération avait échoué en levant l'exception donnée. Cela peut, bien sûr, provoquer le blocage de l'application, mais il peut également s'agir d'une exception temporaire ou d'une autre exception gérée par EF. Par exemple, il peut être utilisé dans les environnements de test pour tester le comportement d'une application lorsque l'exécution de la commande échoue.

## Modification du résultat après l'exécution

Si un intercepteur définit la propriété Result après l'exécution de la commande (dans l'un des... Méthodes exécutées) ensuite, EF utilise le résultat modifié au lieu du résultat qui a été réellement retourné par l'opération. De même, si un intercepteur définit la propriété d'exception après l'exécution de la commande, EF lèvera l'exception Set comme si l'opération avait levé l'exception.

Un intercepteur peut également définir la propriété d'exception sur null pour indiquer qu'aucune exception ne doit être levée. Cela peut être utile si l'exécution de l'opération a échoué, mais que l'intercepteur souhaite que EF

continue comme si l'opération avait réussi. Cela implique généralement de définir le résultat afin que EF ait une valeur de résultat à utiliser lorsqu'il continue.

#### **OriginalResult et OriginalException**

Une fois que EF a exécuté une opération, il définit les propriétés result et OriginalResult si l'exécution n'a pas échoué ou les propriétés exception et OriginalException si l'exécution a échoué avec une exception.

Les propriétés OriginalResult et OriginalException sont en lecture seule et ne sont définies par EF qu'après l'exécution d'une opération. Ces propriétés ne peuvent pas être définies par des intercepteurs. Cela signifie qu'un intercepteur peut faire la distinction entre une exception ou un résultat qui a été défini par un autre intercepteur, par opposition à l'exception réelle ou au résultat qui s'est produit lors de l'exécution de l'opération.

#### **Inscription des intercepteurs**

Une fois qu'une classe qui implémente une ou plusieurs interfaces d'interception a été créée, elle peut être inscrite auprès d'EF à l'aide de la classe DbInterception. Exemple :

```
DbInterception.Add(new NLogCommandInterceptor());
```

Les intercepteurs peuvent également être inscrits au niveau du domaine d'application à l'aide du mécanisme de configuration basé sur le code DbConfiguration.

#### **Exemple : journalisation dans NLog**

Nous allons rassembler tout cela dans un exemple qui utilise IDbCommandInterceptor et [nlog](#) pour :

- Consigne un avertissement pour toute commande exécutée de façon non asynchrone
- Consigner une erreur pour toute commande qui lève une exception quand elle est exécutée

Voici la classe qui effectue la journalisation, qui doit être enregistrée comme indiqué ci-dessus :

```

public class NLogCommandInterceptor : IDbCommandInterceptor
{
    private static readonly Logger Logger = LogManager.GetCurrentClassLogger();

    public void NonQueryExecuting(
        SqlCommand command, SqlCommandInterceptionContext<int> interceptionContext)
    {
        LogIfNonAsync(command, interceptionContext);
    }

    public void NonQueryExecuted(
        SqlCommand command, SqlCommandInterceptionContext<int> interceptionContext)
    {
        LogIfError(command, interceptionContext);
    }

    public void ReaderExecuting(
        SqlCommand command, SqlCommandInterceptionContext<DbDataReader> interceptionContext)
    {
        LogIfNonAsync(command, interceptionContext);
    }

    public void ReaderExecuted(
        SqlCommand command, SqlCommandInterceptionContext<DbDataReader> interceptionContext)
    {
        LogIfError(command, interceptionContext);
    }

    public void ScalarExecuting(
        SqlCommand command, SqlCommandInterceptionContext<object> interceptionContext)
    {
        LogIfNonAsync(command, interceptionContext);
    }

    public void ScalarExecuted(
        SqlCommand command, SqlCommandInterceptionContext<object> interceptionContext)
    {
        LogIfError(command, interceptionContext);
    }

    private void LogIfNonAsync<TResult>(
        SqlCommand command, SqlCommandInterceptionContext<TResult> interceptionContext)
    {
        if (!interceptionContext.IsAsync)
        {
            Logger.Warn("Non-async command used: {0}", command.CommandText);
        }
    }

    private void LogIfError<TResult>(
        SqlCommand command, SqlCommandInterceptionContext<TResult> interceptionContext)
    {
        if (interceptionContext.Exception != null)
        {
            Logger.Error("Command {0} failed with exception {1}",
                command.CommandText, interceptionContext.Exception);
        }
    }
}

```

Notez que ce code utilise le contexte d'interception pour découvrir quand une commande est exécutée de façon non asynchrone et pour découvrir quand une erreur s'est produite lors de l'exécution d'une commande.

# Considérations relatives aux performances pour EF 4, 5 et 6

23/11/2019 • 138 minutes to read

Par David Obando, Eric Dettinger et autres

Publication : 2012 avril

Dernière mise à jour : mai 2014

## 1. Introduction

Les infrastructures de mappage objet-relationnel sont un moyen pratique de fournir une abstraction pour l'accès aux données dans une application orientée objet. Pour les applications .NET, l'O/RM recommandé de Microsoft est Entity Framework. Malgré toute abstraction, les performances peuvent devenir un problème.

Ce livre blanc a été rédigé pour illustrer les considérations relatives aux performances lors du développement d'applications à l'aide de Entity Framework, afin de donner aux développeurs une idée des algorithmes internes Entity Framework qui peuvent affecter les performances et de fournir des conseils pour l'investigation et amélioration des performances dans leurs applications qui utilisent Entity Framework. Il existe un certain nombre de bonnes rubriques sur les performances déjà disponibles sur le Web et nous avons également essayé de pointer vers ces ressources dans la mesure du possible.

Les performances sont un sujet délicat. Ce livre blanc est destiné à vous aider à prendre des décisions relatives aux performances de vos applications qui utilisent Entity Framework. Nous avons inclus des mesures de test pour illustrer les performances, mais ces mesures ne sont pas conçues comme indicateurs absolus des performances que vous verrez dans votre application.

Pour des raisons pratiques, ce document suppose Entity Framework 4 s'exécute sous .NET 4,0 et Entity Framework 5 et 6 sont exécutés sous .NET 4,5. La plupart des améliorations de performances apportées à Entity Framework 5 résident dans les composants de base fournis avec .NET 4,5.

Entity Framework 6 est une version hors bande et ne dépend pas des composants Entity Framework fournis avec .NET. Entity Framework 6 fonctionnent à la fois sur .NET 4,0 et .NET 4,5, et peuvent offrir un gain de performances considérable à ceux qui n'ont pas été mis à niveau à partir de .NET 4,0, mais qui veulent les derniers Entity Framework bits dans leur application. Lorsque ce document mentionne Entity Framework 6, il fait référence à la dernière version disponible au moment de la rédaction de cet article : version 6.1.0.

## 2. exécution des requêtes à froid et à chaud

La première fois qu'une requête est effectuée sur un modèle donné, le Entity Framework fait beaucoup de travail en arrière-plan pour charger et valider le modèle. Nous faisons souvent référence à cette première requête sous la forme d'une requête « à froid ». Les requêtes supplémentaires sur un modèle déjà chargé sont appelées requêtes « chaudes » et sont beaucoup plus rapides.

Examinons de plus haut niveau le temps consacré à l'exécution d'une requête à l'aide de Entity Framework, et voyons où les choses s'améliorent dans Entity Framework 6.

### Exécution de la première requête-requête à froid

| ÉCRITURES UTILISATEUR DU CODE  | ACTION                               | IMPACT SUR LES PERFORMANCES DE EF4   | IMPACT SUR LES PERFORMANCES DE EF5   | IMPACT SUR LES PERFORMANCES DE EF6   |
|--|--------------------------------------|--|--|--|
| <pre>using(var db = new MyContext()) {}</pre>                          | Création de contexte                 | Moyenne  | Moyenne  | Basse  |
| <pre>var q1 = from c in db.Customers where c.Id == id1 select c;</pre> | Création d'une expression de requête | Basse  | Basse  | Basse  |
| <pre>var c1 = q1.First();</pre>  | Exécution de requêtes LINQ           | <ul style="list-style-type: none"> <li>-Chargement des métadonnées : élevé mais mis en cache</li> <li>-Génération de vues : potentiellement très élevée mais mise en cache</li> <li>-Évaluation des paramètres : moyenne</li> <li>-Traduction de requête : moyenne</li> <li>-Génération de matérialisé : moyenne mais mise en cache</li> <li>-Exécution de la requête de base de données : potentiellement élevé</li> <li>+ Connection. Open</li> <li>+ Command.ExecuteReader</li> <li>+ DataReader.Read</li> <li>Matérialisation d'objets : moyenne</li> <li>-Recherche d'identité : moyenne</li> </ul> | <ul style="list-style-type: none"> <li>-Chargement des métadonnées : élevé mais mis en cache</li> <li>-Génération de vues : potentiellement très élevée mais mise en cache</li> <li>-Évaluation des paramètres : faible</li> <li>-Traduction de requête : moyenne mais mise en cache</li> <li>-Génération de matérialisé : moyenne mais mise en cache</li> <li>-Exécution des requêtes de base de données :</li> <li>potentiellement élevé (meilleures requêtes dans certaines situations)</li> <li>+ Connection. Open</li> <li>+ Command.ExecuteReader</li> <li>+ DataReader.Read</li> <li>Matérialisation d'objets : moyenne (plus rapide que EF5)</li> <li>-Recherche d'identité : moyenne</li> </ul> | <ul style="list-style-type: none"> <li>-Chargement des métadonnées : élevé mais mis en cache</li> <li>-Génération de vues : moyenne mais mise en cache</li> <li>-Évaluation des paramètres : faible</li> <li>-Traduction de requête : moyenne mais mise en cache</li> <li>-Génération de matérialisé : moyenne mais mise en cache</li> <li>-Exécution des requêtes de base de données :</li> <li>potentiellement élevé (meilleures requêtes dans certaines situations)</li> <li>+ Connection. Open</li> <li>+ Command.ExecuteReader</li> <li>+ DataReader.Read</li> <li>Matérialisation d'objets : moyenne (plus rapide que EF5)</li> <li>-Recherche d'identité : moyenne</li> </ul> |
| }  | Connection. Close                    | Basse  | Basse  | Basse  |

### Exécution de la deuxième requête-requête à chaud

| ÉCRITURES UTILISATEUR DU CODE                 | ACTION               | IMPACT SUR LES PERFORMANCES DE EF4 | IMPACT SUR LES PERFORMANCES DE EF5 | IMPACT SUR LES PERFORMANCES DE EF6 |
|---|----------------------|------------------------------------|------------------------------------|------------------------------------|
| <pre>using(var db = new MyContext()) {}</pre> | Création de contexte | Moyenne                            | Moyenne                            | Basse                              |

| ÉCRITURES UTILISATEUR DU CODE  | ACTION                               | IMPACT SUR LES PERFORMANCES DE EF4   | IMPACT SUR LES PERFORMANCES DE EF5   | IMPACT SUR LES PERFORMANCES DE EF6   |
|--|--------------------------------------|--|--|--|
| <pre>var q1 =<br/>from c in<br/>db.Customers<br/>where c.Id == id1<br/>select c;</pre> | Création d'une expression de requête | Basse  | Basse  | Basse  |
| <pre>var c1 =<br/>q1.First();</pre>  | Exécution de requêtes LINQ           | <ul style="list-style-type: none"> <li>-Recherche du chargement des métadonnées : haute, mais mise en cache faible</li> <li>-Afficher la recherche de génération : potentiellement très élevé mais mis en cache faible</li> <li>-Évaluation des paramètres : moyenne</li> <li>-Recherche de traduction de requête : moyenne</li> <li>-Recherche de la génération de matérialise : moyenne mais mise en cache faible</li> <li>-Exécution de la requête de base de données : potentiellement élevé + Connection. Open + Command.ExecuteReader + DataReader.Read Matérialisation d'objets : moyenne</li> <li>-Recherche d'identité : moyenne</li> </ul> | <ul style="list-style-type: none"> <li>-Recherche du chargement des métadonnées : haute, mais mise en cache faible</li> <li>-Afficher la recherche de génération : potentiellement très élevé mais mis en cache faible</li> <li>-Évaluation des paramètres : faible</li> <li>-Recherche de traduction de requête : moyenne mais mise en cache faible</li> <li>-Recherche de la génération de matérialise : moyenne mais mise en cache faible</li> <li>-Exécution des requêtes de base de données : potentiellement élevé (meilleures requêtes dans certaines situations) + Connection. Open + Command.ExecuteReader + DataReader.Read Matérialisation d'objets : moyenne</li> <li>-Recherche d'identité : moyenne</li> </ul> | <ul style="list-style-type: none"> <li>-Recherche du chargement des métadonnées : haute, mais mise en cache faible</li> <li>-Afficher la recherche de génération : moyenne mais mise en cache faible</li> <li>-Évaluation des paramètres : faible</li> <li>-Recherche de traduction de requête : moyenne mais mise en cache faible</li> <li>-Recherche de la génération de matérialise : moyenne mais mise en cache faible</li> <li>-Exécution des requêtes de base de données : potentiellement élevé (meilleures requêtes dans certaines situations) + Connection. Open + Command.ExecuteReader + DataReader.Read Matérialisation d'objets : moyenne (plus rapide que EF5)</li> <li>-Recherche d'identité : moyenne</li> </ul> |
| }  | Connection. Close                    | Basse  | Basse  | Basse  |

Il existe plusieurs façons de réduire le coût de performance des requêtes à froid et à chaud, et nous examinerons ces éléments dans la section suivante. Plus précisément, nous examinerons la réduction du coût du chargement de modèle dans les requêtes à froid en utilisant des vues prégénérées, ce qui devrait aider à atténuer les problèmes de performances rencontrés lors de la génération de vues. Pour les requêtes à chaud, nous allons aborder la mise en cache des plans de requête, les requêtes de suivi et les différentes options d'exécution de requête.

## 2.1 qu'est-ce que la génération de vues ?

Pour comprendre la génération de vues, nous devons d'abord comprendre ce que sont les « vues de mappage ». Les vues de mappage sont des représentations exécutables des transformations spécifiées dans le mappage pour chaque jeu d'entités et Association. En interne, ces vues de mappage prennent la forme de CQTs (arborescences de

requêtes canoniques). Il existe deux types de vues de mappage :

- Vues des requêtes : elles représentent la transformation nécessaire pour passer du schéma de base de données au modèle conceptuel.
- Vues de mise à jour : elles représentent la transformation nécessaire pour passer du modèle conceptuel au schéma de base de données.

Gardez à l'esprit que le modèle conceptuel peut différer du schéma de base de données de différentes façons. Par exemple, une seule table peut être utilisée pour stocker les données de deux types d'entités différents. Les mappages d'héritage et non trivial jouent un rôle dans la complexité des vues de mappage.

Le processus de calcul de ces vues en fonction de la spécification du mappage est ce que nous appelons la génération de vues. La génération de vues peut être exécutée dynamiquement lorsqu'un modèle est chargé, ou au moment de la génération, à l'aide de « vues pré générées »; ces dernières sont sérialisées sous la forme d'instructions Entity SQL dans un fichier C# ou VB.

Lorsque des vues sont générées, elles sont également validées. Du point de vue des performances, la grande majorité du coût de la génération de vues est la validation des vues, qui garantit que les connexions entre les entités ont un sens et qu'elles disposent de la cardinalité correcte pour toutes les opérations prises en charge.

Lorsqu'une requête sur un jeu d'entités est exécutée, la requête est associée à la vue de requête correspondante, et le résultat de cette composition est exécuté via le compilateur de plan pour créer la représentation de la requête que le magasin de stockage peut comprendre. Par SQL Server, le résultat final de cette compilation sera une instruction T-SQL SELECT. La première fois qu'une mise à jour est effectuée sur un jeu d'entités, la vue de mise à jour est exécutée à l'aide d'un processus similaire pour la transformer en instructions DML pour la base de données cible.

## 2.2 facteurs qui affectent les performances de la génération de vues

Les performances de l'étape de génération d'affichage dépendent non seulement de la taille de votre modèle, mais également de la manière dont le modèle est interconnecté. Si deux entités sont connectées via une chaîne d'héritage ou une association, elles sont dites connectées. De même, si deux tables sont connectées via une clé étrangère, elles sont connectées. À mesure que le nombre d'entités et de tables connectées dans vos schémas augmente, le coût de la génération d'affichage augmente.

L'algorithme que nous utilisons pour générer et valider des vues est exponentiel dans le pire des cas, bien que nous utilisions des optimisations pour améliorer cela. Les facteurs les plus importants qui semblent nuire aux performances sont les suivants :

- Taille du modèle, en faisant référence au nombre d'entités et à la quantité d'associations entre ces entités.
- Complexité du modèle, en particulier l'héritage impliquant un grand nombre de types.
- À l'aide d'associations indépendantes, plutôt que d'associations de clé étrangère.

Pour les petits modèles simples, le coût peut être suffisamment petit pour ne pas se soucier de l'utilisation de vues pré générées. À mesure que la taille et la complexité du modèle augmentent, plusieurs options sont disponibles pour réduire le coût de la génération et de la validation de la vue.

## 2.3 utilisation de vues pré générées pour réduire le temps de chargement du modèle

Pour plus d'informations sur l'utilisation des affichages pré générés sur Entity Framework 6, consultez [affichages de mappage pré générés](#)

### 2.3.1 affichages pré générés à l'aide de Entity Framework Power Tools Community Edition

Vous pouvez utiliser l'[Entity Framework 6 Power Tools Community Edition](#) pour générer des vues de modèles EDMX et code First en cliquant avec le bouton droit sur le fichier de classe de modèle et en utilisant le menu Entity Framework pour sélectionner « générer des vues ». L'édition Community d'Entity Framework Power Tools ne fonctionne que sur les contextes dérivés de DbContext.

### 2.3.2 Comment utiliser des vues pré générées avec un modèle créé par EDMGen

EDMGen est un utilitaire fourni avec .NET et fonctionne avec Entity Framework 4 et 5, mais pas avec Entity Framework 6. EDMGen vous permet de générer un fichier de modèle, la couche objet et les vues à partir de la ligne de commande. L'une des sorties est un fichier de vues dans le langage de votre choix, VB ou C#. Il s'agit d'un fichier de code contenant Entity SQL extraits de code pour chaque jeu d'entités. Pour activer les vues pré-générées, il vous suffit d'inclure le fichier dans votre projet.

Si vous apportez manuellement des modifications aux fichiers de schéma pour le modèle, vous devrez générer à nouveau le fichier de vues. Pour ce faire, exécutez EDMGen avec l'indicateur **/mode : ViewGeneration**.

### 2.3.3 comment utiliser des vues pré-générées avec un fichier EDMX

Vous pouvez également utiliser EDMGen pour générer des vues pour un fichier EDMX : la rubrique MSDN précédemment référencée explique comment ajouter un événement pré-build pour effectuer cette opération, mais cela est complexe et dans certains cas, il n'est pas possible. Il est généralement plus facile d'utiliser un modèle T4 pour générer les vues lorsque votre modèle se trouve dans un fichier edmx.

Blog de l'équipe ADO.NET a une requête post qui explique comment utiliser un modèle T4 pour la génération de vues ( <http://blogs.msdn.com/b/adonet/archive/2008/06/20/how-to-use-a-t4-template-for-view-generation.aspx> ). Ce billet comprend un modèle qui peut être téléchargé et ajouté à votre projet. Le modèle a été écrit pour la première version de Entity Framework. Il n'est donc pas garanti qu'il fonctionne avec les dernières versions de Entity Framework. Toutefois, vous pouvez télécharger un ensemble plus à jour de modèles de génération d'affichage pour Entity Framework 4 et 5 from la Galerie Visual Studio :

- VB.NET : <http://visualstudiogallery.msdn.microsoft.com/118b44f2-1b91-4de2-a584-7a680418941d>
- C# : <http://visualstudiogallery.msdn.microsoft.com/ae7730ce-ddab-470f-8456-1b313cd2c44d>

Si vous utilisez Entity Framework 6 vous pouvez obtenir l'affichage de modèles de génération T4 à partir de la galerie Visual Studio à <http://visualstudiogallery.msdn.microsoft.com/18a7db90-6705-4d19-9dd1-0a6c23d0751f>.

## 2.4 réduction du coût de la génération de vues

L'utilisation de vues pré-générées déplace le coût de la génération de vues du chargement du modèle (au moment de l'exécution) au moment de la conception. Bien que cela améliore les performances de démarrage lors de l'exécution, vous serez toujours confronté à la difficulté de la génération d'affichages pendant le développement. Il existe plusieurs astuces supplémentaires qui peuvent aider à réduire le coût de la génération de vues, à la fois au moment de la compilation et au moment de l'exécution.

### 2.4.1 utilisation d'associations de clé étrangère pour réduire le coût de la génération de vues

Nous avons vu un certain nombre de cas dans lesquels le basculement des associations de modèles indépendants vers des associations de clé étrangère a considérablement amélioré le temps passé dans la génération de vues.

Pour illustrer cette amélioration, nous avons généré deux versions du modèle Navision à l'aide d'EDMGen.

*Remarque : consultez l'annexe C pour obtenir une description du modèle Navision.* Le modèle Navision est intéressant pour cet exercice en raison de son très grand nombre d'entités et de relations entre eux.

Une version de ce modèle très volumineux a été générée avec des associations de clés étrangères et l'autre a été générée avec des associations indépendantes. Nous avons ensuite dépassé le temps nécessaire à la génération des vues pour chaque modèle. Entity Framework 5 a utilisé la méthode GenerateViews () de la classe EntityViewGenerator pour générer les vues, tandis que le test Entity Framework 6 a utilisé la méthode GenerateViews () à partir de la classe StorageMappingItemCollection. Ceci en raison de la restructuration du code qui s'est produite dans le Entity Framework 6 code base.

À l'aide de Entity Framework 5, la génération de vues pour le modèle avec des clés étrangères a duré 65 minutes sur un ordinateur de laboratoire. Il s'agit d'un nombre inconnu de la durée nécessaire pour générer les vues du modèle qui utilisaient des associations indépendantes. Nous avons laissé le test en cours d'exécution pendant plus d'un mois avant le redémarrage de l'ordinateur dans notre laboratoire pour installer les mises à jour mensuelles.

À l'aide de Entity Framework 6, la génération de la vue pour le modèle avec des clés étrangères a pris 28 secondes

dans le même ordinateur Lab. La génération d'affichage pour le modèle qui utilise des associations indépendantes a duré 58 secondes. Les améliorations apportées à Entity Framework 6 sur son code de génération d'affichage signifient que de nombreux projets n'ont pas besoin de vues pré générées pour obtenir des temps de démarrage plus rapides.

Il est important de noter que les affichages pré générés dans Entity Framework 4 et 5 peuvent être réalisés avec EDMGen ou les outils d'Entity Framework Power Tools. Pour la génération de Entity Framework 6, la génération peut s'effectuer via les outils Entity Framework Power Tools ou par programme, comme décrit dans les [vues de mappage pré générées](#).

#### 2.4.1.1 comment utiliser des clés étrangères plutôt que des associations indépendantes

Lorsque vous utilisez EDMGen ou le Entity Designer dans Visual Studio, vous recevez clés étrangères par défaut, et il n'accepte qu'un seul indicateur de case à cocher ou de ligne de commande pour basculer entre clés étrangères et IAs.

Si vous avez un modèle de Code First volumineux, l'utilisation d'associations indépendantes aura le même effet sur la génération de la vue. Vous pouvez éviter cet impact en incluant des propriétés de clé étrangère sur les classes pour vos objets dépendants, bien que certains développeurs considèrent cela comme un modèle d'objet polluant. Vous trouverez plus d'informations sur ce sujet dans <<http://blog.oneunicorn.com/2011/12/11/what's-the-deal-with-mapping-foreign-keys-using-the-entity-framework/>>.

| QUAND VOUS UTILISEZ  | FAITES CELA  |
|----------------------|--|
| Concepteur d'entités | Après avoir ajouté une association entre deux entités, vérifiez que vous disposez d'une contrainte référentielle. Les contraintes référentielles indiquent Entity Framework utiliser des clés étrangères plutôt que des associations indépendantes. Pour plus d'informations, consultez < <a href="http://blogs.msdn.com/b/efdesign/archive/2009/03/16/foreign-keys-in-the-entity-framework.aspx">http://blogs.msdn.com/b/efdesign/archive/2009/03/16/foreign-keys-in-the-entity-framework.aspx</a> >. |
| EDMGen               | Lorsque vous utilisez EDMGen pour générer vos fichiers à partir de la base de données, vos clés étrangères sont respectées et ajoutées au modèle en tant que tel. Pour plus d'informations sur les différentes options exposées par EDMGen visitez <a href="http://msdn.microsoft.com/library/bb387165.aspx">http://msdn.microsoft.com/library/bb387165.aspx</a> .   |
| Code First           | Pour plus d'informations sur l'inclusion de propriétés de clé étrangère sur les objets dépendants lors de l'utilisation de Code First, consultez la section « Convention de relation » de la rubrique <a href="#">conventions de code First</a> .  |

#### 2.4.2 déplacement de votre modèle vers un assembly distinct

Lorsque votre modèle est inclus directement dans le projet de votre application et que vous générez des vues par le biais d'un modèle d'événement pré-build ou T4, la génération et la validation de la vue sont effectuées chaque fois que le projet est régénéré, même si le modèle n'a pas été modifié. Si vous déplacez le modèle vers un assembly distinct et le référencez à partir du projet de votre application, vous pouvez apporter d'autres modifications à votre application sans avoir à régénérer le projet contenant le modèle.

*Remarque :* lors du déplacement de votre modèle vers des assemblies distincts, n'oubliez pas de copier les chaînes de connexion du modèle dans le fichier de configuration de l'application du projet client.

#### 2.4.3 désactiver la validation d'un modèle basé sur edmx

Les modèles EDMX sont validés au moment de la compilation, même si le modèle n'est pas modifié. Si votre modèle a déjà été validé, vous pouvez supprimer la validation au moment de la compilation en affectant à la propriété « valider à la génération » la valeur false dans la fenêtre Propriétés. Lorsque vous modifiez votre mappage ou modèle, vous pouvez réactiver temporairement la validation pour vérifier vos modifications.

Notez que les améliorations des performances ont été apportées au Entity Framework Designer pour Entity Framework 6, et que le coût de la « validation sur Build » est bien plus faible que dans les versions précédentes du concepteur.

## 3 Caching dans le Entity Framework

Entity Framework présente les formes de mise en cache intégrées suivantes :

1. Mise en cache d'objets : le ObjectStateManager intégré à une instance ObjectContext garde le suivi en mémoire des objets qui ont été récupérés à l'aide de cette instance. Cela est également appelé cache de premier niveau.
2. Mise en cache du plan de requête-réutilisation de la commande de stockage générée lorsqu'une requête est exécutée plusieurs fois.
3. Mise en cache des métadonnées : partage des métadonnées pour un modèle sur différentes connexions au même modèle.

Outre les caches fournis par EF, un type spécial de fournisseur de données ADO.NET connu sous le nom de fournisseur d'encapsulation peut également être utilisé pour étendre Entity Framework avec un cache pour les résultats récupérés à partir de la base de données, également appelée mise en cache de second niveau.

### Mise en cache d'objets 3,1

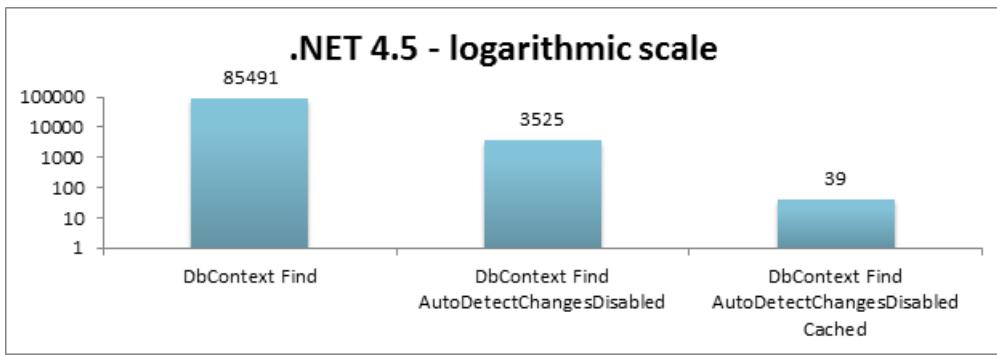
Par défaut, lorsqu'une entité est retournée dans les résultats d'une requête, juste avant que EF ne le matérialise, ObjectContext vérifie si une entité avec la même clé a déjà été chargée dans son ObjectStateManager. Si une entité avec les mêmes clés est déjà présente, EF l'inclut dans les résultats de la requête. Bien que EF émette toujours la requête sur la base de données, ce comportement peut contourner la majeure partie du coût de la matérialisation de l'entité plusieurs fois.

#### 3.1.1 obtention d'entités à partir du cache d'objets à l'aide de DbContext Find

Contrairement à une requête normale, la méthode Find dans DbSet (les API incluses pour la première fois dans EF 4,1) effectue une recherche dans la mémoire avant même d'émettre la requête sur la base de données. Il est important de noter que deux instances ObjectContext auront deux instances ObjectStateManager différentes, ce qui signifie qu'elles ont des caches d'objets distincts.

Find utilise la valeur de clé primaire pour tenter de trouver une entité suivie par le contexte. Si l'entité n'est pas dans le contexte, une requête est exécutée et évaluée par rapport à la base de données, et la valeur null est retournée si l'entité est introuvable dans le contexte ou dans la base de données. Notez que Find retourne également les entités qui ont été ajoutées au contexte mais qui n'ont pas encore été enregistrées dans la base de données.

Vous pouvez prendre en compte les performances lors de l'utilisation de la recherche. Par défaut, les appels à cette méthode déclenchent une validation du cache d'objets afin de détecter les modifications qui sont toujours en attente de validation dans la base de données. Ce processus peut être très onéreux s'il existe un très grand nombre d'objets dans le cache d'objets ou dans un graphique d'objets volumineux qui est ajouté au cache d'objets, mais il peut également être désactivé. Dans certains cas, vous pouvez percevoir un ordre de grandeur de différence lors de l'appel de la méthode Find lorsque vous désactivez la détection automatique des modifications. Toutefois, un deuxième ordre de grandeur est perçu lorsque l'objet est réellement dans le cache par rapport au moment où l'objet doit être récupéré de la base de données. Voici un exemple de graphique avec des mesures effectuées à l'aide de quelques tests, exprimés en millisecondes, avec une charge de 5000 entités :



Exemple de recherche avec détection automatique des modifications désactivées :

```
context.Configuration.AutoDetectChangesEnabled = false;
var product = context.Products.Find(productId);
context.Configuration.AutoDetectChangesEnabled = true;
...
```

Les éléments à prendre en compte lors de l'utilisation de la méthode Find sont les suivants :

- Si l'objet n'est pas dans le cache, les avantages de Find sont inversés, mais la syntaxe est toujours plus simple qu'une requête par clé.
- Si la détection automatique des modifications est activée, le coût de la méthode Find peut augmenter d'un ordre de grandeur, voire plus, en fonction de la complexité de votre modèle et de la quantité d'entités dans votre cache d'objets.

En outre, n'oubliez pas que la recherche ne retourne que l'entité que vous recherchez et qu'elle ne charge pas automatiquement les entités associées si elles ne se trouvent pas déjà dans le cache d'objets. Si vous devez récupérer les entités associées, vous pouvez utiliser une requête par clé avec un chargement hâtif. Pour plus d'informations, consultez **8.1 chargement différé et chargement hâtif**.

### **3.1.2 problèmes de performances lorsque le cache d'objets a de nombreuses entités**

Le cache d'objets permet d'augmenter la réactivité globale de Entity Framework. Toutefois, lorsque le cache d'objets a une très grande quantité d'entités chargées, il peut affecter certaines opérations, telles que ajouter, supprimer, Rechercher, entrée, SaveChanges et bien plus encore. En particulier, les opérations qui déclenchent un appel à DetectChanges seront affectées par des caches d'objets très volumineux. DetectChanges synchronise le graphique d'objets avec le gestionnaire d'état d'objet et ses performances sont déterminées directement par la taille du graphique d'objets. Pour plus d'informations sur DetectChanges, consultez [suivi des modifications dans les entités POCO](#).

Lorsque vous utilisez Entity Framework 6, les développeurs peuvent appeler AddRange et RemoveRange directement sur un DbSet, au lieu d'effectuer une itération sur une collection et d'appeler Add once par instance. L'avantage de l'utilisation des méthodes de plage est que le coût de DetectChanges n'est payé qu'une seule fois pour l'ensemble des entités, par opposition à une seule fois pour chaque entité ajoutée.

### **3.2 mise en cache du plan de requête**

La première fois qu'une requête est exécutée, elle passe par le compilateur de plan interne pour traduire la requête conceptuelle dans la commande de stockage (par exemple, T-SQL qui est exécutée lors de l'exécution sur SQL Server). Si la mise en cache du plan de requête est activée, la prochaine fois que la requête est exécutée, la commande de stockage est récupérée directement à partir du cache du plan de requête pour exécution, en ignorant le compilateur du plan.

Le cache du plan de requête est partagé entre les instances ObjectContext au sein du même AppDomain. Vous n'avez pas besoin de conserver une instance ObjectContext pour tirer parti de la mise en cache du plan de requête.

#### **3.2.1 quelques remarques sur la mise en cache du plan de requête**

- Le cache du plan de requête est partagé pour tous les types de requêtes : les objets Entity SQL, LINQ to

Entities et CompiledQuery.

- Par défaut, la mise en cache du plan de requête est activée pour les requêtes Entity SQL, qu'elles soient exécutées via un EntityCommand ou via un ObjectQuery. Elle est également activée par défaut pour les requêtes LINQ to Entities dans Entity Framework sur .NET 4.5 et dans Entity Framework 6
  - La mise en cache du plan de requête peut être désactivée en affectant à la propriété EnablePlanCaching (sur EntityCommand ou ObjectQuery) la valeur false. Exemple :

```
var query = from customer in context.Customer
            where customer.CustomerId == id
            select new
            {
                customer.CustomerId,
                customer.Name
            };
ObjectQuery oQuery = query as ObjectQuery;
oQuery.EnablePlanCaching = false;
```

- Pour les requêtes paramétrables, la modification de la valeur du paramètre atteint toujours la requête mise en cache. Toutefois, la modification des facettes d'un paramètre (par exemple, la taille, la précision ou l'échelle) atteint une entrée différente dans le cache.
- Lors de l'utilisation de Entity SQL, la chaîne de requête fait partie de la clé. La modification de la requête peut entraîner des entrées de cache différentes, même si les requêtes sont fonctionnellement équivalentes. Cela comprend les modifications apportées à la casse ou à l'espace blanc.
- Lors de l'utilisation de LINQ, la requête est traitée pour générer une partie de la clé. La modification de l'expression LINQ générera donc une clé différente.
- D'autres limitations techniques peuvent s'appliquer ; Pour plus d'informations, consultez requêtes autocompilées.

### 3.2.2-algorithme d'éviction du cache

Comprendre le fonctionnement de l'algorithme interne vous aide à déterminer quand activer ou désactiver la mise en cache du plan de requête. L'algorithme de nettoyage est le suivant :

1. Une fois que le cache contient un nombre défini d'entrées (800), nous commençons un minuteur qui balaye régulièrement le cache (une fois par minute).
2. Pendant les nettoyages du cache, les entrées sont supprimées du cache sur une base LFRU (le moins fréquemment utilisé récemment). Cet algorithme prend en compte le nombre d'accès et l'âge pour déterminer quelles entrées sont éjectées.
3. À la fin de chaque balayage du cache, le cache contient 800 entrées.

Toutes les entrées de cache sont traitées de manière égale lors de la détermination des entrées à supprimer. Cela signifie que la commande de stockage pour un CompiledQuery a le même risque d'éviction que la commande de stockage pour une requête de Entity SQL.

Notez que le minuteur d'éviction du cache est lancé lorsqu'il y a 800 entités dans le cache, mais que le cache n'est balayé que 60 secondes après le démarrage de ce minuteur. Cela signifie que jusqu'à 60 secondes, votre cache peut croître pour être assez volumineux.

### 3.2.3 métriques de test illustrant les performances de mise en cache du plan de requête

Pour illustrer l'effet de la mise en cache du plan de requête sur les performances de votre application, nous avons effectué un test sur lequel nous avons exécuté un certain nombre de requêtes Entity SQL sur le modèle Navision. Reportez-vous à l'annexe pour obtenir une description du modèle Navision et des types de requêtes qui ont été exécutés. Dans ce test, nous allons d'abord itérer au sein de la liste des requêtes et les exécuter une fois pour les ajouter au cache (si la mise en cache est activée). Cette étape n'est pas terminée. Ensuite, nous mettons en veille le thread principal pendant plus de 60 secondes pour permettre le balayage du cache. Enfin, nous parcourons la liste une deuxième fois pour exécuter les requêtes mises en cache. En outre, le cache du plan de SQL Server est vidé

avant l'exécution de chaque ensemble de requêtes, de sorte que les fois que nous obtenons précisément l'avantage donné par le cache du plan de requête.

#### 3.2.3.1 Résultats des tests

| TEST  | EF5 AUCUN CACHE | EF5 MIS EN CACHE | EF6 AUCUN CACHE | EF6 MIS EN CACHE |
|---|-----------------|------------------|-----------------|------------------|
| Énumération de toutes les requêtes 18723  | 124             | 125,4            | 124,3           | 125,3            |
| Éviter le balayage (uniquement les premières requêtes 800, quelle que soit la complexité) | 41,7            | 5,5              | 40,5            | 5,4              |
| Uniquement les requêtes AggregatingSubtotals (178 au total-qui évite le balayage)         | 39,5            | 4,5              | 38,1            | 4,6              |

Toutes les fois en secondes.

Moral : lors de l'exécution d'un grand nombre de requêtes distinctes (par exemple, des requêtes créées dynamiquement), la mise en cache n'est pas utile et le vidage résultant du cache peut conserver les requêtes qui tireraient le meilleur parti de la mise en cache de plan pour l'utiliser en fait.

Les requêtes AggregatingSubtotals sont les plus complexes des requêtes que nous avons testées avec. Comme prévu, plus la requête est complexe, plus l'avantage que vous verrez dans la mise en cache du plan de requête est avantageux.

Étant donné qu'un CompiledQuery est en réalité une requête LINQ avec son plan mis en cache, la comparaison entre un CompiledQuery et l'équivalent Entity SQL requête doit avoir des résultats similaires. En fait, si une application possède un grand nombre de requêtes Entity SQL dynamiques, le remplissage du cache avec des requêtes entraînera également la « décompilation » de CompiledQueries lorsqu'elles seront vidées du cache. Dans ce scénario, les performances peuvent être améliorées en désactivant la mise en cache sur les requêtes dynamiques afin de hiérarchiser les CompiledQueries. Mieux encore, bien sûr, il serait de réécrire l'application pour utiliser des requêtes paramétrables au lieu de requêtes dynamiques.

### 3.3 utilisation de CompiledQuery pour améliorer les performances avec des requêtes LINQ

Nos tests indiquent que l'utilisation de CompiledQuery peut apporter un avantage de 7% sur les requêtes LINQ autocompilées. Cela signifie que vous allez consacrer 7% moins de temps à exécuter du code à partir de la pile Entity Framework ; Cela ne signifie pas que votre application sera 7% plus rapide. En règle générale, le coût de l'écriture et de la gestion des objets CompiledQuery dans EF 5,0 peut ne pas poser de problème par rapport aux avantages. Votre kilométrage peut varier. Vous devez donc faire preuve de cette possibilité si votre projet nécessite une transmission de type push supplémentaire. Notez que les CompiledQueries sont uniquement compatibles avec les modèles dérivés de ObjectContext et ne sont pas compatibles avec les modèles dérivés de DbContext.

Pour plus d'informations sur la création et l'appel d'un CompiledQuery, consultez [requêtes compilées \(LINQ to Entities\)](#).

Il y a deux points à prendre en compte lors de l'utilisation d'un CompiledQuery, à savoir la nécessité d'utiliser des instances statiques et les problèmes qu'ils ont avec la composabilité. Vous trouverez ci-dessous une explication détaillée de ces deux considérations.

#### 3.3.1 utiliser des instances CompiledQuery statiques

Dans la mesure où la compilation d'une requête LINQ est un processus long, nous ne voulons pas la faire chaque fois que nous devons extraire des données de la base de données. Les instances CompiledQuery vous permettent de compiler une seule fois et de les exécuter plusieurs fois, mais vous devez faire attention et demander à réutiliser la même instance de CompiledQuery à chaque fois au lieu de la compiler à nouveau. L'utilisation de membres statiques pour stocker les instances CompiledQuery devient nécessaire ; dans le cas contraire, vous ne verrez aucun avantage.

Par exemple, supposons que votre page possède le corps de méthode suivant pour gérer l'affichage des produits pour la catégorie sélectionnée :

```
// Warning: this is the wrong way of using CompiledQuery
using (NorthwindEntities context = new NorthwindEntities())
{
    string selectedCategory = this.categoriesList.SelectedValue;

    var productsForCategory = CompiledQuery.Compile<NorthwindEntities, string, IQueryable<Product>>(
        (NorthwindEntities nwnd, string category) =>
            nwnd.Products.Where(p => p.Category.CategoryName == category)
    );

    this.productsGrid.DataSource = productsForCategory.Invoke(context, selectedCategory).ToList();
    this.productsGrid.DataBind();
}

this.productsGrid.Visible = true;
```

Dans ce cas, vous allez créer une nouvelle instance CompiledQuery à la volée chaque fois que la méthode est appelée. Au lieu de voir les avantages en matière de performances en récupérant la commande de stockage dans le cache du plan de requête, le CompiledQuery passera par le compilateur de plan chaque fois qu'une nouvelle instance sera créée. En fait, vous allez polluer votre cache de plan de requête avec une nouvelle entrée CompiledQuery chaque fois que la méthode est appelée.

Au lieu de cela, vous souhaitez créer une instance statique de la requête compilée, de sorte que vous appelez la même requête compilée chaque fois que la méthode est appelée. Pour ce faire, vous pouvez ajouter l'instance CompiledQuery en tant que membre du contexte de l'objet. Vous pouvez ensuite faire un petit nettoyage en accédant au CompiledQuery par le biais d'une méthode d'assistance :

```
public partial class NorthwindEntities : ObjectContext
{
    private static readonly Func<NorthwindEntities, string, IEnumerable<Product>> productsForCategoryCQ =
        CompiledQuery.Compile(
            (NorthwindEntities context, string categoryName) =>
                context.Products.Where(p => p.Category.CategoryName == categoryName)
        );

    public IEnumerable<Product> GetProductsForCategory(string categoryName)
    {
        return productsForCategoryCQ.Invoke(this, categoryName).ToList();
    }
}
```

Cette méthode d'assistance serait appelée comme suit :

```
this.productsGrid.DataSource = context.GetProductsForCategory(selectedCategory);
```

### 3.3.2 composer sur un CompiledQuery

La possibilité de composer une requête LINQ est très utile. pour ce faire, il vous suffit d'appeler une méthode après l'objet IQueryable, par exemple *Skip ()* ou *Count ()*. Toutefois, cela retourne essentiellement un nouvel objet IQueryable. Bien qu'il n'y ait rien à vous empêcher techniquelement de composer sur un CompiledQuery, cela

entraînera la génération d'un nouvel objet IQueryble qui nécessitera à nouveau de passer par le compilateur de plan.

Certains composants utilisent des objets IQueryble composés pour activer les fonctionnalités avancées. Par exemple, ASP.NET GridView peut être lié aux données d'un objet IQueryble via la propriété SelectMethod. Le GridView se compose alors de cet objet IQueryble pour permettre le tri et la pagination sur le modèle de données. Comme vous pouvez le voir, l'utilisation d'un CompiledQuery pour le GridView n'atteint pas la requête compilée, mais génère une nouvelle requête autocompilée.

Dans ce cas, il est possible d'ajouter des filtres progressifs à une requête. Par exemple, supposons que vous disposez d'une page clients avec plusieurs listes déroulantes pour les filtres facultatifs (par exemple, pays et OrdersCount). Vous pouvez composer ces filtres sur les résultats IQueryble d'un CompiledQuery, mais cela entraînera le passage de la nouvelle requête par le compilateur de plan chaque fois que vous l'exéutez.

```
using (NorthwindEntities context = new NorthwindEntities())
{
    IQueryable<Customer> myCustomers = context.InvokeCustomersForEmployee();

    if (this.orderCountFilterList.SelectedItem.Value != defaultFilterText)
    {
        int orderCount = int.Parse(orderCountFilterList.SelectedValue);
        myCustomers = myCustomers.Where(c => c.Orders.Count > orderCount);
    }

    if (this.countryFilterList.SelectedItem.Value != defaultFilterText)
    {
        myCustomers = myCustomers.Where(c => c.Address.Country == countryFilterList.SelectedValue);
    }

    this.customersGrid.DataSource = myCustomers;
    this.customersGrid.DataBind();
}
```

Pour éviter cette nouvelle compilation, vous pouvez réécrire le CompiledQuery pour prendre en compte les filtres possibles :

```
private static readonly Func<NorthwindEntities, int, int?, string, IQueryable<Customer>>
customersForEmployeeWithFiltersCQ = CompiledQuery.Compile(
    (NorthwindEntities context, int empId, int? countFilter, string countryFilter) =>
    context.Customers.Where(c => c.Orders.Any(o => o.EmployeeID == empId))
        .Where(c => countFilter.HasValue == false || c.Orders.Count > countFilter)
        .Where(c => countryFilter == null || c.Address.Country == countryFilter)
);
```

Qui serait appelé dans l'interface utilisateur comme :

```

using (NorthwindEntities context = new NorthwindEntities())
{
    int? countFilter = (this.orderCountFilterList.SelectedIndex == 0) ?
        (int?)null :
        int.Parse(this.orderCountFilterList.SelectedValue);

    string countryFilter = (this.countryFilterList.SelectedIndex == 0) ?
        null :
        this.countryFilterList.SelectedValue;

    IQueryable<Customer> myCustomers = context.InvokeCustomersForEmployeeWithFilters(
        countFilter, countryFilter);

    this.customersGrid.DataSource = myCustomers;
    this.customersGrid.DataBind();
}

```

Un compromis ici est que la commande de magasin générée disposera toujours des filtres avec les vérifications de valeur null, mais celles-ci doivent être assez simples pour que le serveur de base de données les optimise :

```

...
WHERE ((0 = (CASE WHEN (@p_linq_1 IS NOT NULL) THEN cast(1 as bit) WHEN (@p_linq_1 IS NULL) THEN cast(0 as bit) END)) OR ([Project3].[C2] > @p_linq_2)) AND (@p_linq_3 IS NULL OR [Project3].[Country] = @p_linq_4)

```

### 3.4 mise en cache des métadonnées

Le Entity Framework prend également en charge la mise en cache des métadonnées. Il s'agit essentiellement d'une mise en cache des informations de type et des informations de mappage de type à base de données entre différentes connexions au même modèle. Le cache des métadonnées est unique par AppDomain.

#### 3.4.1-algorithme de mise en cache des métadonnées

1. Les informations de métadonnées d'un modèle sont stockées dans un ItemCollection pour chaque EntityConnection.
  - En guise de note, il existe différents objets ItemCollection pour différentes parties du modèle. Par exemple, StoreItemCollections contient les informations sur le modèle de base de données ; ObjectItemCollection contient des informations sur le modèle de données ; EdmItemCollection contient des informations sur le modèle conceptuel.
2. Si deux connexions utilisent la même chaîne de connexion, elles partagent la même instance de ItemCollection.
3. Fonctionnellement équivalente mais les chaînes de connexion différentes textuellement peuvent entraîner des caches de métadonnées différents. Comme nous jetons des chaînes de connexion, il suffit de modifier l'ordre des jetons pour créer des métadonnées partagées. Toutefois, deux chaînes de connexion apparemment fonctionnellement identiques peuvent ne pas être évaluées comme identiques après la création de jetons.
4. L'utilisation de ItemCollection est régulièrement vérifiée. S'il est déterminé qu'un espace de travail n'a pas été accédé récemment, il est marqué pour nettoyage lors du prochain balayage du cache.
5. Le simple fait de créer un EntityConnection entraîne la création d'un cache de métadonnées (bien que les collections d'éléments qu'il contient ne soient pas initialisées tant que la connexion n'est pas ouverte). Cet espace de travail reste en mémoire jusqu'à ce que l'algorithme de mise en cache détermine qu'il n'est pas « en cours d'utilisation ».

L'équipe de conseil clientèle a écrit un billet de blog décrivant contenant une référence à un ItemCollection afin d'éviter le « obsolescence » lors de l'utilisation de grands modèles :

<<http://blogs.msdn.com/b/appfabriccat/archive/2010/10/22/metadataworkspace-reference-in-wcf->

services.aspx>.

#### 3.4.2 la relation entre la mise en cache des métadonnées et la mise en cache du plan de requête

L'instance du cache du plan de requête se trouve dans l'ItemCollection des types de magasins de MetadataWorkspace. Cela signifie que les commandes de stockage mises en cache sont utilisées pour les requêtes sur tout contexte instancié à l'aide d'un MetadataWorkspace donné. Cela signifie également que si vous avez deux chaînes de connexion qui sont légèrement différentes et ne correspondent pas après la création de jetons, vous aurez des instances de cache de plan de requête différentes.

### 3.5 mise en cache des résultats

Avec la mise en cache des résultats (également appelée « mise en cache de second niveau »), vous conservez les résultats des requêtes dans un cache local. Lors de l'émission d'une requête, vous voyez d'abord si les résultats sont disponibles localement avant d'effectuer une requête sur le magasin. Si la mise en cache des résultats n'est pas directement prise en charge par Entity Framework, il est possible d'ajouter un cache de second niveau à l'aide d'un fournisseur d'encapsulation. Un exemple de fournisseur d'encapsulation avec un cache de second niveau est le [cache de second niveau de Alachisoft Entity Framework basé sur NCache](#).

Cette implémentation de la mise en cache de second niveau est une fonctionnalité injectée qui a lieu une fois que l'expression LINQ a été évaluée (et funcletized) et que le plan d'exécution de la requête est calculé ou récupéré à partir du cache de premier niveau. Le cache de second niveau stocke alors uniquement les résultats bruts de la base de données, de sorte que le pipeline de matérialisation est toujours exécuté par la suite.

#### 3.5.1 Références supplémentaires pour la mise en cache des résultats avec le fournisseur d'encapsulation

- Julie Lerman a écrit un article « mise en cache de second niveau dans Entity Framework et Windows Azure » qui inclut la mise à jour de l'exemple de fournisseur d'encapsulation pour utiliser la mise en cache de Windows Server AppFabric : <https://msdn.microsoft.com/magazine/hh394143.aspx>
- Si vous travaillez avec Entity Framework 5, le blog de l'équipe a une requête post qui décrit la procédure à effectuer des actions en cours d'exécution avec le fournisseur de mise en cache pour Entity Framework 5 : <<http://blogs.msdn.com/b/adonet/archive/2010/09/13/ef-caching-with-jarek-kowalski-s-provider.aspx>>. Il comprend également un modèle T4 pour vous aider à automatiser l'ajout de la mise en cache de 2e niveau à votre projet.

## 4 requêtes autocompilées

Lorsqu'une requête est émise sur une base de données à l'aide d'Entity Framework, elle doit passer par une série d'étapes avant de matérialiser les résultats ; une de ces étapes est la compilation des requêtes. Entity SQL requêtes étaient connues pour avoir de bonnes performances, car elles sont automatiquement mises en cache. par conséquent, la deuxième ou la troisième fois que vous exécutez la même requête, elle peut ignorer le compilateur de plan et utiliser le plan mis en cache à la place.

Entity Framework 5 a également introduit la mise en cache automatique pour les requêtes LINQ to Entities. Dans les éditions précédentes de Entity Framework la création d'un CompiledQuery pour accélérer vos performances était une pratique courante, car cela permet de mettre en cache votre requête LINQ to Entities. Étant donné que la mise en cache s'effectue automatiquement sans l'utilisation d'un CompiledQuery, nous appelons cette fonctionnalité « requêtes autocompilées ». Pour plus d'informations sur le cache du plan de requête et ses mécanismes, consultez [mise en cache des plans de requête](#).

Entity Framework détecte quand une requête doit être recompilée, et le fait lorsque la requête est appelée même si elle a été compilée auparavant. Les conditions courantes qui provoquent la recompilation de la requête sont les suivantes :

- Modification du MergeOption associé à votre requête. La requête mise en cache ne sera pas utilisée, à la place, le compilateur de plan s'exécutera à nouveau et le plan nouvellement créé est mis en cache.
- Modification de la valeur de ContextOptions. UseCSharpNullComparisonBehavior. Vous avez le même effet que la modification de MergeOption.

D'autres conditions peuvent empêcher votre requête d'utiliser le cache. Voici quelques exemples courants :

- Utilisation d'`IEnumerable<T>.Contient<T>(valeur T)`.
- Utilisation de fonctions qui produisent des requêtes avec des constantes.
- Utilisation des propriétés d'un objet non mappé.
- Liaison de votre requête à une autre requête qui nécessite d'être recompilée.

#### 4.1 utilisation de `IEnumerable<T>.Contient<T>(valeur T)`

Entity Framework ne met pas en cache les requêtes qui appellent `IEnumerable<T>.Contient<T>(T)` par rapport à une collection en mémoire, étant donné que les valeurs de la collection sont considérées comme volatiles.

L'exemple de requête suivant ne sera pas mis en cache et sera donc toujours traité par le compilateur de plan :

```
int[] ids = new int[10000];
...
using (var context = new MyContext())
{
    var query = context.MyEntities
        .Where(entity => ids.Contains(entity.Id));

    var results = query.ToList();
    ...
}
```

Notez que la taille de l'`IEnumerable` par rapport à laquelle est exécutée détermine la vitesse de compilation de votre requête. Les performances peuvent être considérablement affectées lors de l'utilisation de collections volumineuses, telles que celle illustrée dans l'exemple ci-dessus.

Entity Framework 6 contient des optimisations de la façon dont `IEnumerable<T>.Contient<T>(valeur T)` fonctionne lorsque les requêtes sont exécutées. Le code SQL généré est beaucoup plus rapide à produire et plus lisible, et dans la plupart des cas, il s'exécute également plus rapidement sur le serveur.

#### 4.2 utilisation de fonctions qui produisent des requêtes avec des constantes

Les opérateurs LINQ `Skip()`, `Take()`, `Contains()` et `DefaultIfEmpty()` ne génèrent pas de requêtes SQL avec des paramètres, mais placent les valeurs qui leur sont passées en tant que constantes. Pour cette raison, les requêtes qui peuvent sinon être identiques finissent par polluer le cache du plan de requête, à la fois sur la pile EF et sur le serveur de base de données, et ne sont pas réutilisées, sauf si les mêmes constantes sont utilisées lors de l'exécution d'une requête ultérieure. Exemple :

```
var id = 10;
...
using (var context = new MyContext())
{
    var query = context.MyEntities.Select(entity => entity.Id).Contains(id);

    var results = query.ToList();
    ...
}
```

Dans cet exemple, chaque fois que cette requête est exécutée avec une valeur différente pour ID, la requête est compilée dans un nouveau plan.

En particulier, soyez attentif à l'utilisation de `Skip` et `Take` lors de la pagination. Dans EF6, ces méthodes ont une surcharge lambda qui rend effectivement le plan de requête mis en cache réutilisable, car EF peut capturer les variables passées à ces méthodes et les traduire en `SQLparameters`. Cela permet également de conserver le nettoyeur de cache, car dans le cas contraire, chaque requête avec une constante différente pour `Skip` et `Take` obtient sa propre entrée de cache de plan de requête.

Considérez le code suivant, qui est non optimal, mais qui est uniquement destiné à illustrer cette classe de requêtes :

```
var customers = context.Customers.OrderBy(c => c.LastName);
for (var i = 0; i < count; ++i)
{
    var currentCustomer = customers.Skip(i).FirstOrDefault();
    ProcessCustomer(currentCustomer);
}
```

Une version plus rapide de ce même code implique l'appel de SKIP avec une expression lambda :

```
var customers = context.Customers.OrderBy(c => c.LastName);
for (var i = 0; i < count; ++i)
{
    var currentCustomer = customers.Skip(() => i).FirstOrDefault();
    ProcessCustomer(currentCustomer);
}
```

Le second extrait de code peut s'exécuter jusqu'à 11% plus rapidement, car le même plan de requête est utilisé chaque fois que la requête est exécutée, ce qui permet d'économiser du temps processeur et d'éviter de polluer le cache des requêtes. En outre, étant donné que le paramètre à ignorer se trouve dans une fermeture, le code peut également ressembler à ceci :

```
var i = 0;
var skippyCustomers = context.Customers.OrderBy(c => c.LastName).Skip(() => i);
for (; i < count; ++i)
{
    var currentCustomer = skippyCustomers.FirstOrDefault();
    ProcessCustomer(currentCustomer);
}
```

#### 4.3 utilisation des propriétés d'un objet non mappé

Quand une requête utilise les propriétés d'un type d'objet non mappé comme paramètre, la requête n'est pas mise en cache. Exemple :

```
using (var context = new MyContext())
{
    var myObject = new NonMappedType();

    var query = from entity in context.MyEntities
                where entity.Name.StartsWith(myObject.MyProperty)
                select entity;

    var results = query.ToList();
    ...
}
```

Dans cet exemple, supposons que la classe NonMappedType ne fait pas partie du modèle d'entité. Cette requête peut facilement être modifiée pour ne pas utiliser un type non mappé et utiliser à la place une variable locale comme paramètre de la requête :

```

using (var context = new MyContext())
{
    var myObject = new NonMappedType();
    var myValue = myObject.MyProperty;
    var query = from entity in context.MyEntities
                where entity.Name.StartsWith(myValue)
                select entity;

    var results = query.ToList();
    ...
}

```

Dans ce cas, la requête peut être mise en cache et tirer parti du cache du plan de requête.

#### **4.4 liaison à des requêtes qui nécessitent une recompilation**

En suivant le même exemple que ci-dessus, si vous avez une deuxième requête qui s'appuie sur une requête qui doit être recompilée, la totalité de votre seconde requête sera également recompilée. Voici un exemple illustrant ce scénario :

```

int[] ids = new int[10000];
...
using (var context = new MyContext())
{
    var firstQuery = from entity in context.MyEntities
                     where ids.Contains(entity.Id)
                     select entity;

    var secondQuery = from entity in context.MyEntities
                      where firstQuery.Any(otherEntity => otherEntity.Id == entity.Id)
                      select entity;

    var results = secondQuery.ToList();
    ...
}

```

L'exemple est générique, mais il illustre la façon dont la liaison à firstQuery entraîne l'impossibilité pour secondQuery de se mettre en cache. Si firstQuery n'était pas une requête nécessitant une recompilation, secondQuery aurait été mis en cache.

## **5 requêtes de non-suivi**

### **5.1 désactivation du suivi des modifications pour réduire la surcharge de gestion de l'État**

Si vous êtes dans un scénario en lecture seule et que vous souhaitez éviter la surcharge liée au chargement des objets dans le ObjectStateManager, vous pouvez émettre des requêtes « aucune suivi ». Le suivi des modifications peut être désactivé au niveau de la requête.

Notez, toutefois, que si vous désactivez le suivi des modifications, vous désactivez efficacement le cache d'objets. Lorsque vous interrogez une entité, nous ne pouvons pas ignorer la matérialisation en extrayant les résultats de requête matérialisés précédemment à partir de ObjectStateManager. Si vous interrogez à plusieurs reprises les mêmes entités sur le même contexte, vous pouvez en fait voir un avantage en matière de performances de l'activation du suivi des modifications.

Lors de l'interrogation à l'aide d'ObjectContext, les instances ObjectQuery et ObjectSet se souviennent d'un MergeOption une fois qu'il est défini, et les requêtes qui y sont composées héritent du MergeOption effectif de la requête parente. Lors de l'utilisation de DbContext, le suivi peut être désactivé en appelant le modificateur AsNoTracking () sur DbSet.

#### **5.1.1 désactivation du suivi des modifications pour une requête lors de l'utilisation de DbContext**

Vous pouvez basculer le mode d'une requête pour qu'elle soit très suivie en chaîné un appel à la méthode AsNoTracking () dans la requête. Contrairement à ObjectQuery, les classes DbSet et DbQuery de l'API DbContext n'ont pas de propriété mutable pour MergeOption.

```
var productsForCategory = from p in context.Products.AsNoTracking()
                           where p.Category.CategoryName == selectedCategory
                           select p;
```

### 5.1.2 désactivation du suivi des modifications au niveau de la requête à l'aide d'ObjectContext

```
var productsForCategory = from p in context.Products
                           where p.Category.CategoryName == selectedCategory
                           select p;

((ObjectQuery)productsForCategory).MergeOption = MergeOption.NoTracking;
```

### 5.1.3 désactivation du suivi des modifications pour un ensemble d'entités complet à l'aide d'ObjectContext

```
context.Products.MergeOption = MergeOption.NoTracking;

var productsForCategory = from p in context.Products
                           where p.Category.CategoryName == selectedCategory
                           select p;
```

## 5.2 métriques de test illustrant l'avantage en matière de performances des requêtes NoTracking

Dans ce test, nous examinons le coût du remplissage de la ObjectStateManager en comparant le suivi à la non-suivi des requêtes pour le modèle Navision. Reportez-vous à l'annexe pour obtenir une description du modèle Navision et des types de requêtes qui ont été exécutés. Dans ce test, nous parcourons la liste des requêtes et les exécutons une fois. Nous avons exécuté deux variantes du test, une fois avec les requêtes NoTracking et une fois avec l'option de fusion par défaut « AppendOnly ». Nous avons exécuté chaque variante 3 fois et prenons la valeur moyenne des exécutions. Entre les tests, nous effaçons le cache des requêtes sur le SQL Server et réduisons la base de données tempdb en exécutant les commandes suivantes :

1. DBCC DROPCLEANBUFFERS
2. DBCC FREEPROCCACHE
3. DBCC SHRINKDATABASE (tempdb, 0)

Résultats des tests, median sur 3 s'exécute :

|                           | AUCUN SUIVI-PLAGE DE TRAVAIL | AUCUN SUIVI – HEURE | AJOUTER UNIQUEMENT-PLAGE DE TRAVAIL | AJOUTER UNIQUEMENT – HEURE |
|---------------------------|------------------------------|---------------------|-------------------------------------|----------------------------|
| <b>Entity Framework 5</b> | 460361728                    | 1163536 ms          | 596545536                           | 1273042 ms                 |
| <b>Entity Framework 6</b> | 647127040                    | 190228 ms           | 832798720                           | 195521 ms                  |

Entity Framework 5 aura un faible encombrement mémoire à la fin de l'exécution que Entity Framework 6. La mémoire supplémentaire consommée par Entity Framework 6 est le résultat de structures de mémoire et de code supplémentaires qui activent de nouvelles fonctionnalités et de meilleures performances.

Il y a également une différence évidente en matière d'encombrement mémoire lors de l'utilisation de ObjectStateManager. Entity Framework 5 a augmenté son empreinte de 30% pendant le suivi de toutes les entités que nous avons matérialisées à partir de la base de données. Entity Framework 6 a augmenté son empreinte de

28%.

En termes de temps, Entity Framework 6 s'exécute Entity Framework 5 dans ce test par une marge importante. Entity Framework 6 a terminé le test en environ 16% du temps consommé par Entity Framework 5. En outre, Entity Framework 5 prend plus de 9% de temps pour s'exécuter lorsque le ObjectStateManager est utilisé. En comparaison, Entity Framework 6 utilise 3% de temps supplémentaire lors de l'utilisation de ObjectStateManager.

## 6 options d'exécution de requête

Entity Framework propose différentes façons d'interroger. Nous allons examiner les options suivantes, comparer les avantages et les inconvénients de chacun et examiner leurs caractéristiques de performances :

- LINQ to Entities.
- Aucune LINQ to Entities de suivi.
- Entity SQL sur un ObjectQuery.
- Entity SQL sur un EntityCommand.
- ExecuteStoreQuery.
- SqlQuery.
- CompiledQuery.

### 6,1 requêtes de LINQ to Entities

```
var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
```

#### Professionnels

- Adapté aux opérations CUD.
- Objets entièrement matérialisés.
- Plus simple à écrire avec la syntaxe intégrée dans le langage de programmation.
- Bonnes performances.

#### Inconvénients

- Certaines restrictions techniques, telles que :
  - Les modèles utilisant DefaultIfEmpty pour les requêtes de jointure externe génèrent des requêtes plus complexes que les instructions de jointure externe simples dans Entity SQL.
  - Vous ne pouvez toujours pas utiliser LIKE avec les critères spéciaux.

### 6,2 aucune requête de LINQ to Entities de suivi

Lorsque le contexte dérive de ObjectContext :

```
context.Products.MergeOption = MergeOption.NoTracking;  
var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
```

Lorsque le contexte dérive DbContext :

```
var q = context.Products.AsNoTracking()  
    .Where(p => p.Category.CategoryName == "Beverages");
```

#### Professionnels

- Performances améliorées par rapport aux requêtes LINQ normales.
- Objets entièrement matérialisés.

- Plus simple à écrire avec la syntaxe intégrée dans le langage de programmation.

### Inconvénients

- Non adapté aux opérations CUD.
- Certaines restrictions techniques, telles que :
  - Les modèles utilisant DefaultIfEmpty pour les requêtes de jointure externe génèrent des requêtes plus complexes que les instructions de jointure externe simples dans Entity SQL.
  - Vous ne pouvez toujours pas utiliser LIKE avec les critères spéciaux.

Notez que les requêtes qui projettent les propriétés scalaires ne sont pas suivies, même si le NoTracking n'est pas spécifié. Exemple :

```
var q = context.Products.Where(p => p.Category.CategoryName == "Beverages").Select(p => new { p.ProductName });
}
```

Cette requête particulière ne spécifie pas explicitement la non-suivi, mais dans la mesure où elle ne matérialisent pas un type connu du gestionnaire d'état d'objet, le résultat matérialisé n'est pas suivi.

### 6,3 Entity SQL sur un ObjectQuery

```
ObjectQuery<Product> products = context.Products.Where("it.Category.CategoryName = 'Beverages'");
```

### Professionnels

- Adapté aux opérations CUD.
- Objets entièrement matérialisés.
- Prend en charge la mise en cache du plan de requête.

### Inconvénients

- Implique des chaînes de requête textuelles qui sont plus sujettes aux erreurs des utilisateurs que les constructions de requête intégrées au langage.

### 6,4 Entity SQL sur une commande d'entité

```
EntityCommand cmd = eConn.CreateCommand();
cmd.CommandText = "Select p From NorthwindEntities.Products As p Where p.Category.CategoryName = 'Beverages'";

using (EntityDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
{
    while (reader.Read())
    {
        // manually 'materialize' the product
    }
}
```

### Professionnels

- Prend en charge la mise en cache du plan de requête dans .NET 4,0 (la mise en cache du plan est prise en charge par tous les autres types de requêtes dans .NET 4,5).

### Inconvénients

- Implique des chaînes de requête textuelles qui sont plus sujettes aux erreurs des utilisateurs que les constructions de requête intégrées au langage.
- Non adapté aux opérations CUD.

- Les résultats ne sont pas matérialisés automatiquement et doivent être lus à partir du lecteur de données.

## 6,5 SqlQuery et ExecuteStoreQuery

SqlQuery sur la base de données :

```
// use this to obtain entities and not track them
var q1 = context.Database.SqlQuery<Product>("select * from products");
```

SqlQuery sur DbSet :

```
// use this to obtain entities and have them tracked
var q2 = context.Products.SqlQuery("select * from products");
```

ExecuteStoreQuery:

```
var beverages = context.ExecuteStoreQuery<Product>(
@"
    SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
    P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued, P.DiscontinuedDate
    FROM        Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
    WHERE       (C.CategoryName = 'Beverages')"
);
```

### Professionnels

- Performances généralement plus rapides dans la mesure où le compilateur de plan est contourné.
- Objets entièrement matérialisés.
- Adapté aux opérations CUD en cas d'utilisation à partir du DbSet.

### Inconvénients

- La requête est textuelle et sujette aux erreurs.
- La requête est liée à un serveur principal spécifique à l'aide de la sémantique de magasin au lieu de la sémantique conceptuelle.
- Lorsque l'héritage est présent, la requête handcrafted doit tenir compte des conditions de mappage pour le type demandé.

## 6,6 CompiledQuery

```
private static readonly Func<NorthwindEntities, string, IQueryable<Product>> productsForCategoryCQ =
CompiledQuery.Compile(
    (NorthwindEntities context, string categoryName) =>
    context.Products.Where(p => p.Category.CategoryName == categoryName)
);
...
var q = context.InvokeProductsForCategoryCQ("Beverages");
```

### Professionnels

- Offre une amélioration des performances pouvant atteindre 7% par rapport aux requêtes LINQ ordinaires.
- Objets entièrement matérialisés.
- Adapté aux opérations CUD.

### Inconvénients

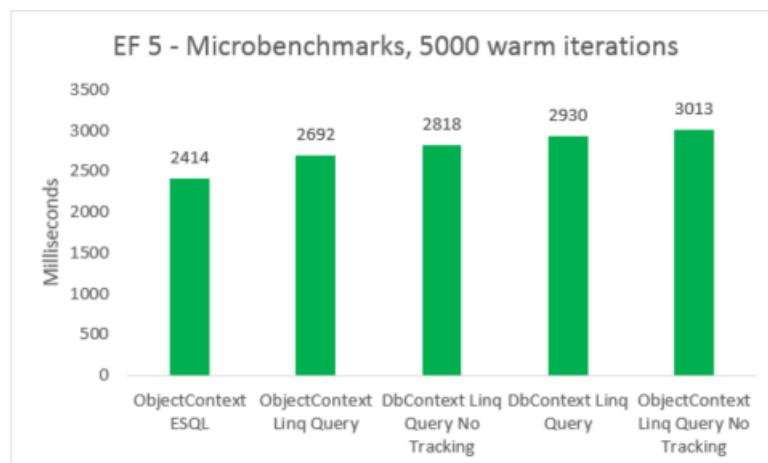
- Complexité accrue et programmation plus lourde.
- L'amélioration des performances est perdue lors de la composition d'une requête compilée.

- Certaines requêtes LINQ ne peuvent pas être écrites en tant que CompiledQuery, par exemple, des projections de types anonymes.

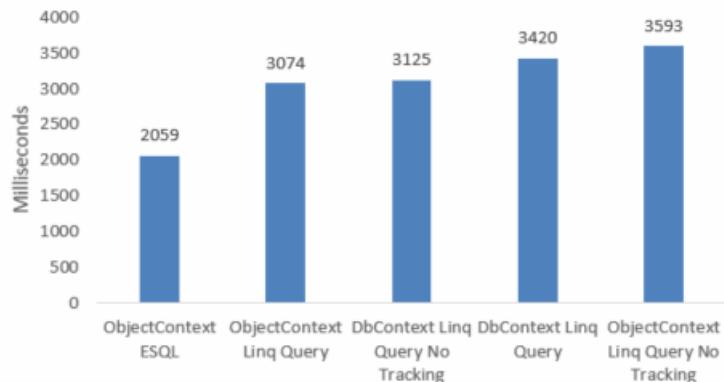
## 6.7 Comparaison des performances des différentes options de requête

Les microtests simples où la création de contexte n'a pas été chronométrée ont été placés dans le test. Nous avons mesuré l'interrogation 5000 fois pour un ensemble d'entités non mises en cache dans un environnement contrôlé. Ces valeurs doivent être prises en compte : elles ne reflètent pas les nombres réels produits par une application, mais elles représentent une mesure très précise de la différence de performances lorsque différentes options d'interrogation sont comparées. les Apples à pommes, à l'exclusion du coût de création d'un nouveau contexte.

| EF  | TEST                                      | TEMPS (MS) | MÉMOIRE  |
|-----|---|------------|----------|
| EF5 | ESQL ObjectContext                        | 2414       | 38801408 |
| EF5 | Requête LINQ ObjectContext                | 2692       | 38277120 |
| EF5 | Interrogation LINQ de DbContext non suivi | 2818       | 41840640 |
| EF5 | Requête LINQ DbContext                    | 2930       | 41771008 |
| EF5 | ObjectContext requête LINQ aucun suivi    | 3013       | 38412288 |
|     |   |            |          |
| EF6 | ESQL ObjectContext                        | 2059       | 46039040 |
| EF6 | Requête LINQ ObjectContext                | 3074       | 45248512 |
| EF6 | Interrogation LINQ de DbContext non suivi | 3125       | 47575040 |
| EF6 | Requête LINQ DbContext                    | 3420       | 47652864 |
| EF6 | ObjectContext requête LINQ aucun suivi    | 3593       | 45260800 |



### EF 6 - Microbenchmarks, 5000 warm iterations

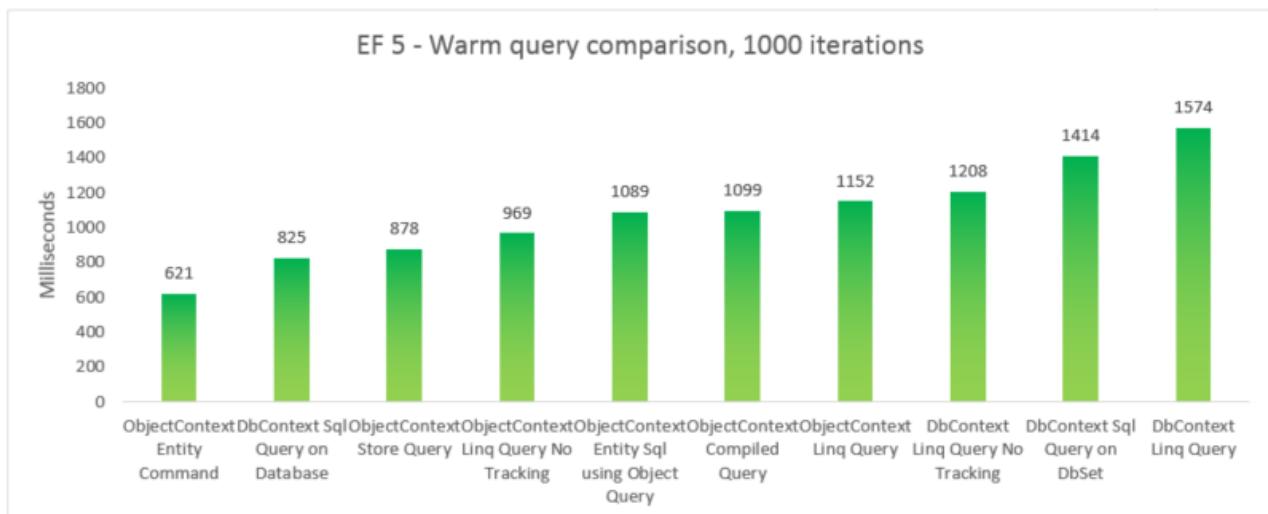


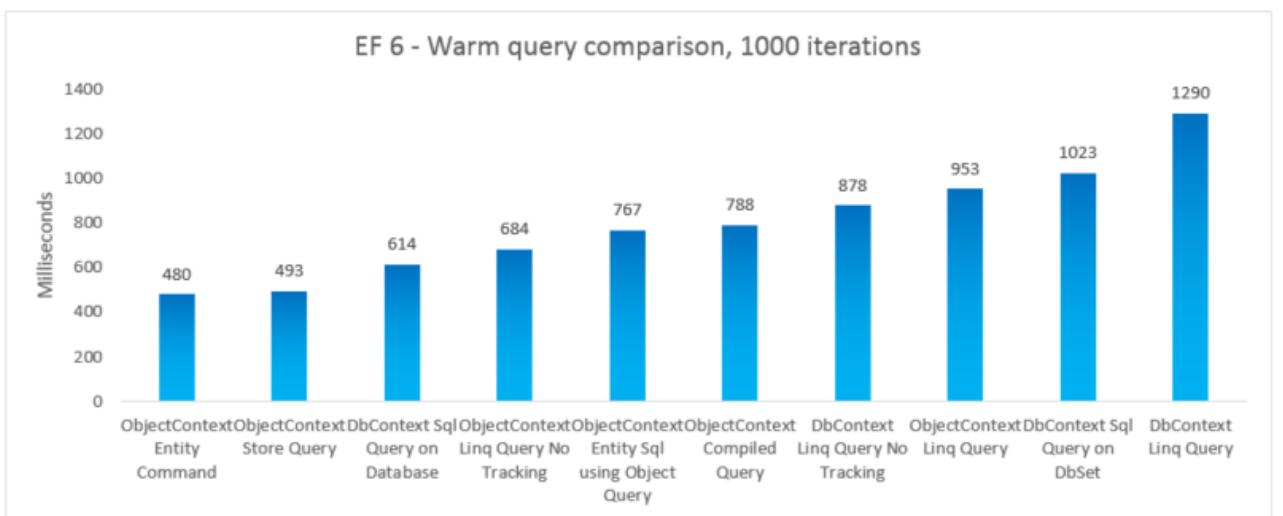
Les microtests sont très sensibles aux modifications mineures apportées au code. Dans ce cas, la différence entre les coûts de Entity Framework 5 et Entity Framework 6 est due à l'ajout d'une [interception](#) et à des [améliorations transactionnelles](#). Toutefois, ces microtests sont une vision amplifiée d'un très petit fragment de ce que Entity Framework fait. Les scénarios réels de requêtes à chaud ne doivent pas voir une régression des performances lors de la mise à niveau de Entity Framework 5 vers Entity Framework 6.

Pour comparer les performances réelles des différentes options de requête, nous avons créé 5 variantes de test distinctes où nous utilisons une option de requête différente pour sélectionner tous les produits dont le nom de catégorie est « boissons ». Chaque itération comprend le coût de la création du contexte et le coût de la matérialisation de toutes les entités rentrées. 10 itérations sont exécutées de façon ininterrompue avant d'effectuer la somme de 1000 itérations chronométrées. Les résultats affichés sont l'exécution médiane effectuée à partir de 5 exécutions de chaque test. Pour plus d'informations, consultez l'annexe B, qui comprend le code du test.

| EF  | TEST   | TEMPS (MS) | MÉMOIRE  |
|-----|--|------------|----------|
| EF5 | Commande d'entité<br>ObjectContext                         | 621        | 39350272 |
| EF5 | Requête SQL DbContext sur<br>la base de données            | 825        | 37519360 |
| EF5 | Requête du magasin<br>ObjectContext                        | 878        | 39460864 |
| EF5 | ObjectContext requête LINQ<br>aucun suivi                  | 969        | 38293504 |
| EF5 | ObjectContext Entity SQL à<br>l'aide d'une requête d'objet | 1089       | 38981632 |
| EF5 | Requête compilée<br>ObjectContext                          | 1099       | 38682624 |
| EF5 | Requête LINQ<br>ObjectContext                              | 1152       | 38178816 |
| EF5 | Interrogation LINQ de<br>DbContext non suivi               | 1208       | 41803776 |
| EF5 | Requête SQL DbContext sur<br>DbSet                         | 1414       | 37982208 |

| EF  | TEST   | TEMPS (MS) | MÉMOIRE  |
|-----|--|------------|----------|
| EF5 | Requête LINQ DbContext                                     | 1574       | 41738240 |
| EF6 | Commande d'entité<br>ObjectContext                         | 480        | 47247360 |
| EF6 | Requête du magasin<br>ObjectContext                        | 493        | 46739456 |
| EF6 | Requête SQL DbContext sur<br>la base de données            | 614        | 41607168 |
| EF6 | ObjectContext requête LINQ<br>aucun suivi                  | 684        | 46333952 |
| EF6 | ObjectContext Entity SQL à<br>l'aide d'une requête d'objet | 767        | 48865280 |
| EF6 | Requête compilée<br>ObjectContext                          | 788        | 48467968 |
| EF6 | Interrogation LINQ de<br>DbContext non suivi               | 878        | 47554560 |
| EF6 | Requête LINQ<br>ObjectContext                              | 953        | 47632384 |
| EF6 | Requête SQL DbContext sur<br>DbSet                         | 1023       | 41992192 |
| EF6 | Requête LINQ DbContext                                     | 1290       | 47529984 |





#### NOTE

À des fins d'exhaustivité, nous avons inclus une variation où nous exécutons une requête Entity SQL sur un EntityCommand. Toutefois, étant donné que les résultats ne sont pas matérialisés pour ces requêtes, la comparaison n'est pas nécessairement de pommes à pommes. Le test comprend une approximation de la matérialisation pour essayer de rendre la comparaison plus équitable.

Dans ce cas de bout en bout, Entity Framework 6 s'exécute Entity Framework 5 en raison des améliorations de performances apportées sur plusieurs parties de la pile, y compris une initialisation beaucoup plus légère et une MetadataCollection plus rapide<T> des recherches.

## 7 Considérations relatives aux performances au moment de la conception

### 7.1 stratégies d'héritage

Une autre considération en matière de performances lors de l'utilisation de Entity Framework est la stratégie d'héritage que vous utilisez. Entity Framework prend en charge 3 types de base d'héritage et leurs combinaisons :

- Table par hiérarchie (TPH) : où chaque ensemble d'héritage est mappé à une table avec une colonne de discriminateur pour indiquer quel type particulier dans la hiérarchie est représenté dans la ligne.
- Table par type (TPT) : où chaque type a sa propre table dans la base de données ; les tables enfants définissent uniquement les colonnes que la table parente ne contient pas.
- Table par classe (TPC) : chaque type possède sa propre table complète dans la base de données ; les tables enfants définissent tous leurs champs, y compris ceux définis dans les types parents.

Si votre modèle utilise l'héritage TPT, les requêtes générées seront plus complexes que celles générées avec les autres stratégies d'héritage, ce qui peut entraîner des durées d'exécution plus longues sur le magasin. Elle prend généralement plus de temps pour générer des requêtes sur un modèle TPT et pour matérialiser les objets résultants.

Consultez les « considérations sur les performances lors de l'utilisation de l'héritage TPT (Table par Type) dans le cadre de l'entité » billet de blog MSDN : <<http://blogs.msdn.com/b/adonet/archive/2010/08/17/performance-considerations-when-using-tpt-table-per-type-inheritance-in-the-entity-framework.aspx>>.

#### 7.1.1 éviter les TPT dans les applications Model First ou Code First

Lorsque vous créez un modèle sur une base de données existante qui a un schéma TPT, vous ne disposez pas de nombreuses options. Toutefois, lors de la création d'une application à l'aide de Model First ou Code First, vous devez éviter l'héritage TPT pour les problèmes de performances.

Lorsque vous utilisez Model First dans l'Assistant Entity Designer, vous obtiendrez TPT pour tout héritage dans

vos modèles. Si vous souhaitez basculer vers une stratégie de l'héritage TPH avec Model First, vous pouvez utiliser le « Entity Designer de base de données Generation Power Pack » disponible à partir de la galerie Visual Studio (<http://visualstudiogallery.msdn.microsoft.com/df3541c3-d833-4b65-b942-989e7ec74c87/>).

Lorsque vous utilisez Code First pour configurer le mappage d'un modèle avec héritage, EF utilise TPH par défaut. par conséquent, toutes les entités de la hiérarchie d'héritage sont mappées à la même table. Consultez la section « Mappage avec l'API Fluent » de l'article « Code première dans entité Framework4.1 » dans MSDN Magazine (<http://msdn.microsoft.com/magazine/hh126815.aspx>) pour plus d'informations.

## 7.2 mise à niveau à partir de EF4 pour améliorer l'heure de génération du modèle

Une amélioration spécifique à l'SQL Server de l'algorithme qui génère la couche de magasin (SSDL) du modèle est disponible dans Entity Framework 5 et 6, et en tant que mise à jour de Entity Framework 4 lorsque Visual Studio 2010 SP1 est installé. Les résultats des tests suivants illustrent l'amélioration lors de la génération d'un modèle très volumineux, dans ce cas le modèle Navision. Pour plus d'informations à ce sujet, consultez l'annexe C.

Le modèle contient les jeux d'entités 1005 et les ensembles d'associations 4227.

| CONFIGURATION                              | RÉPARTITION DU TEMPS CONSOMMÉ   |
|--|---|
| Visual Studio 2010, Entity Framework 4     | Génération SSDL : 2 h 27 min.<br>Génération de mappage : 1 seconde<br>Génération CSDL : 1 seconde<br>Génération ObjectLayer : 1 seconde<br>Génération de vues : 2 h 14 min.   |
| Visual Studio 2010 SP1, Entity Framework 4 | Génération SSDL : 1 seconde<br>Génération de mappage : 1 seconde<br>Génération CSDL : 1 seconde<br>Génération ObjectLayer : 1 seconde<br>Génération de vues : 1 heure 53 min. |
| Visual Studio 2013, Entity Framework 5     | Génération SSDL : 1 seconde<br>Génération de mappage : 1 seconde<br>Génération CSDL : 1 seconde<br>Génération ObjectLayer : 1 seconde<br>Génération de vues : 65 minutes      |
| Visual Studio 2013, Entity Framework 6     | Génération SSDL : 1 seconde<br>Génération de mappage : 1 seconde<br>Génération CSDL : 1 seconde<br>Génération ObjectLayer : 1 seconde<br>Génération de vues : 28 secondes.    |

Il est à noter que lors de la génération du langage SSDL, la charge est presque entièrement dépensée sur le SQL Server, tandis que l'ordinateur de développement client attend que les résultats soient renvoyés par le serveur. Les administrateurs de bases de l'intérêt devraient particulièrement apprécier cette amélioration. Il est également intéressant de noter que l'ensemble du coût de génération de modèle a lieu dans la génération de vues.

## 7.3 fractionnement des modèles volumineux avec Database First et Model First

À mesure que la taille du modèle augmente, l'aire du concepteur devient encombrée et difficile à utiliser. Nous considérons généralement un modèle avec plus de 300 entités comme trop volumineux pour utiliser efficacement le concepteur. Le billet de blog suivant décrit plusieurs options pour les modèles volumineux de fractionnement : <http://blogs.msdn.com/b/adonet/archive/2008/11/25/working-with-large-models-in-entity-framework-part-2.aspx>.

La publication a été écrite pour la première version de Entity Framework, mais les étapes s'appliquent toujours.

## 7.4 Considérations sur les performances avec le contrôle de source de données d'entité

Nous avons vu des cas dans des tests de performances et de contrainte multithread où les performances d'une application Web utilisant le contrôle EntityDataSource se détériorent considérablement. La cause sous-jacente est que le EntityDataSource appelle MetadataWorkspace.LoadFromAssembly à plusieurs reprises sur les assemblies référencés par l'application Web pour découvrir les types à utiliser comme entités.

La solution consiste à définir le ContextTypeName de EntityDataSource sur le nom de type de votre classe ObjectContext dérivée. Cela désactive le mécanisme qui analyse tous les assemblies référencés pour les types d'entité.

La définition du champ ContextTypeName empêche également un problème fonctionnel où le EntityDataSource dans .NET 4,0 lève une ReflectionTypeLoadException lorsqu'il ne peut pas charger un type à partir d'un assembly par le biais de la réflexion. Ce problème a été résolu dans .NET 4,5.

## 7,5 entités POCO et proxy de suivi des modifications

Entity Framework vous permet d'utiliser des classes de données personnalisées avec votre modèle de données sans modifier les classes de données elles-mêmes. Cela signifie que vous pouvez utiliser des objets CLR « classiques » (ou POCO), tels que les objets de domaine existants, avec votre modèle de données. Ces classes de données POCO (également appelées objets ignorant la persistance), qui sont mappées à des entités définies dans un modèle de données, prennent en charge la plupart des mêmes comportements de requête, d'insertion, de mise à jour et de suppression que les types d'entités générés par les outils de Entity Data Model.

Entity Framework pouvez également créer des classes proxy dérivées de vos types POCO, qui sont utilisées lorsque vous souhaitez activer des fonctionnalités telles que le chargement différé et le suivi automatique des modifications sur les entités POCO. Vos classes POCO doivent remplir certaines conditions pour permettre d'Entity Framework utiliser des proxys, comme indiqué ici : <http://msdn.microsoft.com/library/dd468057.aspx>.

Les proxys de suivi des chances notifient le gestionnaire d'état d'objet chaque fois que l'une des propriétés de vos entités a une valeur modifiée, de sorte que Entity Framework connaît tout le temps de l'état réel de vos entités. Pour ce faire, ajoutez des événements de notification au corps des méthodes setter de vos propriétés et faites en sorte que le gestionnaire d'état d'objet traite de tels événements. Notez que la création d'une entité de proxy est généralement plus coûteuse que la création d'une entité POCO non-proxy en raison de l'ensemble d'événements supplémentaires créé par Entity Framework.

Lorsqu'une entité POCO n'a pas de proxy de suivi des modifications, des modifications sont détectées en comparant le contenu de vos entités à une copie d'un état enregistré précédent. Cette comparaison profonde devient un processus long lorsque vous avez de nombreuses entités dans votre contexte, ou lorsque vos entités ont une très grande quantité de propriétés, même si aucune d'elles n'a changé depuis la dernière comparaison.

En Résumé : vous payez une baisse des performances lors de la création du proxy de suivi des modifications, mais le suivi des modifications vous permet d'accélérer le processus de détection des modifications lorsque vos entités ont de nombreuses propriétés ou lorsque vous avez de nombreuses entités dans votre modèle. Pour les entités avec un petit nombre de propriétés où la quantité d'entités n'augmente pas trop, l'utilisation de proxys de suivi des modifications n'est pas très avantageuse.

# 8 chargement des entités associées

## 8,1 chargement différé et chargement hâtif

Entity Framework offre différentes manières de charger les entités associées à votre entité cible. Par exemple, lorsque vous recherchez des produits, les commandes associées sont chargées dans le gestionnaire d'état d'objet de différentes façons. Du point de vue des performances, la plus grande question à prendre en compte lors du chargement des entités associées est l'utilisation du chargement différé ou du chargement hâtif.

Lorsque vous utilisez le chargement hâtif, les entités associées sont chargées en même temps que votre jeu d'entités cible. Vous utilisez une instruction include dans votre requête pour indiquer les entités associées que vous souhaitez importer.

Lorsque vous utilisez le chargement différé, votre requête initiale ne s'insère que dans le jeu d'entités cible. Toutefois, chaque fois que vous accédez à une propriété de navigation, une autre requête est émise sur le magasin pour charger l'entité associée.

Une fois qu'une entité a été chargée, toutes les autres requêtes de l'entité la chargent directement à partir du gestionnaire d'état d'objet, que vous utilisiez le chargement différé ou le chargement hâtif.

## 8.2 Comment choisir entre le chargement différé et le chargement hâtif

L'important est que vous compreniez la différence entre le chargement différé et le chargement hâtif afin que vous puissiez faire le bon choix pour votre application. Cela vous aidera à évaluer le compromis entre plusieurs demandes sur la base de données par rapport à une requête unique qui peut contenir une charge utile importante. Il peut être utile d'utiliser le chargement hâtif dans certaines parties de votre application et le chargement différé dans d'autres parties.

À titre d'exemple, supposons que vous souhaitiez interroger les clients qui vivent au Royaume-Uni et le nombre de commandes.

### Utilisation du chargement hâtif

```
using (NorthwindEntities context = new NorthwindEntities())
{
    var ukCustomers = context.Customers.Include(c => c.Orders).Where(c => c.Address.Country == "UK");
    var chosenCustomer = AskUserToPickCustomer(ukCustomers);
    Console.WriteLine("Customer Id: {0} has {1} orders", customer.CustomerID, customer.Orders.Count);
}
```

### Utilisation du chargement différé

```
using (NorthwindEntities context = new NorthwindEntities())
{
    context.ContextOptions.LazyLoadingEnabled = true;

    //Notice that the Include method call is missing in the query
    var ukCustomers = context.Customers.Where(c => c.Address.Country == "UK");

    var chosenCustomer = AskUserToPickCustomer(ukCustomers);
    Console.WriteLine("Customer Id: {0} has {1} orders", customer.CustomerID, customer.Orders.Count);
}
```

Lorsque vous utilisez le chargement hâtif, vous émettez une seule requête qui retourne tous les clients et toutes les commandes. La commande du Windows Store ressemble à ceci :

```

SELECT
[Project1].[C1] AS [C1],
[Project1].[CustomerID] AS [CustomerID],
[Project1].[CompanyName] AS [CompanyName],
[Project1].[ContactName] AS [ContactName],
[Project1].[ContactTitle] AS [ContactTitle],
[Project1].[Address] AS [Address],
[Project1].[City] AS [City],
[Project1].[Region] AS [Region],
[Project1].[PostalCode] AS [PostalCode],
[Project1].[Country] AS [Country],
[Project1].[Phone] AS [Phone],
[Project1].[Fax] AS [Fax],
[Project1].[C2] AS [C2],
[Project1].[OrderID] AS [OrderID],
[Project1].[CustomerID1] AS [CustomerID1],
[Project1].[EmployeeID] AS [EmployeeID],
[Project1].[OrderDate] AS [OrderDate],
[Project1].[RequiredDate] AS [RequiredDate],
[Project1].[ShippedDate] AS [ShippedDate],
[Project1].[ShipVia] AS [ShipVia],
[Project1].[Freight] AS [Freight],
[Project1].[ShipName] AS [ShipName],
[Project1].[ShipAddress] AS [ShipAddress],
[Project1].[ShipCity] AS [ShipCity],
[Project1].[ShipRegion] AS [ShipRegion],
[Project1].[ShipPostalCode] AS [ShipPostalCode],
[Project1].[ShipCountry] AS [ShipCountry]
FROM (
  SELECT
    [Extent1].[CustomerID] AS [CustomerID],
    [Extent1].[CompanyName] AS [CompanyName],
    [Extent1].[ContactName] AS [ContactName],
    [Extent1].[ContactTitle] AS [ContactTitle],
    [Extent1].[Address] AS [Address],
    [Extent1].[City] AS [City],
    [Extent1].[Region] AS [Region],
    [Extent1].[PostalCode] AS [PostalCode],
    [Extent1].[Country] AS [Country],
    [Extent1].[Phone] AS [Phone],
    [Extent1].[Fax] AS [Fax],
    1 AS [C1],
    [Extent2].[OrderID] AS [OrderID],
    [Extent2].[CustomerID] AS [CustomerID1],
    [Extent2].[EmployeeID] AS [EmployeeID],
    [Extent2].[OrderDate] AS [OrderDate],
    [Extent2].[RequiredDate] AS [RequiredDate],
    [Extent2].[ShippedDate] AS [ShippedDate],
    [Extent2].[ShipVia] AS [ShipVia],
    [Extent2].[Freight] AS [Freight],
    [Extent2].[ShipName] AS [ShipName],
    [Extent2].[ShipAddress] AS [ShipAddress],
    [Extent2].[ShipCity] AS [ShipCity],
    [Extent2].[ShipRegion] AS [ShipRegion],
    [Extent2].[ShipPostalCode] AS [ShipPostalCode],
    [Extent2].[ShipCountry] AS [ShipCountry],
    CASE WHEN ([Extent2].[OrderID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C2]
  FROM [dbo].[Customers] AS [Extent1]
  LEFT OUTER JOIN [dbo].[Orders] AS [Extent2] ON [Extent1].[CustomerID] = [Extent2].[CustomerID]
  WHERE N'UK' = [Extent1].[Country]
) AS [Project1]
ORDER BY [Project1].[CustomerID] ASC, [Project1].[C2] ASC

```

Lorsque vous utilisez le chargement différé, vous émettez initialement la requête suivante :

```

SELECT
[Extent1].[CustomerID] AS [CustomerID],
[Extent1].[CompanyName] AS [CompanyName],
[Extent1].[ContactName] AS [ContactName],
[Extent1].[ContactTitle] AS [ContactTitle],
[Extent1].[Address] AS [Address],
[Extent1].[City] AS [City],
[Extent1].[Region] AS [Region],
[Extent1].[PostalCode] AS [PostalCode],
[Extent1].[Country] AS [Country],
[Extent1].[Phone] AS [Phone],
[Extent1].[Fax] AS [Fax]
FROM [dbo].[Customers] AS [Extent1]
WHERE N'UK' = [Extent1].[Country]

```

Et chaque fois que vous accédez à la propriété de navigation Orders d'un client, une autre requête semblable à la suivante est émise sur le magasin :

```

exec sp_executesql N'SELECT
[Extent1].[OrderID] AS [OrderID],
[Extent1].[CustomerID] AS [CustomerID],
[Extent1].[EmployeeID] AS [EmployeeID],
[Extent1].[OrderDate] AS [OrderDate],
[Extent1].[RequiredDate] AS [RequiredDate],
[Extent1].[ShippedDate] AS [ShippedDate],
[Extent1].[ShipVia] AS [ShipVia],
[Extent1].[Freight] AS [Freight],
[Extent1].[ShipName] AS [ShipName],
[Extent1].[ShipAddress] AS [ShipAddress],
[Extent1].[ShipCity] AS [ShipCity],
[Extent1].[ShipRegion] AS [ShipRegion],
[Extent1].[ShipPostalCode] AS [ShipPostalCode],
[Extent1].[ShipCountry] AS [ShipCountry]
FROM [dbo].[Orders] AS [Extent1]
WHERE [Extent1].[CustomerID] = @EntityKeyValue1',N'@EntityKeyValue1 nchar(5)',@EntityKeyValue1=N'AROUT'

```

Pour plus d'informations, consultez [chargement d'objets connexes](#).

#### 8.2.1 chargement différé et aide-mémoire de chargement hâtif

Il n'existe aucun moyen de choisir un chargement hâtif et un chargement paresseux, à la fois. Essayez tout d'abord de comprendre les différences entre les deux stratégies pour pouvoir prendre une décision bien éclairée. Envisagez également si votre code s'adapte à l'un des scénarios suivants :

| SCÉNARIO   | NOTRE SUGGESTION  |
|--|---|
| Avez-vous besoin d'accéder à de nombreuses propriétés de navigation à partir des entités extraites ? | <p><b>Non</b> - les deux options vont probablement. Toutefois, si la charge de travail que votre requête est en cours d'exécution n'est pas trop importante, vous risquez d'obtenir des avantages en matière de performances en utilisant le chargement hâtif, car cela nécessite moins de boucles réseau pour matérialiser vos objets.</p> <p><b>Oui</b> : Si vous devez accéder à de nombreuses propriétés de navigation à partir des entités, vous pouvez le faire à l'aide de plusieurs instructions <code>Include</code> dans votre requête avec un chargement hâtif. Plus vous incluez d'entités, plus la charge utile que votre requête renverra est importante. Une fois que vous avez inclus trois entités ou plus dans votre requête, envisagez de basculer vers le chargement différé.</p> |

| SCÉNARIO   | NOTRE SUGGESTION  |
|--|---|
| Savez-vous exactement quelles données seront nécessaires au moment de l'exécution ?  | <b>Le chargement</b> en différé est mieux adapté à vos besoins. Dans le cas contraire, vous risquez de vous retrouver dans l'interrogation des données dont vous n'aurez pas besoin.  |
| Votre code s'exécute-t-il loin de votre base de données ? (latence accrue du réseau) | <b>Oui</b> -le chargement hâtif est probablement la meilleure solution. Cela permet de charger plus rapidement les jeux complets. Si votre requête nécessite la récupération d'une très grande quantité de données et que cette opération devient trop lente, essayez plutôt le chargement différé.<br><br><b>Non</b> -lorsque la latence du réseau n'est pas un problème, l'utilisation du chargement différé peut simplifier votre code. N'oubliez pas que la topologie de votre application peut changer. par conséquent, n'accordez pas de proximité à la base de données.<br><br><b>Oui</b> -lorsque le réseau est un problème, vous seul pouvez décider de ce qui convient le mieux à votre scénario. En général, le chargement hâtif est meilleur car il nécessite moins de boucles. |

### 8.2.2 problèmes de performances avec plusieurs includes

Lorsque nous entendons des questions sur les performances qui impliquent des problèmes de temps de réponse du serveur, la source du problème est souvent une requête avec plusieurs instructions include. Bien que l'inclusion d'entités associées dans une requête soit puissante, il est important de comprendre ce qui se passe en coulisses.

L'exécution d'une requête contenant plusieurs instructions Include dans le compilateur de plan interne pour générer la commande de stockage prend un temps relativement long. La majeure partie de ce temps est consacrée à la tentative d'optimisation de la requête résultante. La commande Store générée contient une jointure ou une Union externe pour chaque include, en fonction de votre mappage. Les requêtes de ce type importent des graphiques connectés volumineux à partir de votre base de données dans une seule charge utile, ce qui acerbate tous les problèmes de bande passante, en particulier lorsqu'il y a beaucoup de redondance dans la charge utile (par exemple, lorsque plusieurs niveaux de include sont utilisés pour traverser associations dans la direction un-à-plusieurs).

Vous pouvez vérifier les cas où vos requêtes renvoient des charges utiles excessivement volumineuses en accédant à la valeur TSQL sous-jacente pour la requête à l'aide de ToTraceString et en exécutant la commande Store dans SQL Server Management Studio pour afficher la taille de la charge utile. Dans ce cas, vous pouvez essayer de réduire le nombre d'instructions Include dans votre requête pour importer simplement les données dont vous avez besoin. Vous pouvez également décomposer votre requête en une plus petite séquence de sous-requêtes, par exemple :

#### Avant de rompre la requête :

```
using (NorthwindEntities context = new NorthwindEntities())
{
    var customers = from c in context.Customers.Include(c => c.Orders)
                    where c.LastName.StartsWith(lastNameParameter)
                    select c;

    foreach (Customer customer in customers)
    {
        ...
    }
}
```

#### Après avoir endommagé la requête :

```

using (NorthwindEntities context = new NorthwindEntities())
{
    var orders = from o in context.Orders
                 where o.Customer.LastName.StartsWith(lastNameParameter)
                 select o;

    orders.Load();

    var customers = from c in context.Customers
                    where c.LastName.StartsWith(lastNameParameter)
                    select c;

    foreach (Customer customer in customers)
    {
        ...
    }
}

```

Cela ne fonctionne que sur les requêtes suivies, car nous utilisons la capacité du contexte à effectuer automatiquement la résolution d'identité et la correction d'association.

Comme avec le chargement différé, le compromis sera plus de requêtes pour les charges utiles plus petites. Vous pouvez également utiliser des projections de propriétés individuelles pour sélectionner explicitement uniquement les données dont vous avez besoin à partir de chaque entité, mais vous ne devez pas charger d'entités dans ce cas, et les mises à jour ne seront pas prises en charge.

#### **solution de contournement 8.2.3 pour bénéficier du chargement différé des propriétés**

Entity Framework ne prend actuellement pas en charge le chargement différé de propriétés scalaires ou complexes. Toutefois, dans les cas où vous avez une table qui comprend un objet volumineux, tel qu'un objet BLOB, vous pouvez utiliser le fractionnement de table pour séparer les grandes propriétés dans une entité distincte. Supposons, par exemple, que vous disposiez d'une table Product qui comprend une colonne de photos varbinary. Si vous n'avez pas besoin d'accéder fréquemment à cette propriété dans vos requêtes, vous pouvez utiliser le fractionnement de table pour importer uniquement les parties de l'entité dont vous avez normalement besoin. L'entité représentant la photo de produit est chargée uniquement lorsque vous en avez besoin de manière explicite.

Une bonne ressource qui montre comment activer le fractionnement de table est le billet de blog « Table fractionnement dans Entity Framework » de Gil Fink :

<<http://blogs.microsoft.co.il/blogs/gilf/archive/2009/10/13/table-splitting-in-entity-framework.aspx>>.

## 9 autres considérations

### **Garbage collection de serveur 9,1**

Certains utilisateurs peuvent rencontrer des conflits de ressources qui limitent le parallélisme qu'ils attendent quand le garbage collector n'est pas correctement configuré. Chaque fois que EF est utilisé dans un scénario multithread ou dans une application qui ressemble à un système côté serveur, veillez à activer le garbage collection de serveur. Pour cela, vous avez besoin d'un paramètre simple dans votre fichier de configuration d'application :

```

<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <runtime>
        <gcServer enabled="true" />
    </runtime>
</configuration>

```

Cela devrait réduire la contention de vos threads et augmenter votre débit de 30% dans les scénarios de saturation de l'UC. En général, vous devez toujours tester le comportement de votre application à l'aide du garbage collection

classique (qui est mieux adapté aux scénarios d'interface utilisateur et côté client), ainsi qu'au garbage collection de serveur.

## 9,2 AutoDetectChanges

Comme mentionné précédemment, Entity Framework peut présenter des problèmes de performances lorsque le cache d'objets a de nombreuses entités. Certaines opérations, telles que Add, Remove, Find, Entry et SaveChanges, déclenchent des appels à DetectChanges qui peuvent consommer une grande quantité d'UC en fonction de la taille du cache d'objets. Cela est dû au fait que le cache d'objets et le gestionnaire d'état d'objet essaient de rester synchronisés autant que possible sur chaque opération effectuée dans un contexte afin que les données produites soient correctes dans un vaste éventail de scénarios.

Il est généralement recommandé de conserver la détection automatique des modifications de Entity Framework activée pendant toute la durée de vie de votre application. Si votre scénario est affecté négativement par une utilisation intensive du processeur et que vos profils indiquent que le coupable est l'appel à DetectChanges, envisagez de désactiver temporairement AutoDetectChanges dans la partie sensible de votre code :

```
try
{
    context.Configuration.AutoDetectChangesEnabled = false;
    var product = context.Products.Find(productId);
    ...
}
finally
{
    context.Configuration.AutoDetectChangesEnabled = true;
}
```

Avant de désactiver AutoDetectChanges, il est judicieux de comprendre qu'il est possible que Entity Framework perde sa capacité à effectuer le suivi de certaines informations sur les modifications qui ont lieu sur les entités. En cas de gestion incorrecte, cela peut entraîner une incohérence des données sur votre application. Pour plus d'informations sur la désactivation de AutoDetectChanges lire <<http://blog.oneunicorn.com/2012/03/12/secrets-of-detectchanges-part-3-switching-off-automatic-detectchanges/>>.

## 9,3 contexte par requête

Les contextes de Entity Framework sont destinés à être utilisés en tant qu'instances courtes afin de fournir une expérience de performances optimale. Les contextes sont supposés être à courte durée de vie et sont ignorés, et en tant que tels ont été implémentés pour être très légers et réutiliser les métadonnées chaque fois que cela est possible. Dans les scénarios Web, il est important de garder cela à l'esprit et de ne pas avoir de contexte plus que la durée d'une requête unique. De même, dans les scénarios non Web, le contexte doit être ignoré en fonction de votre compréhension des différents niveaux de mise en cache dans le Entity Framework. En règle générale, il est préférable d'éviter d'avoir une instance de contexte tout au long de la vie de l'application, ainsi que des contextes par thread et des contextes statiques.

## sémantique null de la base de données 9,4

Entity Framework par défaut génère du code SQL dont la sémantique de comparaison est C# null. Prenons l'exemple de requête suivant :

```

int? categoryId = 7;
int? supplierId = 8;
decimal? unitPrice = 0;
short? unitsInStock = 100;
short? unitsOnOrder = 20;
short? reorderLevel = null;

var q = from p in context.Products
        where p.Category.CategoryName == "Beverages"
            || (p.CategoryID == categoryId)
            || p.SupplierID == supplierId
            || p.UnitPrice == unitPrice
            || p.UnitsInStock == unitsInStock
            || p.UnitsOnOrder == unitsOnOrder
            || p.ReorderLevel == reorderLevel
    select p;

var r = q.ToList();

```

Dans cet exemple, nous comparons un certain nombre de variables Nullable à des propriétés Nullable sur l'entité, telles que RéfFournisseur et UnitPrice. Le SQL généré pour cette requête demande si la valeur du paramètre est la même que la valeur de la colonne, ou si les valeurs du paramètre et de la colonne sont null. Cela permet de masquer la manière dont le serveur de base de données gère les valeurs NULL et fournit une expérience de type C# cohérente sur les différents fournisseurs de bases de données. En revanche, le code généré est un peu compliqué et peut ne pas s'exécuter correctement lorsque le nombre de comparaisons dans l'instruction WHERE de la requête augmente jusqu'à un grand nombre.

Une façon de traiter cette situation consiste à utiliser la sémantique null de la base de données. Notez que cela peut potentiellement se comporter différemment de la sémantique C# null dans la mesure où Entity Framework générera désormais un SQL plus simple qui expose la manière dont le moteur de base de données gère les valeurs NULL. La sémantique null de la base de données peut être activée par contexte avec une seule ligne de configuration par rapport à la configuration du contexte :

```
context.Configuration.UseDatabaseNullSemantics = true;
```

Les requêtes de taille petite à moyenne n'affichent pas une amélioration notable des performances lors de l'utilisation de la sémantique null de la base de données, mais la différence est plus perceptible sur les requêtes avec un grand nombre de comparaisons de valeurs NULL potentielles.

Dans l'exemple de requête ci-dessus, la différence de performances était inférieure à 2% dans un microtest exécuté dans un environnement contrôlé.

## 9.5 Async

Entity Framework 6 a introduit la prise en charge des opérations asynchrones lors de l'exécution sur .NET 4.5 ou une version ultérieure. Pour l'essentiel, les applications qui ont une contention liée à l'e/s tireront le meilleur parti de l'utilisation des opérations asynchrones de requête et d'enregistrement. Si votre application ne souffre pas d'une contention d'e/s, l'utilisation de Async va, dans les meilleurs cas, s'exécuter de façon synchrone et retourner le résultat dans le même laps de temps qu'un appel synchrone, ou dans le pire des cas, différer l'exécution à une tâche asynchrone et ajouter un Tim supplémentaire e à la fin de votre scénario.

Pour plus d'informations sur le travail de programmation asynchrone qui aide à vous de décider si async améliorera les performances de votre application visitez <http://msdn.microsoft.com/library/hh191443.aspx>. Pour plus d'informations sur l'utilisation des opérations asynchrones sur Entity Framework, consultez [requête Async et enregistrement](#).

## 9.6 NGEN

Entity Framework 6 ne figure pas dans l'installation par défaut de .NET Framework. Par conséquent, les assemblies

Entity Framework ne sont pas NGEN par défaut, ce qui signifie que tous les Entity Framework code sont soumis aux mêmes coûts JIT'ing que tout autre assembly MSIL. Cela peut dégrader l'expérience F5 lors du développement et du démarrage à froid de votre application dans les environnements de production. Afin de réduire les coûts liés au processeur et à la mémoire de JIT'ing, il est recommandé de NGEN les images de Entity Framework, le cas échéant. Pour plus d'informations sur la façon d'améliorer les performances de démarrage de Entity Framework 6 avec NGEN, consultez [amélioration des performances de démarrage avec NGen](#).

## 9,7 Code First versus EDMX

Entity Framework raisons du problème d'incompatibilité d'impédance entre la programmation orientée objet et les bases de données relationnelles en utilisant une représentation en mémoire du modèle conceptuel (les objets), le schéma de stockage (la base de données) et un mappage entre directionnelle. Ces métadonnées sont appelées Entity Data Model ou EDM pour Short. À partir de ce modèle EDM, Entity Framework dérivera les vues pour aller-retour des données des objets en mémoire vers la base de données et inversement.

Lorsque Entity Framework est utilisé avec un fichier EDMX qui spécifie formellement le modèle conceptuel, le schéma de stockage et le mappage, alors l'étape de chargement du modèle doit uniquement valider que l'EDM est correct (par exemple, vérifier qu'aucun mappage n'est manquant), puis Générez les vues, puis validez les vues et préparez ces métadonnées pour une utilisation. Seule une requête peut être exécutée ou de nouvelles données doivent être enregistrées dans le magasin de données.

L'approche Code First est, au cœur, un générateur de Entity Data Model sophistiqué. Le Entity Framework doit générer un modèle EDM à partir du code fourni ; pour ce faire, il analyse les classes impliquées dans le modèle, en appliquant des conventions et en configurant le modèle via l'API Fluent. Une fois le modèle EDM généré, le Entity Framework se comporte essentiellement de la même façon qu'un fichier EDMX était présent dans le projet. Par conséquent, la génération du modèle à partir de Code First ajoute une complexité supplémentaire qui se traduit par un temps de démarrage plus lent pour le Entity Framework par rapport à un EDMX. Le coût dépend entièrement de la taille et de la complexité du modèle en cours de génération.

Lorsque vous choisissez d'utiliser EDMX et Code First, il est important de savoir que la flexibilité introduite par Code First augmente le coût de la création du modèle pour la première fois. Si votre application peut résister au coût de cette première charge, il est généralement Code First est la meilleure façon de procéder.

# 10 examen des performances

## 10,1 utilisation du profileur Visual Studio

Si vous rencontrez des problèmes de performances avec la Entity Framework, vous pouvez utiliser un profileur comme celui qui est intégré à Visual Studio pour voir où votre application consacre son temps. Ceci est l'outil que nous avons utilisé pour générer les graphiques à secteurs dans le billet de blog « Exploring the Performance d'ADO.NET Entity Framework - partie 1 » (<http://blogs.msdn.com/b/adonet/archive/2008/02/04/exploring-the-performance-of-the-ado-net-entity-framework-part-1.aspx>) qui indiquent où Entity Framework consacre son temps pendant les requêtes à froid et à chaudes.

Le billet de blog « profilage Entity Framework à l'aide du profileur Visual Studio 2010 » écrit par l'équipe de Conseil clients de données et de modélisation illustre un exemple concret de la façon dont il a utilisé le profileur pour examiner un problème de performances. <http://blogs.msdn.com/b/dmcat/archive/2010/04/30/profiling-entity-framework-using-the-visual-studio-2010-profiler.aspx>. Ce billet a été écrit pour une application Windows. Si vous devez profiler une application Web, l'enregistreur de performances Windows (WPR) et les outils de l'analyseur de performances Windows (WPA) peuvent fonctionner mieux que si vous utilisez Visual Studio. WPR et WPA font partie de la boîte à outils de performances Windows, qui est inclus avec le Kit de déploiement et d'évaluation de Windows (<http://www.microsoft.com/download/details.aspx?id=39982>).

## profilage de base de données/application 10,2

Les outils tels que le profileur intégré à Visual Studio vous indiquent où votre application consacre du temps. Un autre type de profileur est disponible pour effectuer une analyse dynamique de votre application en cours

d'exécution, soit en production, soit en préproduction en fonction des besoins, et recherche des pièges et des anti-modèles courants d'accès aux bases de données.

Deux profileurs de code disponibles dans le commerce sont le Profiler de Framework d'entité (<http://efprof.com>) et ORMPProfiler (<http://ormprofiler.com>).

Si votre application est une application MVC utilisant Code First, vous pouvez utiliser mini de StackExchange.

Scott Hanselman décrit cet outil dans son blog à l'adresse :

<http://www.hanselman.com/blog/NuGetPackageOfTheWeek9ASPNETMiniProfilerFromStackExchangeRocksYourWorld.aspx>.

Pour plus d'informations sur le profilage de l'activité de la base de données de votre application, consultez l'article du magazine MSDN de Julie Lerman intitulé [profilage de l'activité des bases de données dans le Entity Framework](#).

### **enregistreur d'événements de base de données 10,3**

Si vous utilisez Entity Framework 6, pensez également à utiliser la fonctionnalité de journalisation intégrée. La propriété de base de données du contexte peut être invitée à consigner son activité via une simple configuration à une seule ligne :

```
using (var context = newQueryComparison.DbC.NorthwindEntities())
{
    context.Database.Log = Console.WriteLine;
    var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
    q.ToList();
}
```

Dans cet exemple, l'activité de base de données est enregistrée dans la console, mais la propriété log peut être configurée pour appeler n'importe quelle action <chaîne> déléguée.

Si vous souhaitez activer la journalisation de base de données sans recompilation, et que vous utilisez Entity Framework 6,1 ou une version ultérieure, vous pouvez le faire en ajoutant un intercepteur dans le fichier Web.config ou App.config de votre application.

```
<interceptors>
    <interceptor type="System.Data.Entity.Infrastructure.Interception.DatabaseLogger, EntityFramework">
        <parameters>
            <parameter value="C:\Path\To\My\LogOutput.txt"/>
        </parameters>
    </interceptor>
</interceptors>
```

Pour plus d'informations sur la façon d'ajouter la journalisation sans avoir à recompiler aller à <http://blog.oneunicorn.com/2014/02/09/ef-6-1-turning-on-logging-without-recompiling/>.

## **11 Annexe**

### **11,1 A. environnement de test**

Cet environnement utilise une configuration de 2 machines avec la base de données sur un ordinateur distinct de l'application cliente. Les machines se trouvent dans le même rack, de sorte que la latence du réseau est relativement faible, mais plus réaliste qu'un environnement à un seul ordinateur.

#### **11.1.1 serveur d'applications**

Environnement logiciel 11.1.1.1

- Entity Framework 4 environnement logiciel
  - Nom du système d'exploitation : Windows Server 2008 R2 Entreprise SP1.

- Visual Studio 2010 – Ultimate.
- Visual Studio 2010 SP1 (uniquement pour certaines comparaisons).
- Entity Framework un environnement logiciel de 5 et 6
  - Nom du système d'exploitation : Windows 8.1 Enterprise
  - Visual Studio 2013 – Ultimate.

#### Environnement matériel 11.1.1.2

- Processeur double : Intel (R) Xeon (R) CPU L5520 W3530 @ 2,27 GHz, 2261 Mhz8 GHz, 4 cœurs, 84 processeur logique (s).
- 2412 Go RamRAM.
- lecteur 136 Go SCSI250GB SATA 7200 RPM 3 Go/s divisé en 4 partitions.

#### serveur 11.1.2 DB

##### Environnement logiciel 11.1.2.1

- Nom du système d'exploitation : Windows Server 2008 R 28.1 Enterprise SP1.
- SQL Server 2008 R22012.

##### Environnement matériel 11.1.2.2

- Processeur unique : Intel (R) Xeon (R) CPU L5520 @ 2,27 GHz, 2261 MhzES-1620 0 @ 3,60 GHz, 4 cœurs (s), 8 processeurs logiques.
- 824 Go RamRAM.
- lecteur 465 Go ATA500GB SATA 7200 RPM 6 Go/s divisé en 4 partitions.

## 11.2 B. tests de comparaison des performances des requêtes

Le modèle Northwind a été utilisé pour exécuter ces tests. Il a été généré à partir de la base de données à l'aide du concepteur de Entity Framework. Ensuite, le code suivant a été utilisé pour comparer les performances des options d'exécution de la requête :

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Common;
using System.Data.Entity.Infrastructure;
using System.Data.EntityClient;
using System.Data.Objects;
using System.Linq;

namespace QueryComparison
{
    public partial class NorthwindEntities : ObjectContext
    {
        private static readonly Func<NorthwindEntities, string, IQueryable<Product>> productsForCategoryCQ =
            CompiledQuery.Compile(
                (NorthwindEntities context, string categoryName) =>
                    context.Products.Where(p => p.Category.CategoryName == categoryName)
            );

        public IQueryable<Product> InvokeProductsForCategoryCQ(string categoryName)
        {
            return productsForCategoryCQ(this, categoryName);
        }
    }

    public class QueryTypePerfComparison
    {
        private static string entityConnectionString =
            @"metadata=res://*/Northwind.csdl|res://*/Northwind.ssdl|res://*/Northwind.msl;provider=System.Data.SqlClient;
provider connection string='data source=.;initial catalog=Northwind;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework"";

        public void LINQIncludingContextCreation()
        {

```

```

    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
            q.ToList();
        }
    }

    public void LINQNoTracking()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            context.Products.MergeOption = MergeOption.NoTracking;

            var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
            q.ToList();
        }
    }

    public void CompiledQuery()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            var q = context.InvokeProductsForCategoryCQ("Beverages");
            q.ToList();
        }
    }

    public void ObjectQuery()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            ObjectQuery<Product> products = context.Products.Where("it.Category.CategoryName =
'Beverages'");
            products.ToList();
        }
    }

    public void EntityCommand()
    {
        using (EntityConnection eConn = new EntityConnection(entityConnectionString))
        {
            eConn.Open();
            EntityCommand cmd = eConn.CreateCommand();
            cmd.CommandText = "Select p From NorthwindEntities.Products As p Where p.Category.CategoryName =
'Beverages'";

            using (EntityDataReader reader = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
            {
                List<Product> productsList = new List<Product>();
                while (reader.Read())
                {
                    DbDataRecord record = (DbDataRecord)reader.GetValue(0);

                    // 'materialize' the product by accessing each field and value. Because we are
materializing products, we won't have any nested data readers or records.
                    int fieldCount = record.FieldCount;

                    // Treat all products as Product, even if they are the subtype DiscontinuedProduct.
                    Product product = new Product();

                    product.ProductID = record.GetInt32(0);
                    product.ProductName = record.GetString(1);
                    product.SupplierID = record.GetInt32(2);
                    product.CategoryID = record.GetInt32(3);
                    product.QuantityPerUnit = record.GetString(4);
                    product.UnitPrice = record.GetDecimal(5);
                    product.UnitsInStock = record.GetInt16(6);
                    product.UnitsOnOrder = record.GetInt16(7);
                }
            }
        }
    }
}

```

```

        product.ReorderLevel = record.GetInt16(8);
        product.Discontinued = record.GetBoolean(9);

        productsList.Add(product);
    }
}
}

public void ExecuteStoreQuery()
{
    using (NorthwindEntities context = new NorthwindEntities())
    {
        ObjectResult<Product> beverages = context.ExecuteStoreQuery<Product>(
@"
    SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued
    FROM        Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
    WHERE       (C.CategoryName = 'Beverages')"
);
        beverages.ToList();
    }
}

public void ExecuteStoreQueryDbContext()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var beverages = context.Database.SqlQuery<QueryComparison.DbC.Product>(
@"
    SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued
    FROM        Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
    WHERE       (C.CategoryName = 'Beverages')"
);
        beverages.ToList();
    }
}

public void ExecuteStoreQueryDbSet()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var beverages = context.Products.SqlQuery(
@"
    SELECT      P.ProductID, P.ProductName, P.SupplierID, P.CategoryID, P.QuantityPerUnit, P.UnitPrice,
P.UnitsInStock, P.UnitsOnOrder, P.ReorderLevel, P.Discontinued
    FROM        Products AS P INNER JOIN Categories AS C ON P.CategoryID = C.CategoryID
    WHERE       (C.CategoryName = 'Beverages')"
);
        beverages.ToList();
    }
}

public void LINQIncludingContextCreationDbContext()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var q = context.Products.Where(p => p.Category.CategoryName == "Beverages");
        q.ToList();
    }
}

public void LINQNoTrackingDbContext()
{
    using (var context = new QueryComparison.DbC.NorthwindEntities())
    {
        var q = context.Products.AsNoTracking().Where(p => p.Category.CategoryName == "Beverages");
        q.ToList();
    }
}
}

```

```
}
```

### 11.3 C. modèle Navision

La base de données Navision est une base de données volumineuse utilisée pour la démonstration de Microsoft Dynamics-NAV. Le modèle conceptuel généré contient les jeux d'entités 1005 et les ensembles d'associations 4227. Le modèle utilisé dans le test est « Flat », aucun héritage n'a été ajouté à ce dernier.

#### requêtes 11.3.1 utilisées pour les tests Navision

La liste de requêtes utilisée avec le modèle Navision contient 3 catégories de Entity SQL requêtes :

recherche 11.3.1.1

Requête de recherche simple sans agrégations

- Nombre : 16232
- Exemple :

```
<Query complexity="Lookup">
  <CommandText>Select value distinct top(4) e.Idle_Time From NavisionFKContext.Session as e</CommandText>
</Query>
```

11.3.1.2 SingleAggregating

Requête BI normale avec plusieurs agrégations, mais aucun sous-total (requête unique)

- Nombre : 2313
- Exemple :

```
<Query complexity="SingleAggregating">
  <CommandText>NavisionFK.MDF_SessionLogin_Time_Max()</CommandText>
</Query>
```

Où MDF\_SessionLogin\_Time\_Max () est défini dans le modèle comme suit :

```
<Function Name="MDF_SessionLogin_Time_Max" ReturnType="Collection(DateTime)">
  <DefiningExpression>SELECT VALUE Edm.Min(E.Login_Time) FROM NavisionFKContext.Session as E</DefiningExpression>
</Function>
```

11.3.1.3 AggregatingSubtotals

Une requête BI avec des agrégations et des sous-totaux (par le biais de l'instruction Union All)

- Nombre : 178
- Exemple :

```

<Query complexity="AggregatingSubtotals">
  <CommandText>
using NavisionFK;
function AmountConsumed(entities Collection([CRONUS_International_Ltd__Zone])) as
(
  Edm.Sum(select value N.Block_Movement FROM entities as E, E.CRONUS_International_Ltd__Bin as N)
)
function AmountConsumed(P1 Edm.Int32) as
(
  AmountConsumed(select value e from NavisionFKContext.CRONUS_International_Ltd__Zone as e where
e.Zone_Ranking = P1)
)
-----
-----
(
  select top(10) Zone_Ranking, Cross_Dock_Bin_Zone, AmountConsumed(GroupPartition(E))
  from NavisionFKContext.CRONUS_International_Ltd__Zone as E
  where AmountConsumed(E.Zone_Ranking) > @MinAmountConsumed
  group by E.Zone_Ranking, E.Cross_Dock_Bin_Zone
)
union all
(
  select top(10) Zone_Ranking, Cast(null as Edm.Byte) as P2, AmountConsumed(GroupPartition(E))
  from NavisionFKContext.CRONUS_International_Ltd__Zone as E
  where AmountConsumed(E.Zone_Ranking) > @MinAmountConsumed
  group by E.Zone_Ranking
)
union all
{
  Row(Cast(null as Edm.Int32) as P1, Cast(null as Edm.Byte) as P2, AmountConsumed(select value E
                                         from
NavisionFKContext.CRONUS_International_Ltd__Zone as E
                                         where AmountConsumed(E.Zone_Ranking)
> @MinAmountConsumed))
}</CommandText>
<Parameters>
  <Parameter Name="MinAmountConsumed" DbType="Int32" Value="10000" />
</Parameters>
</Query>

```

# Amélioration des performances de démarrage avec NGen

17/10/2019 • 10 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement :** Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Le .NET Framework prend en charge la génération d'images natives pour les applications et les bibliothèques managées afin d'aider les applications à démarrer plus rapidement et, dans certains cas, à utiliser moins de mémoire. Les images natives sont créées en traduisant les assemblies de code managé en fichiers contenant des instructions machine natives avant l'exécution de l'application, ce qui allège le compilateur .NET JIT (juste-à-temps) d'avoir à générer les instructions natives à l'adresse exécution de l'application.

Avant la version 6, les bibliothèques principales du runtime EF faisaient partie du .NET Framework et les images natives ont été générées automatiquement. Depuis la version 6, l'intégralité du runtime EF a été combinée dans le package NuGet EntityFramework. Les images natives doivent maintenant être générées à l'aide de l'outil en ligne de commande NGen. exe pour obtenir des résultats similaires.

Les observations empiriques montrent que les images natives des assemblies du runtime EF peuvent réduire de 1 à 3 secondes le temps de démarrage de l'application.

## Comment utiliser NGen. exe

La fonction la plus basique de l'outil NGen. exe consiste à « installer » (autrement dit, pour créer et rendre persistant sur disque) des images natives pour un assembly et toutes ses dépendances directes. Voici comment y parvenir :

1. Ouvrez une fenêtre d'invite de commandes en tant qu'administrateur.
2. Remplacez le répertoire de travail actuel par l'emplacement des assemblies pour lesquels vous souhaitez générer des images natives :

```
cd <*Assemblies location*>
```

3. En fonction de votre système d'exploitation et de la configuration de l'application, vous devrez peut-être générer des images natives pour l'architecture 32 bits, l'architecture 64 bits ou pour les deux.

Pour une exécution de 32 bits :

```
%WINDIR%\Microsoft.NET\Framework\v4.0.30319\ngen install <Assembly name>
```

Pour une exécution de 64 bits :

```
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\ngen install <Assembly name>
```

#### TIP

La génération d'images natives pour une architecture incorrecte est une erreur très courante. En cas de doute, vous pouvez simplement générer des images natives pour toutes les architectures qui s'appliquent au système d'exploitation installé sur l'ordinateur.

NGen. exe prend également en charge d'autres fonctions, telles que la désinstallation et l'affichage des images natives installées, la mise en file d'attente de la génération de plusieurs images, etc. Pour plus d'informations sur l'utilisation, consultez la [documentation de Ngen. exe](#).

## Quand utiliser NGen. exe

Lorsqu'il s'agit de déterminer les assemblies pour lesquels générer des images natives dans une application basée sur EF version 6 ou ultérieure, vous devez prendre en compte les options suivantes :

- **L'assembly principal du runtime EF, EntityFramework. dll:** une application EF classique exécute une quantité significative de code à partir de cet assembly au démarrage ou lors de son premier accès à la base de données. Par conséquent, la création d'images natives de cet assembly produit les plus grandes améliorations des performances de démarrage.
- **Tout assembly de fournisseur EF utilisé par votre application :** le temps de démarrage peut également tirer parti de la génération d'images natives. Par exemple, si l'application utilise le fournisseur EF pour SQL Server vous souhaiterez générer une image native pour EntityFramework.SqlServer. dll.
- **Les assemblies et autres dépendances de votre application:** la [documentation de Ngen. exe](#) couvre les critères généraux permettant de choisir les assemblies pour lesquels générer des images natives et l'impact des images natives sur la sécurité, des options avancées telles que «Hard Binding», des scénarios tels que l'utilisation d'images natives dans des scénarios de débogage et de profilage, etc.

#### TIP

Veillez à mesurer avec soin l'impact de l'utilisation d'images natives sur les performances de démarrage et les performances globales de votre application, et comparez-les aux exigences réelles. Bien que les images natives permettent généralement d'améliorer les performances de démarrage et, dans certains cas, de réduire l'utilisation de la mémoire, tous les scénarios ne sont pas avantageux. Par exemple, lors d'une exécution stable de l'État (autrement dit, une fois que toutes les méthodes utilisées par l'application ont été appelées au moins une fois), le code généré par le compilateur JIT peut, en fait, produire des performances légèrement meilleures que les images natives.

## Utilisation de NGen. exe sur un ordinateur de développement

Pendant le développement, le compilateur JIT .NET offrira le meilleur compromis global pour le code qui change fréquemment. La génération d'images natives pour les dépendances compilées telles que les assemblies du runtime EF peut accélérer le développement et le test en coupant quelques secondes au début de chaque exécution.

L'emplacement du package NuGet pour la solution est un bon emplacement pour rechercher les assemblies du runtime EF. Par exemple, pour une application utilisant EF 6.0.2 avec SQL Server et ciblant .NET 4,5 ou une version ultérieure, vous pouvez taper la commande suivante dans une fenêtre d'invite de commandes (n'oubliez pas de l'ouvrir en tant qu'administrateur) :

```
cd <Solution directory>\packages\EntityFramework.6.0.2\lib\net45  
%WINDIR%\Microsoft.NET\Framework\v4.0.30319\ngen install EntityFramework.SqlServer.dll  
%WINDIR%\Microsoft.NET\Framework64\v4.0.30319\ngen install EntityFramework.SqlServer.dll
```

#### **NOTE**

Cela tire parti du fait que l'installation des images natives pour le fournisseur EF pour SQL Server entraîne également l'installation par défaut des images natives pour l'assembly de runtime EF principal. Cela fonctionne, car NGen. exe peut détecter que l'EntityFramework. dll est une dépendance directe de l'assembly EntityFramework. SqlServer. dll situé dans le même répertoire.

## Création d'images natives pendant l'installation

WiX Toolkit prend en charge la mise en file d'attente de la génération d'images natives pour les assemblies managés au cours de l'installation, comme expliqué dans ce [Guide de procédure](#). Une autre solution consiste à créer une tâche d'installation personnalisée qui exécute la commande NGen. exe.

## Vérification de l'utilisation des images natives pour EF

Vous pouvez vérifier qu'une application spécifique utilise un assembly natif en recherchant les assemblies chargés portant l'extension « .ni. dll » ou « .ni. exe ». Par exemple, une image native de l'assembly de Runtime principal d'EF sera appelée EntityFramework. ni. dll. Un moyen simple d'inspecter les assemblies .NET chargés d'un processus consiste à utiliser [Process Explorer](#).

## Autres éléments à connaître

La **création d'une image native d'un assembly ne doit pas être confondue avec l'inscription de l'assembly dans le GAC (global assembly cache)** . NGen. exe permet de créer des images d'assemblies qui ne se trouvent pas dans le GAC, et en fait, plusieurs applications qui utilisent une version spécifique d'EF peuvent partager la même image native. Bien que Windows 8 puisse créer automatiquement des images natives pour les assemblies placés dans le GAC, l'exécution d'EF est optimisée pour être déployée en même temps que votre application. nous vous déconseillons de l'inscrire dans le GAC, car cela a un impact négatif sur la résolution de l'assembly et maintenance de vos applications entre autres aspects.

# Vues pré générées mappage

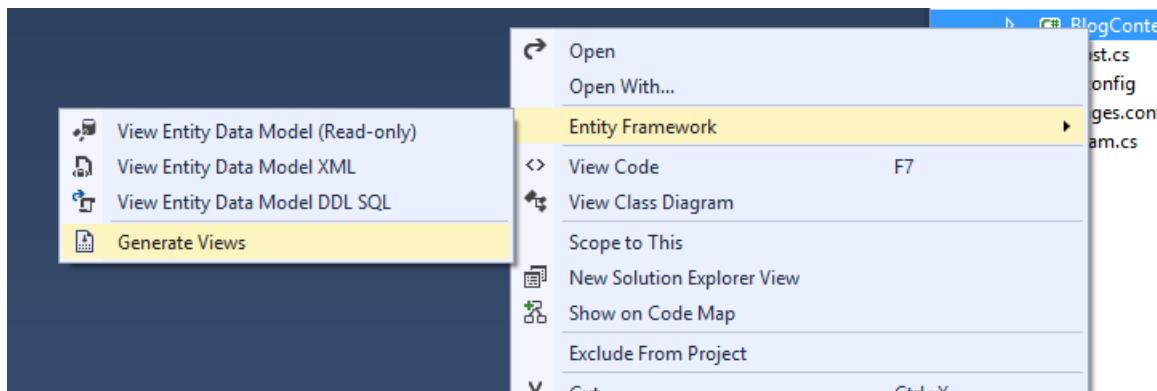
01/10/2018 • 8 minutes to read

Avant de l'Entity Framework peut exécuter une requête ou enregistrer les modifications apportées à la source de données, il doit générer un ensemble de vues de mappage pour accéder à la base de données. Ces vues de mappage sont un ensemble de l'instruction Entity SQL qui représentent la base de données de manière abstraite et font partie des métadonnées qui sont mis en cache par domaine d'application. Si vous créez plusieurs instances du même contexte dans le même domaine d'application, elles réutiliseront les vues de mappage depuis les métadonnées mises en cache au lieu de les régénérer. Étant donné que la génération de vues de mappage constitue une partie significative du coût total de l'exécution de la première requête, Entity Framework vous permet de pré-générer des vues de mappage et de les inclure dans le projet compilé. Pour plus d'informations, consultez [considérations sur les performances \(Entity Framework\)](#).

## Génération de mappage des vues avec l'édition Community EF Power Tools

Pour pré-générer des vues le plus simple consiste à utiliser le [EF Power Tools Community Edition](#). Une fois que vous avez installé les outils Power avoir une option de menu pour générer des vues, comme indiqué ci-dessous.

- Pour **Code First** modèles avec le bouton droit sur le fichier de code qui contient votre classe DbContext.
- Pour **Entity Framework Designer** modèles avec le bouton droit sur votre fichier EDMX.



Une fois le processus est terminé vous offrira une classe similaire à ce qui suit générée

The screenshot shows the Visual Studio interface. On the left, the Solution Explorer displays a project named 'BlogApp' with files like 'Blog.cs', 'BlogContext.cs', and 'BlogContext.Views.cs'. The 'BlogContext.Views.cs' file is currently selected and open in the code editor. The code editor shows the following C# code:

```
1 //-----  
2 // <auto-generated>  
3 //   This code was generated by a tool.  
4 //  
5 //   Changes to this file may cause incorrect behavior and will be lost if  
6 //   the code is regenerated.  
7 // </auto-generated>  
8 //-----  
9  
10 using System.Data.Entity.Infrastructure.MappingViews;  
11  
12 [assembly: DbMappingViewCacheTypeAttribute(  
13     typeof(BlogApp.Models.BlogContext),  
14     typeof(Edm_EntityMappingGeneratedViews.ViewsForBaseEntitySetsa0b843f03dd29abee99789e190a6fb70ce8e93dc97945d437d9a58fb8e2afde))]  
15  
16 namespace Edm_EntityMappingGeneratedViews  
17 {  
18     using System;  
19     using System.CodeDom.Compiler;  
20     using System.Data.Entity.Core.Metadata.Edm;  
21  
22     /// <summary>  
23     /// Implements a mapping view cache.  
24     /// </summary>  
25     [GeneratedCode("Entity Framework Power Tools", "0.9.0.0")]  
26     internal sealed class ViewsForBaseEntitySetsa0b843f03dd29abee99789e190a6fb70ce8e93dc97945d437d9a58fb8e2afde : DbMappingViewCache  
27     {  
28         /// <summary>  
29         /// Gets a hash value computed over the mapping closure.  
30         /// </summary>  
31     }  
32 }
```

Désormais, lorsque vous exécutez votre application EF utilise cette classe pour charger des vues en fonction des

besoins. Si votre modèle change et vous ne ré-générez pas de cette classe EF lève une exception.

## Génération de mappage des vues à partir du Code - EF6 et versions ultérieures

Les autres pour générer des vues consiste à utiliser les API Entity Framework fournit. Lorsque vous utilisez cette méthode que vous avez la possibilité de sérialiser les vues mais vous le souhaitez, mais vous devez également charger les vues vous-même.

### NOTE

**EF6 et versions ultérieures uniquement** -les API présentés dans cette section ont été introduits dans Entity Framework 6. Si vous utilisez une version antérieure, que ces informations ne s'applique pas.

### Génération de vues

Les API pour générer des vues se trouvent sur la classe System.Data.Entity.Core.Mapping.StorageMappingItemCollection. Vous pouvez récupérer une StorageMappingCollection pour un contexte à l'aide d'un ObjectContext MetadataWorkspace. Si vous utilisez la nouvelle API DbContext, puis vous pouvez y accéder à l'aide de la IObjectContextAdapter comme ci-dessous, dans ce code, nous avons une instance de votre DbContext dérivée appelée dbContext :

```
var objectContext = ((IObjectContextAdapter) dbContext).ObjectContext;
var mappingCollection = (StorageMappingItemCollection) objectContext.MetadataWorkspace
    .GetItemCollection(DataSpace.CSSpace);
```

Une fois que vous avez StorageMappingItemCollection puis vous pouvez accéder aux méthodes GenerateViews et ComputeMappingHashValue.

```
public Dictionary<EntitySetBase, DbMappingView> GenerateViews(IList<EdmSchemaError> errors)
public string ComputeMappingHashValue()
```

La première méthode crée un dictionnaire avec une entrée pour chaque vue dans le mappage de conteneur. La deuxième méthode calcule une valeur de hachage pour le mappage de conteneur unique et est utilisée lors de l'exécution pour valider que le modèle n'a pas changé dans la mesure où les vues ont été préalablement générées. Remplacements des deux méthodes sont fournies pour les scénarios complexes impliquant plusieurs mappages de conteneur.

Lors de la génération de vues vous appelez la méthode GenerateViews, puis écrire EntitySetBase et DbMappingView résultats. Vous devez également stocker le hachage généré par la méthode ComputeMappingHashValue.

### Chargement des vues

Pour charger les vues générées par la méthode GenerateViews, vous pouvez fournir EF avec une classe qui hérite de la classe abstraite DbMappingViewCache. DbMappingViewCache spécifie deux méthodes que vous devez implémenter :

```
public abstract string MappingHashValue { get; }
public abstract DbMappingView GetView(EntitySetBase extent);
```

La propriété MappingHashValue doit retourner le hachage généré par la méthode ComputeMappingHashValue. Lorsque EF est accédant à poser pour les vues, il sera tout d'abord générer et comparer la valeur de hachage du modèle avec le hachage retourné par cette propriété. Si elles ne correspondent pas EF lève une exception

## EntityCommandCompilationException.

La méthode GetView acceptera une EntitySetBase et vous devez renvoyer un DbMappingView contenant EntitySql qui a été généré pour qui a été associé à la EntitySetBase donné dans le dictionnaire généré par la méthode GenerateViews. Si la demande à EF pour une vue que vous n'avez pas puis GetView doit retourner null.

Voici un extrait de DbMappingViewCache qui est généré avec les outils Power Tools, comme décrit ci-dessus, que nous voyons comment stocker et récupérer le EntitySql requis.

```
public override string MappingHashValue
{
    get { return "a0b843f03dd29abee99789e190a6fb70ce8e93dc97945d437d9a58fb8e2af2e"; }
}

public override DbMappingView GetView(EntitySetBase extent)
{
    if (extent == null)
    {
        throw new ArgumentNullException("extent");
    }

    var extentName = extent.EntityContainer.Name + "." + extent.Name;

    if (extentName == "BlogContext.Blogs")
    {
        return GetView2();
    }

    if (extentName == "BlogContext.Posts")
    {
        return GetView3();
    }

    return null;
}

private static DbMappingView GetView2()
{
    return new DbMappingView(@"
        SELECT VALUE -- Constructing Blogs
        [BlogApp.Models.Blog](T1.Blog_BlogId, T1.Blog_Test, T1.Blog_title, T1.Blog_Active,
        T1.Blog_SomeDecimal)
        FROM (
        SELECT
            T.BlogId AS Blog_BlogId,
            T.Test AS Blog_Test,
            T.title AS Blog_title,
            T.Active AS Blog_Active,
            T.SomeDecimal AS Blog_SomeDecimal,
            True AS _from0
        FROM CodeFirstDatabase.Blog AS T
        ) AS T1");
}
```

Pour que l'utilisation d'EF votre DbMappingViewCache vous ajouter utilisent le DbMappingViewCacheTypeAttribute, en spécifiant le contexte dans lequel il a été créé pour. Dans le code ci-dessous, nous associons la BlogContext avec la classe MyMapViewCache.

```
[assembly: DbMappingViewCacheType(typeof(BlogContext), typeof(MyMapViewCache))]
```

Pour des scénarios plus complexes, les instances de cache de vue de mappage peuvent être fournies en spécifiant une fabrique de cache de vue de mappage. Cela est possible en implémentant la classe abstraite

System.Data.Entity.Infrastructure.MappingViews.DbMappingViewCacheFactory. L'instance de la fabrique de cache de vue de mappage qui est utilisée peut être récupérée ou définie à l'aide de la StorageMappingItemCollection.MappingViewCacheFactoryproperty.

# Fournisseurs Entity Framework 6

07/11/2019 • 7 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Entity Framework est maintenant développé sous une licence open source, et EF6 et les versions ultérieures ne font pas partie du .NET Framework. Cela présente de nombreux avantages, mais nécessite aussi que les fournisseurs EF soient régénérés sur les assemblies EF6. Cela signifie que les fournisseurs EF pour EF5 et les versions antérieures ne fonctionnent pas avec EF6 s'ils ne sont pas regénérés.

## Quels sont les fournisseurs disponibles pour EF6 ?

Les fournisseurs regénérés pour EF6 dont nous avons connaissance sont notamment :

- **Fournisseur Microsoft SQL Server**
  - Généré à partir de la [base de code open source Entity Framework](#)
  - Partie intégrante du [package NuGet EntityFramework](#)
- **Fournisseur de l'édition Microsoft SQL Server Compact**
  - Généré à partir de la [base de code open source Entity Framework](#)
  - Partie intégrante du [package NuGet EntityFramework.SqlServerCompact](#)
- **Fournisseurs de données Devart dotConnect**
  - Il existe des fournisseurs tiers de [Devart](#) pour diverses bases de données, notamment Oracle, MySQL, PostgreSQL, SQLite, Salesforce, DB2 et SQL Server
- **Fournisseurs de logiciels CData**
  - Il existe des fournisseurs tiers de [logiciels CData](#) pour divers magasins de données, notamment Salesforce, Stockage Table Azure, MySql et bien plus encore
- **Fournisseur Firebird**
  - Disponible sous forme de [package NuGet](#)
- **Fournisseur Visual Fox Pro**
  - Disponible sous forme de [package NuGet](#)
- **MySQL**
  - [Connecteur MySQL /NET pour Entity Framework](#)
- **PostgreSQL**
  - Npgsql est disponible sous forme de [package NuGet](#)
- **Oracle**
  - ODP.NET est disponible sous forme de [package NuGet](#)

Notez que le niveau de fonctionnalité ou de prise en charge n'est pas indiqué pour un fournisseur donné, la liste indique uniquement qu'une build pour EF6 a été rendue disponible.

## Inscription de fournisseurs EF

À partir d'Entity Framework 6, les fournisseurs EF peuvent être inscrits à l'aide d'une configuration basée sur le code ou dans le fichier config de l'application.

## Inscription dans le fichier config

L'inscription du fournisseur EF dans le fichier app.config ou web.config a le format suivant :

```
<entityFramework>
  <providers>
    <provider invariantName="My.Invariant.Name" type="MyProvider.MyProviderServices, MyAssembly" />
  </providers>
</entityFramework>
```

Notez que, souvent, si le fournisseur EF est installé à partir de NuGet, le package NuGet ajoute automatiquement cette inscription au fichier config. Si vous installez le package NuGet dans un projet qui n'est pas le projet de démarrage de votre application, vous devez peut-être copier l'inscription dans le fichier config de votre projet de démarrage.

Le paramètre « InvariantName » dans cette inscription est le même nom invariant que celui utilisé pour identifier un fournisseur ADO.NET. Il s'agit de l'attribut « invariant » dans une inscription DbProviderFactories et de l'attribut « providerName » dans une inscription de chaîne de connexion. Le nom invariant à utiliser doit également se trouver dans la documentation du fournisseur. « System.Data.SqlClient » pour SQL Server et « System.Data.SqlServerCe.4.0 » pour SQL Server Compact sont des exemples de noms invariants.

Le « type » dans cette inscription est le nom qualifié d'assembly du type de fournisseur qui dérive de « System.Data.Entity.Core.Common.DbProviderServices ». Par exemple, la chaîne à utiliser pour SQL Compact est « System.Data.Entity.SqlServerCompact.SqlCeProviderServices EntityFramework.SqlServerCompact ». Le type à utiliser ici doit se trouver dans la documentation du fournisseur.

## Inscription basée sur le code

À partir d'Entity Framework 6, la configuration d'EF au niveau de l'application peut être spécifiée dans le code. Pour des détails complets, consultez [Configuration d'Entity Framework basée sur le code](#). La méthode normale pour inscrire un fournisseur EF à l'aide d'une configuration basée sur le code est de créer une classe dérivant de System.Data.Entity.DbConfiguration et de la placer dans le même assembly que votre classe DbContext. Votre classe DbConfiguration doit ensuite inscrire le fournisseur dans son constructeur. Par exemple, pour inscrire le fournisseur SQL Compact, la classe DbConfiguration ressemble à ceci :

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetProviderServices(
            SqlCeProviderServices.ProviderInvariantName,
            SqlCeProviderServices.Instance);
    }
}
```

Dans ce code « SqlCeProviderServices.ProviderInvariantName » représente la chaîne du nom invariant du fournisseur SQL Server Compact (« System.Data.SqlServerCe.4.0 ») et SqlCeProviderServices.Instance retourne l'instance singleton du fournisseur EF SQL Compact.

## Que faire si le fournisseur dont j'ai besoin n'est pas disponible ?

Si le fournisseur est disponible pour les versions précédentes d'EF, nous vous encourageons à contacter le propriétaire du fournisseur et lui demander de créer une version EF6. Vous devez indiquer une référence à la [documentation du modèle de fournisseur EF6](#).

## Puis-je écrire un fournisseur moi-même ?

Il est tout à fait possible de créer un fournisseur EF vous-même, bien que ce ne soit pas une opération quelconque. Le lien ci-dessus concernant le modèle de fournisseur EF6 est un bon point de départ. Vous pouvez aussi utiliser le code du fournisseur SQL Server et SQL CE inclus dans la [base de code open source EF](#) comme point de départ ou de référence.

Notez qu'à partir d'EF6, le fournisseur EF dépend moins du fournisseur ADO.NET sous-jacent. Vous pouvez donc plus facilement écrire un fournisseur EF sans écrire ou wrapper les classes ADO.NET.

# Modèle de fournisseur Entity Framework 6

11/10/2019 • 31 minutes to read

Le modèle de fournisseur Entity Framework permet d'utiliser des Entity Framework avec différents types de serveur de base de données. Par exemple, un fournisseur peut être connecté pour permettre l'utilisation d'EF sur Microsoft SQL Server, alors qu'un autre fournisseur peut être connecté pour permettre l'utilisation d'EF sur Microsoft SQL Server Compact édition. Les fournisseurs de EF6 que nous avons pris en charge sont disponibles sur la page [fournisseurs de Entity Framework](#).

Certaines modifications ont été apportées à la façon dont EF interagit avec les fournisseurs pour permettre la libération d'EF sous une licence Open source. Ces modifications nécessitent la régénération des fournisseurs EF sur les assemblys EF6 avec de nouveaux mécanismes d'inscription du fournisseur.

## La reconstruction

Avec EF6, le code de base qui faisait précédemment partie du .NET Framework est désormais fourni en tant qu'assemblys hors bande (OOB). Vous trouverez plus d'informations sur la création d'applications sur EF6 sur la page [mise à jour des applications pour EF6](#). Les fournisseurs devront également être reconstruits à l'aide de ces instructions.

## Vue d'ensemble des types de fournisseurs

Un fournisseur EF est en fait une collection de services spécifiques au fournisseur définis par les types CLR que ces services étendent (pour une classe de base) ou implémentent (pour une interface). Deux de ces services sont essentiels et nécessaires au fonctionnement d'EF. D'autres sont facultatifs et doivent être implémentés uniquement si une fonctionnalité spécifique est requise et/ou si les implémentations par défaut de ces services ne fonctionnent pas pour le serveur de base de données spécifique ciblé.

## Types de fournisseurs fondamentaux

### **DbProviderFactory**

EF dépend d'un type dérivé de [System.Data.Common.DbProviderFactory](#) pour effectuer tous les accès aux bases de données de bas niveau. DbProviderFactory ne fait pas réellement partie d'EF mais est une classe du .NET Framework qui sert de point d'entrée pour les fournisseurs ADO.NET qui peuvent être utilisés par EF, autre O/RMs ou directement par une application pour obtenir des instances de connexions, commandes, paramètres et autres abstractions ADO.NET de manière agnostique du fournisseur. Vous trouverez plus d'informations sur DbProviderFactory dans la [documentation MSDN relative à ADO.net](#).

### **DbProviderServices**

EF dépend d'un type dérivé de DbProviderServices pour fournir des fonctionnalités supplémentaires requises par EF sur les fonctionnalités déjà fournies par le fournisseur ADO.NET. Dans les versions antérieures d'EF, la classe DbProviderServices faisait partie du .NET Framework et a été trouvée dans l'espace de noms System.Data.Common. À partir de EF6, cette classe fait désormais partie d'EntityFramework.dll et se trouve dans l'espace de noms System.Data.Entity.Core.Common.

Vous trouverez plus d'informations sur les fonctionnalités fondamentales d'une implémentation de DbProviderServices sur [MSDN](#). Toutefois, Notez qu'au moment de la rédaction de ces informations n'est pas mis à jour pour EF6, bien que la plupart des concepts soient toujours valides. Les implémentations SQL Server et SQL Server Compact de DbProviderServices sont également consignées dans le code [base open source](#) et peuvent servir de références utiles pour d'autres implémentations.

Dans les versions antérieures d'EF, l'implémentation de DbProviderServices à utiliser a été obtenue directement à partir d'un fournisseur ADO.NET. Cela a été fait en convertissant DbProviderFactory en IServiceProvider et en appelant la méthode GetService. Cela a étroitement couplé le fournisseur EF à DbProviderFactory. Cela a pour effet d'empêcher le déplacement du EF bloqué de la .NET Framework. par conséquent, pour EF6, ce couplage étroit a été supprimé et une implémentation de DbProviderServices est maintenant inscrite directement dans le fichier de configuration de l'application ou dans un code configuration comme décrit plus en détail la section *inscription de DbProviderServices* ci-dessous.

## Services supplémentaires

Outre les services fondamentaux décrits ci-dessus, il existe également de nombreux autres services utilisés par EF, qui sont toujours ou parfois spécifiques au fournisseur. Les implémentations spécifiques au fournisseur par défaut de ces services peuvent être fournies par une implémentation de DbProviderServices. Les applications peuvent également substituer les implémentations de ces services ou fournir des implémentations lorsqu'un type DbProviderServices ne fournit pas de valeur par défaut. Cela est décrit plus en détail dans la section *résolution des services supplémentaires* ci-dessous.

Les types de service supplémentaires qu'un fournisseur peut présenter un intérêt pour un fournisseur sont répertoriés ci-dessous. Vous trouverez plus d'informations sur chacun de ces types de service dans la documentation de l'API.

### **IDbExecutionStrategy**

Il s'agit d'un service facultatif qui permet à un fournisseur d'implémenter de nouvelles tentatives ou un autre comportement lorsque des requêtes et des commandes sont exécutées sur la base de données. Si aucune implémentation n'est fournie, EF exécute simplement les commandes et propage toutes les exceptions levées. Par SQL Server ce service est utilisé pour fournir une stratégie de nouvelle tentative qui est particulièrement utile lors de l'exécution sur des serveurs de base de données basés sur le Cloud, tels que des SQL Azure.

### **IDbConnectionFactory**

Il s'agit d'un service facultatif qui permet à un fournisseur de créer des objets DbConnection par Convention lorsque seul un nom de base de données est fourni. Notez que même si ce service peut être résolu par une implémentation de DbProviderServices, il est présent depuis EF 4,1 et peut également être défini explicitement dans le fichier de configuration ou dans le code. Le fournisseur n'aura la possibilité de résoudre ce service que s'il est inscrit en tant que fournisseur par défaut (voir *le fournisseur par défaut* ci-dessous) et si une fabrique de connexion par défaut n'a pas été définie ailleurs.

### **DbSpatialServices**

Il s'agit d'un service facultatif qui permet à un fournisseur d'ajouter la prise en charge des types spatiaux Geography et Geometry. Une implémentation de ce service doit être fournie pour qu'une application puisse utiliser EF avec des types spatiaux. DbSpatialServices est demandé de deux manières. Tout d'abord, les services spatiaux spécifiques au fournisseur sont demandés à l'aide d'un objet DbProviderInfo (qui contient le nom invariant et le jeton de manifeste) en tant que clé. Deuxièmement, DbSpatialServices peut être demandé sans clé. Utilisé pour résoudre le « fournisseur spatial global » utilisé lors de la création de types DbGeography ou DbGeometry autonomes.

### **MigrationSqlGenerator**

Il s'agit d'un service facultatif qui permet d'utiliser des migrations EF pour la génération de SQL utilisée pour la création et la modification de schémas de base de données par Code First. Une implémentation est requise pour prendre en charge les migrations. Si une implémentation est fournie, elle est également utilisée lors de la création de bases de données à l'aide d'initialiseurs de base de données ou de la méthode Database.Create.

### **Func < DbConnection, String, HistoryContextFactory >**

Il s'agit d'un service facultatif qui permet à un fournisseur de configurer le mappage du HistoryContext à la table `__MigrationHistory` utilisée par les migrations EF. Le HistoryContext est un Code First DbContext et

peut être configuré à l'aide de l'API Fluent normale pour modifier des éléments tels que le nom de la table et les spécifications de mappage de colonne. L'implémentation par défaut de ce service retournée par EF pour tous les fournisseurs peut fonctionner pour un serveur de base de données donné si tous les mappages de tables et de colonnes par défaut sont pris en charge par ce fournisseur. Dans ce cas, le fournisseur n'a pas besoin de fournir une implémentation de ce service.

### **IDbProviderFactoryResolver**

Il s'agit d'un service facultatif permettant d'obtenir le DbProviderFactory approprié à partir d'un objet DbConnection donné. L'implémentation par défaut de ce service retournée par EF pour tous les fournisseurs est conçue pour fonctionner pour tous les fournisseurs. Toutefois, en cas d'exécution sur .NET 4, DbProviderFactory n'est pas accessible publiquement à partir de l'un de ses DbConnections. Par conséquent, EF utilise des heuristiques pour rechercher des correspondances dans les fournisseurs inscrits. Il est possible que pour certains fournisseurs, ces heuristiques échouent et, dans de telles situations, le fournisseur doit fournir une nouvelle implémentation.

## Inscription de DbProviderServices

L'implémentation de DbProviderServices à utiliser peut être inscrite dans le fichier de configuration de l'application (App. config ou Web. config) ou à l'aide d'une configuration basée sur le code. Dans les deux cas, l'inscription utilise le « nom invariant » du fournisseur comme clé. Cela permet d'inscrire plusieurs fournisseurs et de les utiliser dans une même application. Le nom invariant utilisé pour les inscriptions EF est le même que le nom invariant utilisé pour l'inscription du fournisseur ADO.NET et les chaînes de connexion. Par exemple, pour SQL Server le nom invariant « System. Data. SqlClient » est utilisé.

### **Inscription dans le fichier config**

Le type DbProviderServices à utiliser est inscrit en tant qu'élément de fournisseur dans la liste des fournisseurs de la section entityFramework du fichier de configuration de l'application. Exemple :

```
<entityFramework>
  <providers>
    <provider invariantName="My.Invariant.Name" type="MyProvider.MyProviderServices, MyAssembly" />
  </providers>
</entityFramework>
```

Le *type* chaîne doit être le nom de type qualifié par un assembly de l'implémentation DbProviderServices à utiliser.

### **Inscription basée sur le code**

Le démarrage des fournisseurs EF6 peut également être inscrit à l'aide de code. Cela permet l'utilisation d'un fournisseur EF sans aucune modification du fichier de configuration de l'application. Pour utiliser la configuration basée sur le code, une application doit créer une classe DbConfiguration comme décrit dans la [documentation relative à la configuration basée sur le code](#). Le constructeur de la classe DbConfiguration doit ensuite appeler SetProviderServices pour inscrire le fournisseur EF. Exemple :

```
public class MyConfiguration : DbConfiguration
{
    public MyConfiguration()
    {
        SetProviderServices("My.New.Provider", new MyProviderServices());
    }
}
```

## Résolution de services supplémentaires

Comme indiqué ci-dessus dans la section *vue d'ensemble des types de fournisseurs*, une classe DbProviderServices peut également être utilisée pour résoudre des services supplémentaires. Cela est possible car DbProviderServices implémente IDbDependencyResolver et chaque type de DbProviderServices inscrit est ajouté comme « programme de résolution par défaut ». Le mécanisme IDbDependencyResolver est décrit plus en détail dans [résolution des dépendances](#). Toutefois, il n'est pas nécessaire de comprendre tous les concepts de cette spécification pour résoudre des services supplémentaires dans un fournisseur.

La façon la plus courante pour un fournisseur de résoudre des services supplémentaires consiste à appeler DbProviderServices. AddDependencyResolver pour chaque service dans le constructeur de la classe DbProviderServices. Par exemple, SqlProviderServices (fournisseur EF pour SQL Server) a du code similaire à celui-ci pour l'initialisation :

```
private SqlProviderServices()
{
    AddDependencyResolver(new SingletonDependencyResolver<IDbConnectionFactory>(
        new SqlConnectionFactory()));

    AddDependencyResolver(new ExecutionStrategyResolver<DefaultSqlExecutionStrategy>(
        "System.data.SqlClient", null, () => new DefaultSqlExecutionStrategy()));

    AddDependencyResolver(new SingletonDependencyResolver<Func<MigrationSqlGenerator>>(
        () => new SqlServerMigrationSqlGenerator(), "System.data.SqlClient"));

    AddDependencyResolver(new SingletonDependencyResolver<DbSpatialServices>(
        SqlSpatialServices.Instance,
        k =>
    {
        var asSpatialKey = k as DbProviderInfo;
        return asSpatialKey == null
            || asSpatialKey.ProviderInvariantName == ProviderInvariantName;
    }));
}
```

Ce constructeur utilise les classes d'assistance suivantes :

- SingletonDependencyResolver : fournit un moyen simple de résoudre les services Singleton, autrement dit, les services pour lesquels la même instance est retournée chaque fois que GetService est appelé. Les services temporaires sont souvent inscrits en tant que fabrique singleton qui sera utilisée pour créer des instances transitoires à la demande.
- ExecutionStrategyResolver : programme de résolution spécifique pour retourner des implémentations de IExecutionStrategy.

Au lieu d'utiliser DbProviderServices. AddDependencyResolver, il est également possible de remplacer DbProviderServices. GetService et de résoudre directement les services supplémentaires. Cette méthode est appelée quand EF a besoin d'un service défini par un certain type et, dans certains cas, pour une clé donnée. La méthode doit retourner le service si possible, ou retourner la valeur null pour refuser le retour du service et autoriser à la place une autre classe à le résoudre. Par exemple, pour résoudre la fabrique de connexion par défaut, le code de GetService peut se présenter comme suit :

```
public override object GetService(Type type, object key)
{
    if (type == typeof(IDbConnectionFactory))
    {
        return new SqlConnectionFactory();
    }
    return null;
}
```

## Ordre d'inscription

Lorsque plusieurs implémentations de DbProviderServices sont inscrites dans le fichier de configuration d'une application, elles sont ajoutées en tant que programmes de résolution secondaires dans l'ordre dans lequel elles sont répertoriées. Étant donné que les programmes de résolution sont toujours ajoutés en haut de la chaîne de résolution secondaire, cela signifie que le fournisseur à la fin de la liste aura la possibilité de résoudre les dépendances avant les autres. (Cela peut paraître un peu intuitif dans un premier temps, mais il est judicieux de faire en sorte que chaque fournisseur soit sorti de la liste et qu'il soit empilé par-dessus les fournisseurs existants.)

Ce classement n'a généralement pas d'importance, car la plupart des services de fournisseur sont spécifiques au fournisseur et indexés par nom invariant du fournisseur. Toutefois, pour les services qui ne sont pas indexés par le nom invariant du fournisseur ou une autre clé spécifique au fournisseur, le service est résolu en fonction de ce classement. Par exemple, s'il n'est pas explicitement défini différemment à un autre endroit, la fabrique de connexion par défaut proviendra du fournisseur le plus élevé dans la chaîne.

## Inscriptions de fichier de configuration supplémentaires

Il est possible d'inscrire explicitement certains des services de fournisseur supplémentaires décrits ci-dessus directement dans le fichier de configuration d'une application. Une fois cette opération effectuée, l'enregistrement dans le fichier de configuration sera utilisé à la place de tout ce qui est retourné par la méthode GetService de l'implémentation DbProviderServices.

### Inscription de la fabrique de connexion par défaut

À compter de EF5, le package NuGet EntityFramework a inscrit automatiquement la fabrique de connexion SQL Express ou la fabrique de connexion de la base de données locale dans le fichier de configuration.

Exemple :

```
<entityFramework>
  <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework">
  </defaultConnectionFactory>
</entityFramework>
```

Le *type* est le nom de type qualifié d'assembly pour la fabrique de connexion par défaut, qui doit implémenter IDbConnectionFactory.

Il est recommandé qu'un package NuGet de fournisseur définisse la fabrique de connexion par défaut de cette manière lors de son installation. Consultez *packages NuGet pour les fournisseurs* ci-dessous.

## Modifications supplémentaires du fournisseur EF6

### Modifications du fournisseur spatial

Les fournisseurs qui prennent en charge les types spatiaux doivent maintenant implémenter des méthodes supplémentaires sur les classes dérivant de DbSpatialDataReader :

- `public abstract bool IsGeographyColumn(int ordinal)`
- `public abstract bool IsGeometryColumn(int ordinal)`

Il existe également de nouvelles versions asynchrones des méthodes existantes qui sont recommandées pour être substituées, car les implémentations par défaut délèguent aux méthodes synchrones et ne s'exécutent donc pas de façon asynchrone :

- `public virtual Task<DbGeography> GetGeographyAsync(int ordinal, CancellationToken cancellationToken)`
- `public virtual Task<DbGeometry> GetGeometryAsync(int ordinal, CancellationToken cancellationToken)`

## Prise en charge native de Enumerable. Contains

EF6 introduit un nouveau type d'expression, `DbInExpression`, qui a été ajouté pour résoudre les problèmes de performances liés à l'utilisation de `Enumerable.Contains` dans les requêtes LINQ. La classe `DbProviderManifest` a une nouvelle méthode virtuelle, `SupportsInExpression`, qui est appelée par EF pour déterminer si un fournisseur gère le nouveau type d'expression. Pour la compatibilité avec les implémentations de fournisseur existantes, la méthode retourne `false`. Pour tirer parti de cette amélioration, un fournisseur EF6 peut ajouter du code pour gérer `DbInExpression` et remplacer `SupportsInExpression` pour retourner la valeur `true`. Une instance de `DbInExpression` peut être créée en appelant la méthode `DbExpressionBuilder.In`. Une instance `DbInExpression` est composée d'un `DbExpression`, représentant généralement une colonne de table et d'une liste de `DbConstantExpression` à tester pour rechercher une correspondance.

## Packages NuGet pour les fournisseurs

L'une des façons de rendre un fournisseur EF6 disponible est de le publier en tant que package NuGet.

L'utilisation d'un package NuGet présente les avantages suivants :

- Il est facile d'utiliser NuGet pour ajouter l'inscription du fournisseur au fichier de configuration de l'application
- Des modifications supplémentaires peuvent être apportées au fichier de configuration pour définir la fabrique de connexion par défaut afin que les connexions établies par convention utilisent le fournisseur inscrit
- NuGet gère l'ajout de redirections de liaison afin que le fournisseur EF6 doit continuer à fonctionner même après la publication d'un nouveau package EF

Par exemple, le package `EntityFramework.SqlServerCompact` inclus dans le code [base open source](#). Ce package fournit un bon modèle pour créer des packages NuGet du fournisseur EF.

## Commandes PowerShell

Quand le package NuGet `EntityFramework` est installé, il inscrit un module PowerShell qui contient deux commandes qui sont très utiles pour les packages de fournisseur :

- `Add-EFProvider` ajoute une nouvelle entité pour le fournisseur dans le fichier de configuration du projet cible et vérifie qu'il se trouve à la fin de la liste des fournisseurs inscrits.
- `Add-EFDefaultConnectionFactory` ajoute ou met à jour l'inscription de `defaultConnectionFactory` dans le fichier de configuration du projet cible.

Ces deux commandes prennent en charge l'ajout d'une section `entityFramework` au fichier de configuration et l'ajout d'une collection de fournisseurs si nécessaire.

Il est prévu que ces commandes soient appelées à partir du script NuGet `install.ps1`. Par exemple, `install.ps1` pour le fournisseur SQL Compact se présente comme suit :

```
param($installPath, $toolsPath, $package, $project)
Add-EFDefaultConnectionFactory $project 'System.Data.Entity.Infrastructure.SqlCeConnectionFactory,
EntityFramework' -ConstructorArguments 'System.Data.SqlClient.4.0'
Add-EFProvider $project 'System.Data.SqlClient.4.0'
'System.Data.Entity.SqlServerCompact.SqlCeProviderServices, EntityFramework.SqlServerCompact'</pre>
```

Pour plus d'informations sur ces commandes, consultez l'aide de la commande `obtenir-aide` dans la fenêtre de la console du gestionnaire de package.

## Encapsuler des fournisseurs

Un fournisseur d'encapsulation est un fournisseur EF et/ou ADO.NET qui encapsule un fournisseur existant pour l'étendre avec d'autres fonctionnalités telles que les fonctionnalités de profilage ou de suivi. Les

fournisseurs d'encapsulation peuvent être enregistrés de manière normale, mais il est souvent plus pratique de configurer le fournisseur d'encapsulation au moment de l'exécution en interceptant la résolution des services liés au fournisseur. L'événement statique OnLockingConfiguration sur la classe DbConfiguration peut être utilisé pour effectuer cette opération.

OnLockingConfiguration est appelé une fois que EF a déterminé où toutes les configurations EF pour le domaine d'application seront obtenues à partir de, mais avant qu'il ne soit verrouillé pour utilisation. Au démarrage de l'application (avant l'utilisation d'EF), l'application doit inscrire un gestionnaire d'événements pour cet événement. (Nous envisageons d'ajouter la prise en charge de l'inscription de ce gestionnaire dans le fichier de configuration, mais cela n'est pas encore pris en charge.) Le gestionnaire d'événements doit ensuite effectuer un appel à ReplaceService pour chaque service qui doit être encapsulé.

Par exemple, pour encapsuler IDbConnectionFactory et DbProviderService, un gestionnaire comme celui-ci doit être inscrit :

```
DbConfiguration.OnLockingConfiguration +=  
    (_, a) =>  
    {  
        a.ReplaceService<DbProviderServices>(  
            (s, k) => new MyWrappedProviderServices(s));  
  
        a.ReplaceService<IDbConnectionFactory>(  
            (s, k) => new MyWrappedConnectionFactory(s));  
    };
```

Le service qui a été résolu et doit maintenant être encapsulé avec la clé qui a été utilisée pour résoudre le service est passé au gestionnaire. Le gestionnaire peut ensuite encapsuler ce service et remplacer le service retourné par la version encapsulée.

## Résolution d'un DbProviderFactory avec EF

DbProviderFactory est l'un des types de fournisseurs fondamentaux nécessaires à EF, comme décrit dans la section *vue d'ensemble des types de fournisseurs* ci-dessus. Comme mentionné précédemment, il ne s'agit pas d'un type EF et l'inscription ne fait généralement pas partie de la configuration d'EF, mais plutôt de l'inscription normale du fournisseur ADO.NET dans le fichier machine.config et/ou le fichier de configuration de l'application.

Bien que ce système EF utilise toujours son mécanisme de résolution de dépendance normal lorsqu'il recherche un DbProviderFactory à utiliser. Le programme de résolution par défaut utilise l'inscription normale ADO.NET dans les fichiers de configuration, ce qui est généralement transparent. Toutefois, étant donné que le mécanisme de résolution de dépendances normal est utilisé, cela signifie qu'un IDbDependencyResolver peut être utilisé pour résoudre un DbProviderFactory même lorsque l'inscription de ADO.NET normale n'a pas été effectuée.

La résolution de DbProviderFactory de cette manière a plusieurs implications :

- Une application utilisant la configuration basée sur du code peut ajouter des appels dans sa classe DbConfiguration pour inscrire le DbProviderFactory approprié. Cela s'avère particulièrement utile pour les applications qui ne veulent pas (ou ne peuvent pas) utiliser une configuration basée sur des fichiers.
- Le service peut être encapsulé ou remplacé à l'aide de ReplaceService, comme décrit dans la section *fournisseurs d'encapsulation* ci-dessus.
- En théorie, une implémentation de DbProviderServices peut résoudre un DbProviderFactory.

Le point important à prendre en compte dans ces cas de figure est qu'ils n'affectent que la recherche de DbProviderFactory par EF. D'autres codes non EF peuvent toujours s'attendre à ce que le fournisseur ADO.NET soit enregistré de manière normale et risque d'échouer si l'inscription est introuvable. Pour cette

raison, il est généralement préférable d'inscrire un DbProviderFactory en mode ADO.NET normal.

## Services connexes

Si EF est utilisé pour résoudre un DbProviderFactory, il doit également résoudre les services IProviderInvariantName et IDbProviderFactoryResolver.

IProviderInvariantName est un service utilisé pour déterminer un nom invariant de fournisseur pour un type donné de DbProviderFactory. L'implémentation par défaut de ce service utilise l'inscription du fournisseur ADO.NET. Cela signifie que si le fournisseur ADO.NET n'est pas enregistré de manière normale, car DbProviderFactory est résolu par EF, il est également nécessaire de résoudre ce service. Notez qu'un programme de résolution pour ce service est automatiquement ajouté lors de l'utilisation de la méthode DbConfiguration.SetProviderFactory.

Comme décrit dans la section *vue d'ensemble des types de fournisseurs* ci-dessus, IDbProviderFactoryResolver est utilisé pour obtenir le DbProviderFactory approprié à partir d'un objet DbConnection donné.

L'implémentation par défaut de ce service lors de l'exécution sur .NET 4 utilise l'inscription du fournisseur ADO.NET. Cela signifie que si le fournisseur ADO.NET n'est pas enregistré de manière normale, car DbProviderFactory est résolu par EF, il est également nécessaire de résoudre ce service.

# Prise en charge des fournisseurs pour les types spatiaux

11/10/2019 • 5 minutes to read

Entity Framework prend en charge l'utilisation des données spatiales via les classes DbGeography ou DbGeometry. Ces classes reposent sur les fonctionnalités spécifiques à la base de données offertes par le fournisseur Entity Framework. Tous les fournisseurs ne prennent pas en charge les données spatiales et ceux qui peuvent avoir des conditions préalables supplémentaires telles que l'installation d'assemblies de type spatial. Vous trouverez ci-dessous des informations supplémentaires sur la prise en charge des fournisseurs pour les types spatiaux.

Vous trouverez des informations supplémentaires sur l'utilisation des types spatiaux dans une application dans deux procédures pas à pas, une pour Code First, l'autre pour Database First ou Model First :

- [Types de données spatiales dans Code First](#)
- [Types de données spatiales dans le concepteur EF](#)

## Versions d'EF qui prennent en charge les types spatiaux

La prise en charge des types spatiaux a été introduite dans EF5. Toutefois, dans les types spatiaux EF5 sont uniquement pris en charge lorsque l'application cible et s'exécute sur .NET 4,5.

Le démarrage des types spatiaux EF6 est pris en charge pour les applications ciblant à la fois .NET 4 et .NET 4,5.

## Fournisseurs EF qui prennent en charge les types spatiaux

### EF5

Les fournisseurs de Entity Framework pour EF5 que nous avons conscients de la prise en charge des types spatiaux sont :

- Fournisseur Microsoft SQL Server
  - Ce fournisseur est fourni dans le cadre de EF5.
  - Ce fournisseur dépend de certaines bibliothèques de bas niveau supplémentaires qui devront peut-être être installées. pour plus d'informations, voir ci-dessous.
- [Devart dotConnect pour Oracle](#)
  - Il s'agit d'un fournisseur tiers de Devart.

Si vous connaissez un fournisseur EF5 qui prend en charge les types spatiaux, contactez-nous et nous serons heureux de l'ajouter à cette liste.

### EF6

Les fournisseurs de Entity Framework pour EF6 que nous avons conscients de la prise en charge des types spatiaux sont :

- Fournisseur Microsoft SQL Server
  - Ce fournisseur est fourni dans le cadre de EF6.
  - Ce fournisseur dépend de certaines bibliothèques de bas niveau supplémentaires qui devront peut-être être installées. pour plus d'informations, voir ci-dessous.
- [Devart dotConnect pour Oracle](#)

- Il s'agit d'un fournisseur tiers de Devart.

Si vous connaissez un fournisseur EF6 qui prend en charge les types spatiaux, contactez-nous et nous serons heureux de l'ajouter à cette liste.

## Conditions préalables pour les types spatiaux avec Microsoft SQL Server

SQL Server la prise en charge spatiale dépend des types de SQL Server de bas niveau, SqlGeography et SqlGeometry. Ces types se trouvent dans l'assembly Microsoft.SqlServer.Types.dll, et cet assembly n'est pas fourni avec EF ou dans le cadre de l' .NET Framework.

Quand Visual Studio est installé, il installe également une version de SQL Server, ce qui inclut l'installation de Microsoft.SqlServer.Types.dll.

Si SQL Server n'est pas installé sur l'ordinateur où vous souhaitez utiliser les types spatiaux, ou si les types spatiaux ont été exclus de l'installation SQL Server, vous devez les installer manuellement. Les types peuvent être installés à l'aide de `SQLSysClrTypes.msi`, qui fait partie de Microsoft SQL Server Feature Pack. Les types spatiaux étant spécifiques à la version SQL Server, nous vous recommandons de [Rechercher « SQL Server Feature Pack »](#) dans le centre de téléchargement Microsoft, puis de sélectionner et de télécharger l'option correspondant à la version de SQL Server que vous allez utiliser.

# Utilisation de proxys

13/09/2018 • 4 minutes to read

Lorsque vous créez des instances de types d'entités POCO, Entity Framework crée souvent des instances d'un type dérivé généré dynamiquement qui agit comme un proxy pour l'entité. Ce proxy substitue certaines propriétés virtuelles de l'entité à insérer des raccordements pour effectuer des actions automatiquement lorsque la propriété est accessible. Par exemple, ce mécanisme est utilisé pour prendre en charge le chargement différé des relations. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

## La désactivation de la création de proxy

Il est parfois utile empêcher la création d'instances de proxy d'Entity Framework. Par exemple, la sérialisation des instances de proxy non est considérablement plus facile que la sérialisation des instances de proxy. La création de proxy peut être désactivée en désactivant l'indicateur `ProxyCreationEnabled`. Un seul endroit, vous pouvez procéder ainsi est dans le constructeur de votre contexte. Exemple :

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    {
        this.Configuration.ProxyCreationEnabled = false;
    }

    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

Notez que l'Entity Framework ne crée pas de proxys pour les types où il n'existe rien pour le proxy à faire. Cela signifie que vous pouvez également éviter les proxys grâce à des types qui sont scellés et/ou disposent pas de propriétés virtuels.

## Création explicite d'une instance d'un proxy

Une instance de proxy ne sera pas créée si vous créez une instance d'une entité à l'aide de l'opérateur `new`. Il peut s'agir d'un problème, mais si vous avez besoin créer une instance de proxy (par exemple, pour que différé le chargement ou le proxy le suivi des modifications fonctionnent) puis vous pouvez le faire à l'aide de la méthode `Create` de `DbSet`. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Create();
```

La version générique de création peut être utilisée si vous souhaitez créer une instance d'un type d'entité dérivé. Exemple :

```
using (var context = new BloggingContext())
{
    var admin = context.Users.Create<Administrator>();
```

Notez que la méthode Create ne pas ajouter ou joindre l'entité créée dans le contexte.

Notez que la méthode Create ne crée simplement une instance du type d'entité si la création d'un type de proxy pour l'entité n'aurait aucune valeur, car il ne ferait rien. Par exemple, si le type d'entité est scellé et/ou ne possède aucune propriété virtuelle créer créer simplement une instance du type d'entité.

## Obtenir le type d'entité réelle à partir d'un type de proxy

Types de proxy ont des noms qui ressemblent à ceci :

System.Data.Entity.DynamicProxies.Blog\_5E43C6C196972BF0754973E48C9C941092D86818CD94005E9A759B70BF6E48E6

Vous pouvez trouver le type d'entité pour ce type de proxy à l'aide de la méthode GetObjectType a d'ObjectContext. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    var entityType = ObjectContext.GetObjectType(blog.GetType());
}
```

Notez que si le type passé à GetObjectType a est une instance d'un type d'entité qui n'est pas un type de proxy, le type d'entité est toujours retournée. Cela signifie que vous pouvez toujours utiliser cette méthode pour obtenir le type d'entité réel sans aucune vérification pour voir si le type est un type de proxy ou non.

# Test avec un Framework fictif

23/11/2019 • 14 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Lors de l'écriture de tests pour votre application, il est souvent préférable d'éviter d'atteindre la base de données. Entity Framework vous permet d'y parvenir en créant un contexte, avec un comportement défini par vos tests, qui utilise des données en mémoire.

## Options de création de doubles de test

Il existe deux approches différentes qui peuvent être utilisées pour créer une version en mémoire de votre contexte.

- **Créer vos propres doubles de test** : cette approche implique l'écriture de votre propre implémentation en mémoire de votre contexte et DbSets. Cela vous donne un grand contrôle sur la façon dont les classes se comportent, mais peut impliquer l'écriture et la possession d'une quantité raisonnable de code.
- **Utilisez une infrastructure fictive pour créer des doubles de test** : à l'aide d'une infrastructure fictive (telle que MOQ), vous pouvez avoir les implémentations en mémoire de votre contexte et les jeux créés dynamiquement au moment de l'exécution.

Cet article traite de l'utilisation d'un Framework fictif. Pour créer vos propres doubles de test, consultez [test avec vos propres doubles de test](#).

Pour illustrer l'utilisation d'EF avec une infrastructure fictive, nous allons utiliser MOQ. Le moyen le plus simple d'utiliser MOQ consiste à installer le [package MOQ à partir de NuGet](#).

## Test avec les versions antérieures à EF6

Le scénario présenté dans cet article dépend de certaines modifications apportées à DbSet dans EF6. Pour tester avec EF5 et les versions antérieures, consultez [test avec un contexte factice](#).

## Limitations des doubles de test en mémoire EF

Les doubles de test en mémoire peuvent être un bon moyen de fournir une couverture de niveau test unitaire des bits de votre application qui utilisent EF. Toutefois, lorsque vous procédez ainsi, vous utilisez LINQ to Objects pour exécuter des requêtes sur des données en mémoire. Cela peut entraîner un comportement différent de celui de l'utilisation du fournisseur LINQ d'EF (LINQ to Entities) pour convertir les requêtes en SQL exécuté sur votre base de données.

Le chargement de données associées est un exemple de cette différence. Si vous créez une série de blogs qui ont chacun des publications associées, lorsque vous utilisez des données en mémoire, les publications associées sont toujours chargées pour chaque blog. Toutefois, lors de l'exécution sur une base de données, les données sont chargées uniquement si vous utilisez la méthode `Include`.

Pour cette raison, il est recommandé de toujours inclure un certain niveau de test de bout en bout (en plus de vos tests unitaires) pour garantir le bon fonctionnement de votre application sur une base de données.

## Suivre les étapes de cet article

Cet article fournit des listes de code complètes que vous pouvez copier dans Visual Studio pour suivre la procédure si vous le souhaitez. Il est plus facile de créer un **projet de test unitaire** et vous devrez cibler **.NET Framework 4,5** pour terminer les sections qui utilisent Async.

## Le modèle EF

Le service que nous allons tester utilise un modèle EF constitué du BloggingContext et du blog et des classes de publication. Ce code peut avoir été généré par le concepteur EF ou être un modèle de Code First.

```
using System.Collections.Generic;
using System.Data.Entity;

namespace TestingDemo
{
    public class BloggingContext : DbContext
    {
        public virtual DbSet<Blog> Blogs { get; set; }
        public virtual DbSet<Post> Posts { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
        public string Url { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
}
```

### Propriétés DbSet virtuelles avec le concepteur EF

Notez que les propriétés DbSet sur le contexte sont marquées comme virtuelles. Cela permettra à l'infrastructure factice de dériver de notre contexte et de remplacer ces propriétés par une implémentation fictive.

Si vous utilisez Code First vous pouvez modifier vos classes directement. Si vous utilisez le concepteur EF, vous devez modifier le modèle T4 qui génère votre contexte. Ouvrez le > de model\_name <. Fichier Context.tt imbriqué sous votre fichier edmx, recherchez le fragment de code suivant et ajoutez le mot clé Virtual comme indiqué.

```
public string DbSet(EntitySet entitySet)
{
    return string.Format(
        CultureInfo.InvariantCulture,
        "{0} virtual DbSet\<{1}> {2} {{ get; set; }}",
        Accessibility.ForReadOnlyProperty(entitySet),
        _typeMapper.GetTypeName(entitySet.ElementType),
        _code.Escape(entitySet));
}
```

## Service à tester

Pour illustrer le test avec des doubles de test en mémoire, nous allons écrire deux tests pour un BlogService. Le service est en charge de la création de nouveaux blogs (AddBlog) et de la restauration de tous les blogs classés par nom (GetAllBlogs). En plus de GetAllBlogs, nous avons également fourni une méthode qui permet d'obtenir de façon asynchrone tous les blogs classés par nom (GetAllBlogsAsync).

```
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    public class BlogService
    {
        private BloggingContext _context;

        public BlogService(BloggingContext context)
        {
            _context = context;
        }

        public Blog AddBlog(string name, string url)
        {
            var blog = _context.Blogs.Add(new Blog { Name = name, Url = url });
            _context.SaveChanges();

            return blog;
        }

        public List<Blog> GetAllBlogs()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return query.ToList();
        }

        public async Task<List<Blog>> GetAllBlogsAsync()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return await query.ToListAsync();
        }
    }
}
```

## Test des scénarios non-requête

C'est tout ce dont nous avons besoin pour commencer à tester des méthodes qui ne sont pas des requêtes. Le test suivant utilise MOQ pour créer un contexte. Il crée ensuite un DbSet<blog> et le lie à partir de la propriété blogs du contexte. Ensuite, le contexte est utilisé pour créer un nouveau BlogService qui est ensuite utilisé pour créer un nouveau blog, à l'aide de la méthode AddBlog. Enfin, le test vérifie que le service a ajouté un nouveau blog et a appelé SaveChanges dans le contexte.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using System.Data.Entity;

namespace TestingDemo
{
    [TestClass]
    public class NonQueryTests
    {
        [TestMethod]
        public void CreateBlog_saves_a_blog_via_context()
        {
            var mockSet = new Mock<DbSet<Blog>>();

            var mockContext = new Mock<BloggingContext>();
            mockContext.Setup(m => m.Blogs).Returns(mockSet.Object);

            var service = new BlogService(mockContext.Object);
            service.AddBlog("ADO.NET Blog", "http://blogs.msdn.com/adonet");

            mockSet.Verify(m => m.Add(It.IsAny<Blog>()), Times.Once());
            mockContext.Verify(m => m.SaveChanges(), Times.Once());
        }
    }
}

```

## Test des scénarios de requête

Pour pouvoir exécuter des requêtes sur notre double de test DbSet, nous devons configurer une implémentation de IQueryble. La première étape consiste à créer des données en mémoire : nous utilisons une liste<blog>. Ensuite, nous créons un contexte et DBSet<blog> puis associons l'implémentation IQueryble pour le DbSet : elles se déléguent simplement au fournisseur LINQ to Objects qui fonctionne avec la liste<T>.

Nous pouvons ensuite créer un BlogService basé sur nos doubles de test et vous assurer que les données que nous obtenons de GetAllBlogs sont classées par nom.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace TestingDemo
{
    [TestClass]
    public class QueryTests
    {
        [TestMethod]
        public void GetAllBlogs_orders_by_name()
        {
            var data = new List<Blog>
            {
                new Blog { Name = "BBB" },
                new Blog { Name = "ZZZ" },
                new Blog { Name = "AAA" },
            }.AsQueryable();

            var mockSet = new Mock<DbSet<Blog>>();
            mockSet.As<IQueryable<Blog>>().Setup(m => m.Provider).Returns(data.Provider);
            mockSet.As<IQueryable<Blog>>().Setup(m => m.Expression).Returns(data.Expression);
            mockSet.As<IQueryable<Blog>>().Setup(m => m.ElementType).Returns(data.ElementType);
            mockSet.As<IQueryable<Blog>>().Setup(m => m.GetEnumerator()).Returns(data.GetEnumerator());

            var mockContext = new Mock<BloggingContext>();
            mockContext.Setup(c => c.Blogs).Returns(mockSet.Object);

            var service = new BlogService(mockContext.Object);
            var blogs = service.GetAllBlogs();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

## Test avec des requêtes Async

Entity Framework 6 a introduit un ensemble de méthodes d'extension qui peuvent être utilisées pour exécuter une requête de manière asynchrone. `ToListAsync`, `FirstAsync`, `ForEachAsync`, etc. sont des exemples de ces méthodes.

Étant donné que les requêtes de Entity Framework utilisent LINQ, les méthodes d'extension sont définies sur `IQueryable` et `IEnumerable`. Toutefois, étant donné qu'elles sont uniquement conçues pour être utilisées avec Entity Framework vous pouvez recevoir l'erreur suivante si vous essayez de les utiliser sur une requête LINQ qui n'est pas une requête Entity Framework :

L'`IQueryable` source n'implémente pas `IDbAsyncEnumerable{0}`. Seules les sources qui implémentent `IDbAsyncEnumerable` peuvent être utilisées pour Entity Framework opérations asynchrones. Pour plus d'informations, consultez <http://go.microsoft.com/fwlink/?LinkId=287068>.

Alors que les méthodes `Async` sont uniquement prises en charge lors d'une exécution sur une requête EF, vous pouvez les utiliser dans votre test unitaire en cas d'exécution sur un double de test en mémoire d'un `DbSet`.

Pour pouvoir utiliser les méthodes `Async`, nous devons créer un `DbAsyncQueryProvider` en mémoire pour traiter la requête asynchrone. Bien qu'il soit possible de configurer un fournisseur de requêtes à l'aide de MOQ, il est beaucoup plus facile de créer une implémentation de double de test dans le code. Le code pour cette implémentation est le suivant :

```
using System.Collections.Generic;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Linq.Expressions;
using System.Threading;
using System.Threading.Tasks;

namespace TestingDemo
{
    internal class TestDbAsyncQueryProvider<TEntity> : IDbAsyncQueryProvider
    {
        private readonly IQueryProvider _inner;

        internal TestDbAsyncQueryProvider(IQueryProvider inner)
        {
            _inner = inner;
        }

        public IQueryable CreateQuery(Expression expression)
        {
            return new TestDbAsyncEnumerable<TEntity>(expression);
        }

        public IQueryable<TElement> CreateQuery<TElement>(Expression expression)
        {
            return new TestDbAsyncEnumerable<TElement>(expression);
        }

        public object Execute(Expression expression)
        {
            return _inner.Execute(expression);
        }

        public TResult Execute<TResult>(Expression expression)
        {
            return _inner.Execute<TResult>(expression);
        }

        public Task<object> ExecuteAsync(Expression expression, CancellationToken cancellationToken)
        {
            return Task.FromResult(Execute(expression));
        }

        public Task<TResult> ExecuteAsync<TResult>(Expression expression, CancellationToken cancellationToken)
        {
            return Task.FromResult(Execute<TResult>(expression));
        }
    }

    internal class TestDbAsyncEnumerable<T> : EnumerableQuery<T>, IDbAsyncEnumerable<T>, IQueryable<T>
    {
        public TestDbAsyncEnumerable(IEnumerable<T> enumerable)
            : base(enumerable)
        { }

        public TestDbAsyncEnumerable(Expression expression)
            : base(expression)
        { }

        public IDbAsyncEnumerator<T> GetAsyncEnumerator()
        {
            return new TestDbAsyncEnumerator<T>(this.AsEnumerable().GetEnumerator());
        }

        IDbAsyncEnumerator IDbAsyncEnumerable.GetAsyncEnumerator()
        {
            return GetAsyncEnumerator();
        }
    }
}
```

```

        IQueryProvider IQueryable.Provider
    {
        get { return new TestDbAsyncQueryProvider<T>(this); }
    }
}

internal class TestDbAsyncEnumerator<T> : IDbAsyncEnumerator<T>
{
    private readonly IEnumerator<T> _inner;

    public TestDbAsyncEnumerator(IEnumerator<T> inner)
    {
        _inner = inner;
    }

    public void Dispose()
    {
        _inner.Dispose();
    }

    public Task<bool> MoveNextAsync(CancellationToken cancellationToken)
    {
        return Task.FromResult(_inner.MoveNext());
    }

    public T Current
    {
        get { return _inner.Current; }
    }

    object IDbAsyncEnumerator.Current
    {
        get { return Current; }
    }
}
}

```

Maintenant que nous disposons d'un fournisseur de requêtes asynchrones, nous pouvons écrire un test unitaire pour notre nouvelle méthode GetAllBlogsAsync.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using Moq;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    [TestClass]
    public class AsyncQueryTests
    {
        [TestMethod]
        public async Task GetAllBlogsAsync_orders_by_name()
        {

            var data = new List<Blog>
            {
                new Blog { Name = "BBB" },
                new Blog { Name = "ZZZ" },
                new Blog { Name = "AAA" },
            }.AsQueryable();

            var mockSet = new Mock<DbSet<Blog>>();
            mockSet.As< IDbAsyncEnumerable<Blog>>()
                .Setup(m => m.GetAsyncEnumerator())
                .Returns(new TestDbAsyncEnumerator<Blog>(data.GetEnumerator()));

            mockSet.As< IQueryable<Blog>>()
                .Setup(m => m.Provider)
                .Returns(new TestDbAsyncQueryProvider<Blog>(data.Provider));

            mockSet.As< IQueryable<Blog>>().Setup(m => m.Expression).Returns(data.Expression);
            mockSet.As< IQueryable<Blog>>().Setup(m => m.ElementType).Returns(data.ElementType);
            mockSet.As< IQueryable<Blog>>().Setup(m => m.GetEnumerator()).Returns(data.GetEnumerator());

            var mockContext = new Mock<BloggingContext>();
            mockContext.Setup(c => c.Blogs).Returns(mockSet.Object);

            var service = new BlogService(mockContext.Object);
            var blogs = await service.GetAllBlogsAsync();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

# Test avec vos propres doubles de test

23/11/2019 • 15 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Lors de l'écriture de tests pour votre application, il est souvent préférable d'éviter d'atteindre la base de données. Entity Framework vous permet d'y parvenir en créant un contexte, avec un comportement défini par vos tests, qui utilise des données en mémoire.

## Options de création de doubles de test

Il existe deux approches différentes qui peuvent être utilisées pour créer une version en mémoire de votre contexte.

- **Créer vos propres doubles de test** : cette approche implique l'écriture de votre propre implémentation en mémoire de votre contexte et DbSets. Cela vous donne un grand contrôle sur la façon dont les classes se comportent, mais peut impliquer l'écriture et la possession d'une quantité raisonnable de code.
- **Utilisez une infrastructure fictive pour créer des doubles de test** : à l'aide d'une infrastructure fictive (telle que MOQ), vous pouvez avoir les implémentations en mémoire de votre contexte et les jeux créés dynamiquement au moment de l'exécution.

Cet article traite de la création de votre propre double de test. Pour plus d'informations sur l'utilisation d'une infrastructure fictive, consultez [test avec un Framework fictif](#).

## Test avec les versions antérieures à EF6

Le code présenté dans cet article est compatible avec EF6. Pour tester avec EF5 et les versions antérieures, consultez [test avec un contexte factice](#).

## Limitations des doubles de test en mémoire EF

Les doubles de test en mémoire peuvent être un bon moyen de fournir une couverture de niveau test unitaire des bits de votre application qui utilisent EF. Toutefois, lorsque vous procédez ainsi, vous utilisez LINQ to Objects pour exécuter des requêtes sur des données en mémoire. Cela peut entraîner un comportement différent de celui de l'utilisation du fournisseur LINQ d'EF (LINQ to Entities) pour convertir les requêtes en SQL exécuté sur votre base de données.

Le chargement de données associées est un exemple de cette différence. Si vous créez une série de blogs qui ont chacun des publications associées, lorsque vous utilisez des données en mémoire, les publications associées sont toujours chargées pour chaque blog. Toutefois, lors de l'exécution sur une base de données, les données sont chargées uniquement si vous utilisez la méthode `Include`.

Pour cette raison, il est recommandé de toujours inclure un certain niveau de test de bout en bout (en plus de vos tests unitaires) pour garantir le bon fonctionnement de votre application sur une base de données.

## Suivre les étapes de cet article

Cet article fournit des listes de code complètes que vous pouvez copier dans Visual Studio pour suivre la procédure si vous le souhaitez. Il est plus facile de créer un **projet de test unitaire** et vous devrez cibler **.NET Framework 4.5** pour terminer les sections qui utilisent Async.

## Création d'une interface de contexte

Nous allons examiner le test d'un service qui utilise un modèle EF. Pour pouvoir remplacer notre contexte EF par une version en mémoire à des fins de test, nous allons définir une interface qui sera implémentée par notre contexte EF (et c'est en mémoire double).

Le service que nous allons tester interroge et modifie les données à l'aide des propriétés DbSet de notre contexte et appelle également SaveChanges pour envoyer les modifications à la base de données. Nous incluons donc ces membres sur l'interface.

```
using System.Data.Entity;

namespace TestingDemo
{
    public interface IBloggingContext
    {
        DbSet<Blog> Blogs { get; }
        DbSet<Post> Posts { get; }
        int SaveChanges();
    }
}
```

## Le modèle EF

Le service que nous allons tester utilise un modèle EF constitué du BloggingContext et du blog et des classes de publication. Ce code peut avoir été généré par le concepteur EF ou être un modèle de Code First.

```

using System.Collections.Generic;
using System.Data.Entity;

namespace TestingDemo
{
    public class BloggingContext : DbContext, IBloggingContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
        public string Url { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }
}

```

## Implémentation de l'interface de contexte à l'aide du concepteur EF

Notez que notre contexte implémente l'interface IBloggingContext.

Si vous utilisez Code First vous pouvez modifier votre contexte directement pour implémenter l'interface. Si vous utilisez le concepteur EF, vous devez modifier le modèle T4 qui génère votre contexte. Ouvrez le <. Fichier Context.tt imbriqué sous votre fichier edmx, recherchez le fragment de code suivant et ajoutez-le dans l'interface, comme indiqué.

```
<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : DbContext, IBloggingContext
```

## Service à tester

Pour illustrer le test avec des doubles de test en mémoire, nous allons écrire deux tests pour un BlogService. Le service est en charge de la création de nouveaux blogs (AddBlog) et de la restauration de tous les blogs classés par nom (GetAllBlogs). En plus de GetAllBlogs, nous avons également fourni une méthode qui permet d'obtenir de façon asynchrone tous les blogs classés par nom (GetAllBlogsAsync).

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    public class BlogService
    {
        private IBloggingContext _context;

        public BlogService(IBloggingContext context)
        {
            _context = context;
        }

        public Blog AddBlog(string name, string url)
        {
            var blog = new Blog { Name = name, Url = url };
            _context.Blogs.Add(blog);
            _context.SaveChanges();

            return blog;
        }

        public List<Blog> GetAllBlogs()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return query.ToList();
        }

        public async Task<List<Blog>> GetAllBlogsAsync()
        {
            var query = from b in _context.Blogs
                        orderby b.Name
                        select b;

            return await query.ToListAsync();
        }
    }
}

```

## Création des doubles de test en mémoire

Maintenant que nous disposons du véritable modèle EF et du service qui peut l'utiliser, il est temps de créer le double de test en mémoire que nous pouvons utiliser pour le test. Nous avons créé un double de test `DbContext` pour notre contexte. Dans les doubles de test, nous avons choisi le comportement que nous souhaitons pour prendre en charge les tests que nous allons exécuter. Dans cet exemple, nous capturons simplement le nombre de fois que `SaveChanges` est appelé, mais vous pouvez inclure la logique nécessaire pour vérifier le scénario que vous testez.

Nous avons également créé un `TestDbSet` qui fournit une implémentation en mémoire de `DbSet`. Nous avons fourni une implémentation complète pour toutes les méthodes sur `DbSet` (à l'exception de `Find`), mais vous devez uniquement implémenter les membres que votre scénario de test utilisera.

`TestDbSet` utilise d'autres classes d'infrastructure que nous avons incluses pour s'assurer que les requêtes asynchrones peuvent être traitées.

```
using System;
```

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Linq.Expressions;
using System.Threading;
using System.Threading.Tasks;

namespace TestingDemo
{
    public class TestContext : IBloggingContext
    {
        public TestContext()
        {
            this.Blogs = new TestDbSet<Blog>();
            this.Posts = new TestDbSet<Post>();
        }

        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
        public int SaveChangesCount { get; private set; }
        public int SaveChanges()
        {
            this.SaveChangesCount++;
            return 1;
        }
    }

    public class TestDbSet<TEntity> : DbSet<TEntity>, IQueryble, IEnumerable<TEntity>,
        IDbAsyncEnumerable<TEntity>
        where TEntity : class
    {
        ObservableCollection<TEntity> _data;
        IQueryble _query;

        public TestDbSet()
        {
            _data = new ObservableCollection<TEntity>();
            _query = _data.AsQueryable();
        }

        public override TEntity Add(TEntity item)
        {
            _data.Add(item);
            return item;
        }

        public override TEntity Remove(TEntity item)
        {
            _data.Remove(item);
            return item;
        }

        public override TEntity Attach(TEntity item)
        {
            _data.Add(item);
            return item;
        }

        public override TEntity Create()
        {
            return Activator.CreateInstance<TEntity>();
        }

        public override TDerivedEntity Create<TDerivedEntity>()
        {
            return Activator.CreateInstance<TDerivedEntity>();
        }
    }
}
```

```
public override ObservableCollection< TEntity> Local
{
    get { return _data; }
}

Type IQueryable.ElementType
{
    get { return _query.ElementType; }
}

Expression IQueryable.Expression
{
    get { return _query.Expression; }
}

IQueryProvider IQueryable.Provider
{
    get { return new TestDbAsyncQueryProvider< TEntity>(_query.Provider); }
}

System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
{
    return _data.GetEnumerator();
}

IEnumerator< TEntity> IEnumerable< TEntity>.GetEnumerator()
{
    return _data.GetEnumerator();
}

IDbAsyncEnumerator< TEntity> IDbAsyncEnumerable< TEntity>.GetAsyncEnumerator()
{
    return new TestDbAsyncEnumerator< TEntity>(_data.GetEnumerator());
}
}

internal class TestDbAsyncQueryProvider< TEntity> : IDbAsyncQueryProvider
{
    private readonly IQueryProvider _inner;

    internal TestDbAsyncQueryProvider(IQueryProvider inner)
    {
        _inner = inner;
    }

    public IQueryable CreateQuery(Expression expression)
    {
        return new TestDbAsyncEnumerable< TEntity>(expression);
    }

    public IQueryable< TElement> CreateQuery< TElement>(Expression expression)
    {
        return new TestDbAsyncEnumerable< TElement>(expression);
    }

    public object Execute(Expression expression)
    {
        return _inner.Execute(expression);
    }

    public TResult Execute< TResult>(Expression expression)
    {
        return _inner.Execute< TResult>(expression);
    }

    public Task< object> ExecuteAsync(Expression expression, CancellationToken cancellationToken)
    {
        return Task.FromResult(Execute(expression));
    }
}
```

```

    }

    public Task<TResult> ExecuteAsync<TResult>(Expression expression, CancellationToken cancellationToken)
    {
        return Task.FromResult(Execute<TResult>(expression));
    }
}

internal class TestDbAsyncEnumerable<T> : EnumerableQuery<T>, IDbAsyncEnumerable<T>, IQueryable<T>
{
    public TestDbAsyncEnumerable(IEnumerable<T> enumerable)
        : base(enumerable)
    { }

    public TestDbAsyncEnumerable(Expression expression)
        : base(expression)
    { }

    public IDbAsyncEnumerator<T> GetAsyncEnumerator()
    {
        return new TestDbAsyncEnumerator<T>(this.AsEnumerable().GetEnumerator());
    }

    IDbAsyncEnumerator IDbAsyncEnumerable.GetAsyncEnumerator()
    {
        return GetAsyncEnumerator();
    }

    IQueryProvider IQueryable.Provider
    {
        get { return new TestDbAsyncQueryProvider<T>(this); }
    }
}

internal class TestDbAsyncEnumerator<T> : IDbAsyncEnumerator<T>
{
    private readonly IEnumerator<T> _inner;

    public TestDbAsyncEnumerator(IEnumerator<T> inner)
    {
        _inner = inner;
    }

    public void Dispose()
    {
        _inner.Dispose();
    }

    public Task<bool> MoveNextAsync(CancellationToken cancellationToken)
    {
        return Task.FromResult(_inner.MoveNext());
    }

    public T Current
    {
        get { return _inner.Current; }
    }

    object IDbAsyncEnumerator.Current
    {
        get { return Current; }
    }
}
}

```

## Implémentation de Find

La méthode Find est difficile à implémenter de manière générique. Si vous devez tester du code qui utilise la

méthode Find, il est plus facile de créer un DbSet de test pour chacun des types d'entité devant prendre en charge Find. Vous pouvez ensuite écrire une logique pour rechercher ce type particulier d'entité, comme indiqué ci-dessous.

```
using System.Linq;

namespace TestingDemo
{
    class TestBlogDbSet : TestDbSet<Blog>
    {
        public override Blog Find(params object[] keyValues)
        {
            var id = (int)keyValues.Single();
            return this.SingleOrDefault(b => b.BlogId == id);
        }
    }
}
```

## Écrire des tests

C'est tout ce dont nous avons besoin pour commencer le test. Le test suivant crée un TestContext, puis un service basé sur ce contexte. Le service est ensuite utilisé pour créer un blog, à l'aide de la méthode AddBlog. Enfin, le test vérifie que le service a ajouté un nouveau blog à la propriété blogs du contexte et a appelé SaveChanges sur le contexte.

Il s'agit simplement d'un exemple des types de choses que vous pouvez tester avec un double de test en mémoire et vous pouvez ajuster la logique des doubles de test et la vérification pour répondre à vos besoins.

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;

namespace TestingDemo
{
    [TestClass]
    public class NonQueryTests
    {
        [TestMethod]
        public void CreateBlog_saves_a_blog_via_context()
        {
            var context = new TestContext();

            var service = new BlogService(context);
            service.AddBlog("ADO.NET Blog", "http://blogs.msdn.com/adonet");

            Assert.AreEqual(1, context.Blogs.Count());
            Assert.AreEqual("ADO.NET Blog", context.Blogs.Single().Name);
            Assert.AreEqual("http://blogs.msdn.com/adonet", context.Blogs.Single().Url);
            Assert.AreEqual(1, context.SaveChangesCount());
        }
    }
}
```

Voici un autre exemple de test : cette fois-ci, il exécute une requête. Le test commence par la création d'un contexte de test avec des données dans sa propriété de blog. Notez que les données ne sont pas classées par ordre alphabétique. Nous pouvons ensuite créer un BlogService basé sur notre contexte de test et vous assurer que les données renvoyées par GetAllBlogs sont classées par nom.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace TestingDemo
{
    [TestClass]
    public class QueryTests
    {
        [TestMethod]
        public void GetAllBlogs_orders_by_name()
        {
            var context = new TestContext();
            context.Blogs.Add(new Blog { Name = "BBB" });
            context.Blogs.Add(new Blog { Name = "ZZZ" });
            context.Blogs.Add(new Blog { Name = "AAA" });

            var service = new BlogService(context);
            var blogs = service.GetAllBlogs();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

Enfin, nous allons écrire un test supplémentaire qui utilise notre méthode Async pour s'assurer que l'infrastructure Async que nous avons incluse dans [TestDbSet](#) fonctionne.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace TestingDemo
{
    [TestClass]
    public class AsyncQueryTests
    {
        [TestMethod]
        public async Task GetAllBlogsAsync_orders_by_name()
        {
            var context = new TestContext();
            context.Blogs.Add(new Blog { Name = "BBB" });
            context.Blogs.Add(new Blog { Name = "ZZZ" });
            context.Blogs.Add(new Blog { Name = "AAA" });

            var service = new BlogService(context);
            var blogs = await service.GetAllBlogsAsync();

            Assert.AreEqual(3, blogs.Count);
            Assert.AreEqual("AAA", blogs[0].Name);
            Assert.AreEqual("BBB", blogs[1].Name);
            Assert.AreEqual("ZZZ", blogs[2].Name);
        }
    }
}

```

# Testabilité et Entity Framework 4,0

23/11/2019 • 84 minutes to read

Scott Allen

Date de publication : mai 2010

## Introduction

Ce livre blanc décrit et montre comment écrire du code testable avec ADO.NET Entity Framework 4,0 et Visual Studio 2010. Ce document ne tente pas de se concentrer sur une méthodologie de test spécifique, telle que la conception pilotée par test (TDD) ou la conception pilotée par comportement (BDD). Au lieu de cela, cet article se concentrera sur la façon d'écrire du code qui utilise le ADO.NET Entity Framework tout en restant facile à isoler et à tester de manière automatisée. Nous allons examiner les modèles de conception courants qui facilitent les tests dans les scénarios d'accès aux données et comment appliquer ces modèles quand vous utilisez l'infrastructure. Nous examinerons également les fonctionnalités spécifiques de l'infrastructure pour voir comment ces fonctionnalités peuvent fonctionner dans du code testable.

## Qu'est-ce que le code testable ?

La possibilité de vérifier un logiciel à l'aide de tests unitaires automatisés offre de nombreux avantages souhaitables. Tout le monde sait que les bons tests vont réduire le nombre de défauts logiciels dans une application et augmenter la qualité de l'application, mais que les tests unitaires sont en place bien plus que de trouver des bogues.

Une bonne suite de tests unitaires permet à une équipe de développement de gagner du temps et de garder le contrôle du logiciel qu'elle crée. Une équipe peut apporter des modifications au code existant, Refactoriser, reconcevoir et restructurer des logiciels pour répondre à de nouvelles exigences, et ajouter de nouveaux composants dans une application tout en sachant que la suite de tests peut vérifier le comportement de l'application. Les tests unitaires font partie d'un cycle de commentaires rapide pour faciliter les modifications et préserver la maintenabilité des logiciels à mesure que la complexité augmente.

Toutefois, les tests unitaires sont fournis avec un prix. Une équipe doit consacrer le temps nécessaire à la création et à la maintenance des tests unitaires. L'effort requis pour créer ces tests est directement lié à la **testabilité** du logiciel sous-jacent. Le logiciel est-il facile à tester ? Une équipe qui développe des logiciels avec la testabilité à l'esprit créera des tests efficaces plus rapidement que l'équipe travaillant avec des logiciels non testables.

Microsoft a conçu le ADO.NET Entity Framework 4,0 (EF4) avec la testabilité à l'esprit. Cela ne signifie pas que les développeurs écrivent des tests unitaires sur le code du Framework lui-même. Au lieu de cela, les objectifs de testabilité pour EF4 facilitent la création de code testable qui s'appuie sur l'infrastructure. Avant d'examiner des exemples spécifiques, il est utile de comprendre les qualités du code testable.

### Les qualités du code testable

Le code facile à tester présentera toujours au moins deux traits. Tout d'abord, le code testable est facile à **observer**. Étant donné un ensemble d'entrées, il doit être facile d'observer la sortie du code. Par exemple, le test de la méthode suivante est simple, car la méthode retourne directement le résultat d'un calcul.

```
public int Add(int x, int y) {
    return x + y;
}
```

Le test d'une méthode est difficile si la méthode écrit la valeur calculée dans un socket réseau, une table de base de données ou un fichier comme le code suivant. Le test doit effectuer un travail supplémentaire pour récupérer la valeur.

```
public void AddAndSaveToFile(int x, int y) {
    var results = string.Format("The answer is {0}", x + y);
    File.WriteAllText("results.txt", results);
}
```

Deuxièmement, il est facile d'**isoler** du code testable. Utilisons le pseudo-code suivant comme mauvais exemple de code testable.

```
public int ComputePolicyValue(InsurancePolicy policy) {
    using (var connection = new SqlConnection("dbConnection"))
        using (var command = new SqlCommand(query, connection)) {

            // business calculations omitted ...

            if (totalValue > notificationThreshold) {
                var message = new MailMessage();
                message.Subject = "Warning!";
                var client = new SmtpClient();
                client.Send(message);
            }
        }
    return totalValue;
}
```

La méthode est facile à observer : nous pouvons transmettre une stratégie d'assurance et vérifier que la valeur de retour correspond à un résultat attendu. Toutefois, pour tester la méthode, vous devez disposer d'une base de données installée avec le schéma correct, et configurer le serveur SMTP en cas de tentative d'envoi d'un e-mail par la méthode.

Le test unitaire souhaite uniquement vérifier la logique de calcul à l'intérieur de la méthode, mais le test peut échouer parce que le serveur de messagerie est hors connexion ou que le serveur de base de données a été déplacé. Ces deux échecs ne sont pas liés au comportement que le test veut vérifier. Le comportement est difficile à isoler.

Les développeurs de logiciels qui cherchent à écrire du code testable s'efforcent souvent de conserver une séparation des préoccupations dans le code qu'ils écrivent. La méthode ci-dessus doit se concentrer sur les calculs d'entreprise et déléguer les détails d'implémentation de la base de données et du courrier électronique à d'autres composants. Robert C. Martin appelle ce principe de responsabilité unique. Un objet doit encapsuler une seule responsabilité étroite, comme le calcul de la valeur d'une stratégie. Toutes les autres opérations de base de données et de notification doivent être de la responsabilité d'un autre objet. Le code écrit de cette manière est plus facile à isoler, car il est axé sur une seule tâche.

Dans .NET, nous disposons des abstractions dont nous avons besoin pour suivre le principe de responsabilité unique et parvenir à l'isolation. Nous pouvons utiliser des définitions d'interface et forcer le code à utiliser l'abstraction d'interface au lieu d'un type concret. Plus loin dans ce document, nous verrons comment une méthode telle que l'exemple mal présenté ci-dessus peut fonctionner avec des interfaces qui *semblent* communiquer avec la base de données. Toutefois, au moment du test, nous pouvons remplacer une implémentation factice qui ne communique pas avec la base de données, mais qui contient à la place des données en mémoire. Cette implémentation factice isole le code des problèmes non liés dans le code d'accès aux données ou la configuration de la base de données.

L'isolation présente des avantages supplémentaires. L'exécution du calcul de l'activité dans la dernière méthode ne doit prendre que quelques millisecondes, mais le test lui-même peut s'exécuter pendant plusieurs secondes à

mesure que le code saute sur le réseau et communique avec différents serveurs. Les tests unitaires doivent s'exécuter rapidement pour faciliter les petites modifications. Les tests unitaires doivent également être reproductibles et ne pas échouer car un composant qui n'est pas lié au test a un problème. L'écriture de code facile à observer et à isoler signifie que les développeurs auront un temps plus facile pour écrire des tests pour le code, passer moins de temps à attendre l'exécution des tests et, plus important encore, passer moins de temps à suivre les bogues qui n'existent pas.

Nous espérons que vous pouvez apprécier les avantages des tests et comprendre les qualités qu'il présente. Nous sommes sur le point de traiter l'écriture de code qui fonctionne avec EF4 pour enregistrer des données dans une base de données tout en restant observables et faciles à isoler, mais tout d'abord, nous allons affiner l'étude des conceptions pouvant être testées pour l'accès aux données.

## Modèles de conception pour la persistance des données

Les deux exemples incorrects présentés précédemment avaient trop de responsabilités. Le premier exemple inappropriate devait effectuer un calcul et écrire dans un fichier. Le deuxième exemple incorrect devait lire les données d'une base de données et effectuer un calcul d'entreprise et envoyer des e-mails. En concevant des méthodes plus petites qui distinguent les préoccupations et délèguent la responsabilité à d'autres composants, vous ferez de superbes progrès pour écrire du code testable. L'objectif est de créer des fonctionnalités en composant des actions à partir d'abstractions petites et concentrées.

Lorsqu'il s'agit de la persistance des données, les petites abstractions concentrées que nous recherchons sont tellement courantes qu'elles ont été documentées en tant que modèles de conception. Les modèles de l'architecture d'applications d'entreprise de Martin Fowler étaient le premier travail à décrire ces modèles à l'impression. Nous fournissons une brève description de ces modèles dans les sections suivantes avant de montrer comment ces ADO.NET Entity Framework implémentent et fonctionnent avec ces modèles.

### The Repository Pattern

Fowler indique un référentiel qui suit les couches de mappage de données et de domaine à l'aide d'une interface de type collection pour accéder aux objets de domaine. L'objectif du modèle de référentiel est d'isoler le code du détails d'accès aux données, et comme nous l'avons vu, l'isolation précédente est un critère requis pour la testabilité.

La clé de l'isolation est la façon dont le référentiel expose les objets à l'aide d'une interface de type collection. La logique que vous écrivez pour utiliser le référentiel n'a aucune idée de la façon dont le référentiel matérialisera les objets que vous demandez. Le référentiel peut communiquer avec une base de données, ou il peut simplement retourner des objets à partir d'une collection en mémoire. Tout votre code doit savoir que le référentiel semble gérer la collection et que vous pouvez récupérer, ajouter et supprimer des objets de la collection.

Dans les applications .NET existantes, un référentiel concret hérite souvent d'une interface générique comme suit :

```
public interface IRepository<T> {
    IEnumerable<T> FindAll();
    IEnumerable<T> FindBy(Expression<Func<T, bool>> predicate);
    T FindById(int id);
    void Add(T newEntity);
    void Remove(T entity);
}
```

Nous allons apporter quelques modifications à la définition de l'interface lorsque nous fournissons une implémentation pour EF4, mais le concept de base reste le même. Le code peut utiliser un référentiel concret qui implémente cette interface pour récupérer une entité par sa valeur de clé primaire, pour récupérer une collection d'entités en fonction de l'évaluation d'un prédictat, ou simplement récupérer toutes les entités disponibles. Le code peut également ajouter et supprimer des entités par le biais de l'interface de référentiel.

Étant donné un IRepository d'objets Employee, le code peut effectuer les opérations suivantes.

```

var employeesNamedScott =
    repository
        .FindBy(e => e.Name == "Scott")
        .OrderBy(e => e.HireDate);
var firstEmployee = repository.FindById(1);
var newEmployee = new Employee() { /*... */};
repository.Add(newEmployee);

```

Étant donné que le code utilise une interface (`IRepository` of `Employee`), nous pouvons fournir le code avec différentes implémentations de l'interface. Une implémentation peut être une implémentation sauvegardée par EF4 et la persistance d'objets dans une base de données Microsoft SQL Server. Une implémentation différente (celle que nous utilisons pendant le test) peut être sauvegardée par une liste en mémoire d'objets `Employee`. L'interface permet d'obtenir un isolement dans le code.

Notez que l'interface `IRepository<T>` n'expose pas d'opération d'enregistrement. Comment mettre à jour les objets existants ? Vous pouvez vous trouver parmi les définitions de `IRepository` qui incluent l'opération d'enregistrement, et les implémentations de ces dépôts devront conserver immédiatement un objet dans la base de données. Toutefois, dans de nombreuses applications, nous ne souhaitons pas conserver les objets individuellement. Au lieu de cela, nous voulons donner vie aux objets, peut-être à partir de différents référentiels, modifier ces objets dans le cadre d'une activité d'entreprise, puis conserver tous les objets dans le cadre d'une opération atomique unique. Heureusement, il existe un modèle pour autoriser ce type de comportement.

## Modèle d'unité de travail

Fowler dit qu'une unité de travail conservera une liste d'objets affectés par une transaction commerciale et coordonne l'écriture des modifications et la résolution des problèmes d'accès concurrentiel. Il incombe à l'unité de travail de suivre les modifications apportées aux objets que nous avons apportées à la vie à partir d'un référentiel et de conserver les modifications que nous avons apportées aux objets quand nous indiquons à l'unité de travail de valider les modifications. Il incombe également à l'unité de travail de prendre les nouveaux objets que nous avons ajoutés à tous les référentiels, d'insérer les objets dans une base de données et de gérer également la suppression.

Si vous avez déjà effectué des tâches avec des jeux de données ADO.NET, vous êtes déjà familiarisé avec le modèle d'unité de travail. Les jeux de données ADO.NET pouvaient effectuer le suivi de nos mises à jour, suppressions et insertions d'objets `DataRow` et pouvaient (à l'aide d'un `TableAdapter`) réconcilier toutes les modifications apportées à une base de données. Toutefois, les objets `DataSet` modélisent un sous-ensemble déconnecté de la base de données sous-jacente. Le modèle unité de travail présente le même comportement, mais fonctionne avec les objets métier et les objets de domaine qui sont isolés du code d'accès aux données et ne connaissent pas la base de données.

Une abstraction pour modéliser l'unité de travail dans du code .NET peut se présenter comme suit :

```

public interface IUnitOfWork {
    IRepository<Employee> Employees { get; }
    IRepository<Order> Orders { get; }
    IRepository<Customer> Customers { get; }
    void Commit();
}

```

En exposant les références de référentiel à partir de l'unité de travail, nous pouvons garantir qu'un seul objet d'unité de travail a la possibilité de suivre toutes les entités matérialisées lors d'une transaction commerciale. L'implémentation de la méthode `Commit` pour une unité de travail réelle est l'endroit où tout le Magic se produit pour rapprocher les modifications en mémoire avec la base de données.

À partir d'une référence `IUnitOfWork`, le code peut apporter des modifications aux objets métier récupérés à partir d'un ou de plusieurs référentiels et enregistrer toutes les modifications à l'aide de l'opération de validation atomique.

```
var firstEmployee = unitofWork.Employees.FindById(1);
var firstCustomer = unitofWork.Customers.FindById(1);
firstEmployee.Name = "Alex";
firstCustomer.Name = "Christopher";
unitofWork.Commit();
```

## Le modèle de chargement différé

Fowler utilise la charge différée de nom pour décrire « un objet qui ne contient pas toutes les données dont vous avez besoin, mais qui sait comment l'accéder ». Le chargement différé transparent est une fonctionnalité importante pour l'écriture de code d'entreprise testable et l'utilisation d'une base de données relationnelle. À titre d'exemple, considérez le code suivant.

```
var employee = repository.FindById(id);
// ... and later ...
foreach(var timeCard in employee.TimeCards) {
    // .. manipulate the timeCard
}
```

Comment la collection des entrées de procédure est-elle remplie ? Il existe deux réponses possibles. L'une des réponses est que le dépôt de l'employé, lorsqu'il est invité à extraire un employé, émet une requête pour récupérer à la fois l'employé et les informations de carte de l'employé associées. Dans les bases de données relationnelles, cela nécessite généralement une requête avec une clause JOIN et peut entraîner la récupération d'informations supplémentaires par rapport aux besoins d'une application. Que se passe-t-il si l'application n'a jamais besoin de toucher la propriété de la fonction de la

Une deuxième réponse consiste à charger la propriété de la fonction de la requête « à la demande ». Ce chargement différé est implicite et transparent pour la logique métier, car le code n'appelle pas d'API spéciales pour récupérer les informations de carte de temps. Le code suppose que les informations de carte de temps sont présentes si nécessaire. Une magie est impliquée dans le chargement différé qui implique généralement l'interception de l'exécution d'appels de méthode. Le code d'interception est chargé de communiquer avec la base de données et de récupérer les informations de la carte de temps tout en laissant la logique métier libre pour être logique métier. Cette magie de chargement différé permet au code d'entreprise de s'isoler des opérations d'extraction de données et de se traduire par un code plus testable.

L'inconvénient d'une charge différée est que lorsqu'une application a besoin des informations de carte de temps, le code exécutera une requête supplémentaire. Ce n'est pas un problème pour de nombreuses applications, mais pour les applications sensibles aux performances, en boucle sur un certain nombre d'objets Employee et en exécutant une requête pour récupérer des cartes de temps lors de chaque itération de la boucle (un problème souvent appelé N + 1 problème de requête), le chargement différé est un glissement. Dans ces scénarios, une application peut souhaiter charger de manière dynamique les informations de carte de temps de la manière la plus efficace possible.

Heureusement, nous allons voir comment EF4 prend en charge à la fois les charges tardives implicites et les charges hâtif efficaces à mesure que nous passons à la section suivante et implémentons ces modèles.

## Implémentation de modèles avec l'Entity Framework

La bonne nouvelle, c'est que tous les modèles de conception que nous avons décrits dans la dernière section sont faciles à implémenter avec EF4. Pour démontrer que nous allons utiliser une simple application MVC ASP.NET pour modifier et afficher les employés et leurs informations de carte de temps associées. Nous allons commencer par utiliser les objets « Plain Old CLR Objects » (POCO) suivants.

```

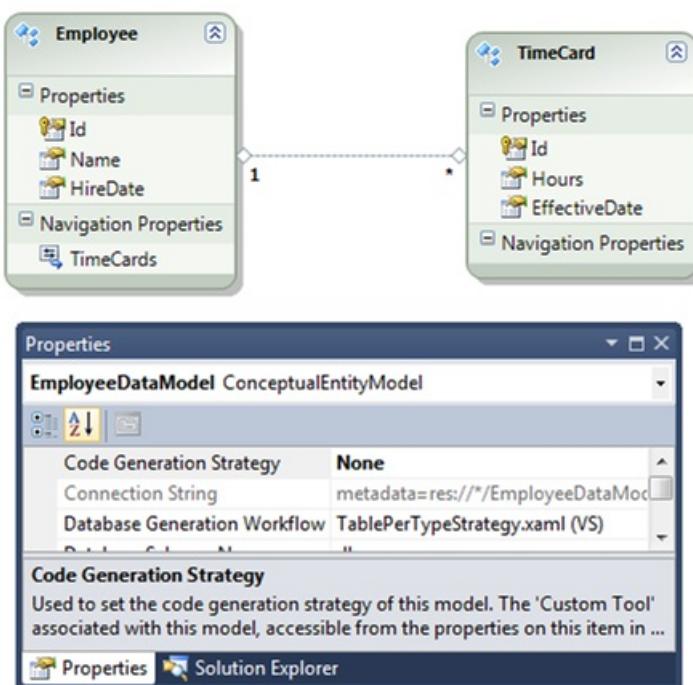
public class Employee {
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime HireDate { get; set; }
    public ICollection<TimeCard> TimeCards { get; set; }
}

public class TimeCard {
    public int Id { get; set; }
    public int Hours { get; set; }
    public DateTime EffectiveDate { get; set; }
}

```

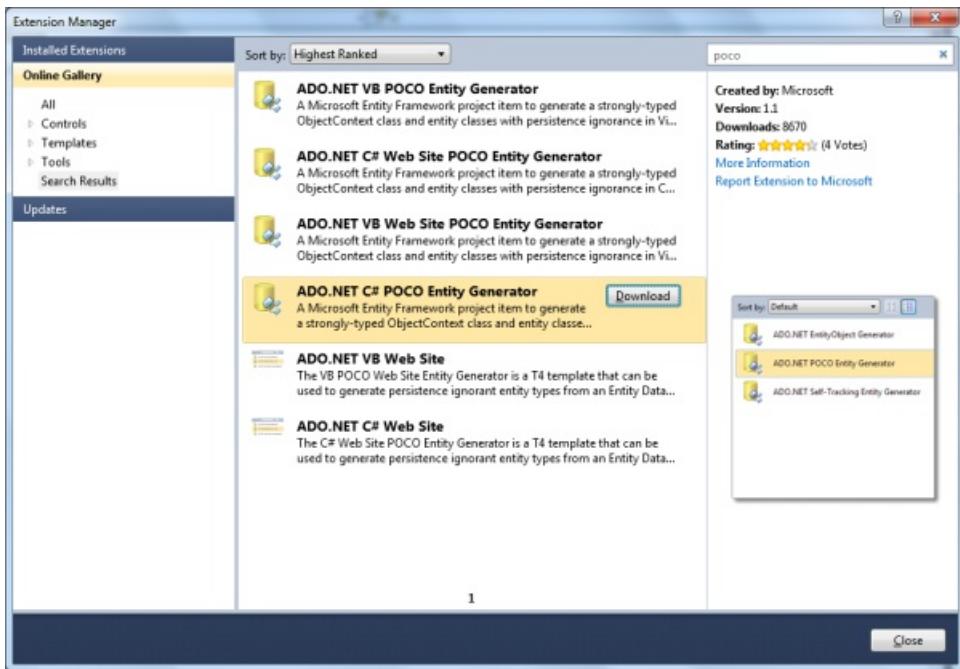
Ces définitions de classe seront légèrement modifiées à mesure que nous explorerons les différentes approches et fonctionnalités de EF4, mais l'objectif est de conserver ces classes comme étant le plus possible (PI). Un objet PI ne sait pas *Comment*, ou même *si*, l'État qu'il contient réside dans une base de données. PI et POCO vont de pair avec des logiciels testables. Les objets utilisant une approche POCO sont moins limités, plus flexibles et plus faciles à tester, car ils peuvent fonctionner sans une base de données présente.

Une fois les POCO en place, nous pouvons créer un Entity Data Model (EDM) dans Visual Studio (voir la figure 1). Nous n'utiliserons pas le modèle EDM pour générer du code pour nos entités. Au lieu de cela, nous souhaitons utiliser les entités qui nous intéressent à la main. Nous n'utiliserons le modèle EDM que pour générer le schéma de base de données et fournir les métadonnées dont EF4 a besoin pour mapper les objets dans la base de données.



**Figure 1**

Remarque : Si vous souhaitez développer le modèle EDM en premier, il est possible de générer du code POCO propre à partir du modèle EDM. Vous pouvez le faire avec une extension Visual Studio 2010 fournie par l'équipe de programmabilité des données. Pour télécharger l'extension, lancez le gestionnaire d'extensions à partir du menu outils de Visual Studio et recherchez « POCO » dans la galerie en ligne de modèles (voir figure 2). Plusieurs modèles POCO sont disponibles pour EF. Pour plus d'informations sur l'utilisation du modèle, consultez la rubrique « [procédure pas à pas : modèle POCO pour le Entity Framework](#)».



**Figure 2**

À partir de ce point de départ POCO, nous explorerons deux approches différentes du code testable. La première approche que j'appelle l'approche EF, c'est qu'elle tire parti des abstractions de l'API Entity Framework pour implémenter des unités de travail et des dépôts. Dans la deuxième approche, nous allons créer nos propres abstractions de référentiel personnalisées, puis voir les avantages et les inconvénients de chaque approche. Nous allons commencer par explorer l'approche EF.

### Une implémentation d'EF centrée

Examinez l'action de contrôleur suivante à partir d'un projet MVC ASP.NET. L'action extrait un objet Employee et retourne un résultat pour afficher une vue détaillée de l'employé.

```
public ViewResult Details(int id) {
    var employee = _unitOfWork.Employees
        .Single(e => e.Id == id);
    return View(employee);
}
```

Le code est-il testable ? Il faut au moins deux tests pour vérifier le comportement de l'action. Tout d'abord, nous souhaitons vérifier que l'action retourne la bonne vue, un test facile. Nous aimerais également écrire un test pour vérifier que l'action récupère le bon employé et nous aimerais le faire sans exécuter de code pour interroger la base de données. N'oubliez pas que nous souhaitons isoler le code testé. L'isolation garantit que le test n'échoue pas en raison d'un bogue dans le code d'accès aux données ou la configuration de la base de données. Si le test échoue, nous savons que nous avons un bogue dans la logique du contrôleur, et non dans un composant système de niveau inférieur.

Pour atteindre l'isolation, nous avons besoin de certaines abstractions comme les interfaces présentées précédemment pour les dépôts et les unités de travail. N'oubliez pas que le modèle de référentiel est conçu pour faire l'objet d'un médiateur entre les objets de domaine et la couche de mappage de données. Dans ce scénario, EF4 est la couche de mappage des données et fournit déjà une abstraction de type dépôt nommée `IObjectSet<t>` (à partir de l'espace de noms `System.Data.Objects`). La définition de l'interface se présente comme suit.

```

public interface IObjectSet<TEntity> :
    IQueryable<TEntity>,
    IEnumerable<TEntity>,
    IQueryable,
    IEnumerable
    where TEntity : class
{
    void AddObject(TEntity entity);
    void Attach(TEntity entity);
    void DeleteObject(TEntity entity);
    void Detach(TEntity entity);
}

```

IObjectSet<T> répond à la configuration requise pour un référentiel, car il ressemble à une collection d'objets (via IEnumerable<T>) et fournit des méthodes pour ajouter et supprimer des objets de la collection simulée. Les méthodes d'attachement et de détachement exposent des fonctionnalités supplémentaires de l'API EF4. Pour utiliser IObjectSet<T> comme interface pour les dépôts, nous avons besoin d'une abstraction d'unité de travail pour lier les référentiels ensemble.

```

public interface IUnitOfWork {
    IObjectSet<Employee> Employees { get; }
    IObjectSet<TimeCard> TimeCards { get; }
    void Commit();
}

```

Une implémentation concrète de cette interface va communiquer avec SQL Server et est facile à créer à l'aide de la classe ObjectContext de EF4. La classe ObjectContext est l'unité de travail réelle dans l'API EF4.

```

public class SqlUnitOfWork : IUnitOfWork {
    public SqlUnitOfWork() {
        var connectionString =
            ConfigurationManager
                .ConnectionStrings[ConnectionStringName]
                .ConnectionString;
        _context = new ObjectContext(connectionString);
    }

    public IObjectSet<Employee> Employees {
        get { return _context.CreateObjectSet<Employee>(); }
    }

    public IObjectSet<TimeCard> TimeCards {
        get { return _context.CreateObjectSet<TimeCard>(); }
    }

    public void Commit() {
        _context.SaveChanges();
    }

    readonly ObjectContext _context;
    const string ConnectionStringName = "EmployeeDataModelContainer";
}

```

Il est aussi facile de placer un IObjectSet<T> à vie que d'appeler la méthode CreateObjectSet de l'objet ObjectContext. En arrière-plan, l'infrastructure utilisera les métadonnées que nous avons fournies dans le modèle EDM pour produire une>ObjectSet<T concrète. Nous allons utiliser le retour de l'interface IObjectSet<T>, car cela permet de préserver la testabilité du code client.

Cette implémentation concrète est utile en production, mais nous devons nous concentrer sur la façon dont nous allons utiliser notre abstraction IUnitOfWork pour faciliter les tests.

## Les doubles de test

Pour isoler l'action du contrôleur, nous avons besoin de la possibilité de basculer entre l'unité de travail réelle (avec un `ObjectContext`) et une unité de travail de test double ou « factice » (en effectuant des opérations en mémoire). L'approche courante pour effectuer ce type de commutation consiste à ne pas laisser le contrôleur MVC instancier une unité de travail, mais à passer à la place l'unité de travail dans le contrôleur en tant que paramètre de constructeur.

```
class EmployeeController : Controller {
    public EmployeeController(IUnitOfWork unitOfWork) {
        _unitOfWork = unitOfWork;
    }
    ...
}
```

Le code ci-dessus est un exemple d'injection de dépendances. Nous n'autorisons pas le contrôleur à créer sa dépendance (l'unité de travail), mais injectent la dépendance dans le contrôleur. Dans un projet MVC, il est courant d'utiliser une fabrique de contrôleurs personnalisée en association avec un conteneur d'inversion de contrôle (IoC) pour automatiser l'injection de dépendances. Ces rubriques n'entrent pas dans le cadre de cet article, mais vous pouvez en savoir plus en suivant les références à la fin de cet article.

Une implémentation fictive d'une unité de travail que nous pouvons utiliser pour le test peut ressembler à ce qui suit.

```
public class InMemoryUnitOfWork : IUnitOfWork {
    public InMemoryUnitOfWork() {
        Committed = false;
    }
    public IObjectSet<Employee> Employees {
        get;
        set;
    }

    public IObjectSet<TimeCard> TimeCards {
        get;
        set;
    }

    public bool Committed { get; set; }
    public void Commit() {
        Committed = true;
    }
}
```

Notez que l'unité de travail factice expose une propriété validée. Il est parfois utile d'ajouter des fonctionnalités à une classe factice qui facilite les tests. Dans ce cas, il est facile d'observer si le code valide une unité de travail en vérifiant la propriété validée.

Nous aurons également besoin d'un faux `IObjectSet<T>` pour contenir les objets `Employee` et la table de pointage en mémoire. Nous pouvons fournir une implémentation unique à l'aide de génériques.

```

public class InMemoryObjectSet<T> : IObjectSet<T> where T : class
{
    public InMemoryObjectSet()
        : this(Enumerable.Empty<T>()) {
    }

    public InMemoryObjectSet(IEnumerable<T> entities) {
        _set = new HashSet<T>();
        foreach (var entity in entities) {
            _set.Add(entity);
        }
        _queryableSet = _set.AsQueryable();
    }

    public void AddObject(T entity) {
        _set.Add(entity);
    }

    public void Attach(T entity) {
        _set.Add(entity);
    }

    public void DeleteObject(T entity) {
        _set.Remove(entity);
    }

    public void Detach(T entity) {
        _set.Remove(entity);
    }

    public Type ElementType {
        get { return _queryableSet.ElementType; }
    }

    public Expression Expression {
        get { return _queryableSet.Expression; }
    }

    public IQueryProvider Provider {
        get { return _queryableSet.Provider; }
    }

    public IEnumerator<T> GetEnumerator() {
        return _set.GetEnumerator();
    }

    Ienumerator IEnumerable.GetEnumerator() {
        return GetEnumerator();
    }

    readonly HashSet<T> _set;
    readonly IQueryable<T> _queryableSet;
}

```

Ce double de test délègue la majeure partie de son travail à un objet de `HashSet<T>` sous-jacent. Notez que `IObjectSet<T>` requiert une contrainte générique appliquant `T` en tant que classe (un type référence), et nous obligent à implémenter `IQueryable<T>`. Il est facile de faire apparaître une collection en mémoire sous la forme d'un `IQueryable<T>` à l'aide de l'opérateur standard LINQ `AsQueryable`.

## Les tests

Les tests unitaires traditionnels utilisent une classe de test unique pour contenir tous les tests pour toutes les actions dans un seul contrôleur MVC. Nous pouvons écrire ces tests, ou n'importe quel type de test unitaire, à l'aide des substituts en mémoire que nous avons générés. Toutefois, pour cet article, nous allons éviter l'approche de la classe de test monolithique et regrouper nos tests pour vous concentrer sur une fonctionnalité spécifique. Par exemple, « Create New Employee » peut être la fonctionnalité que nous voulons tester, donc nous utiliserons une seule classe de test pour vérifier l'action de contrôleur unique responsable de la création d'un nouvel employé.

Il existe un code d'installation commun dont nous avons besoin pour toutes ces classes de test affinées. Par exemple, nous devons toujours créer nos référentiels en mémoire et l'unité de travail factice. Nous avons également besoin d'une instance du contrôleur `Employee` avec l'unité de travail factice injectée. Nous allons partager ce code d'installation commun entre les classes de test à l'aide d'une classe de base.

```

public class EmployeeControllerTestBase {
    public EmployeeControllerTestBase() {
        _employeeData = EmployeeObjectMother.CreateEmployees()
            .ToList();
        _repository = new InMemoryObjectSet<Employee>(_employeeData);
        _unitOfWork = new InMemoryUnitOfWork();
        _unitOfWork.Employees = _repository;
        _controller = new EmployeeController(_unitOfWork);
    }

    protected IList<Employee> _employeeData;
    protected EmployeeController _controller;
    protected InMemoryObjectSet<Employee> _repository;
    protected InMemoryUnitOfWork _unitOfWork;
}

```

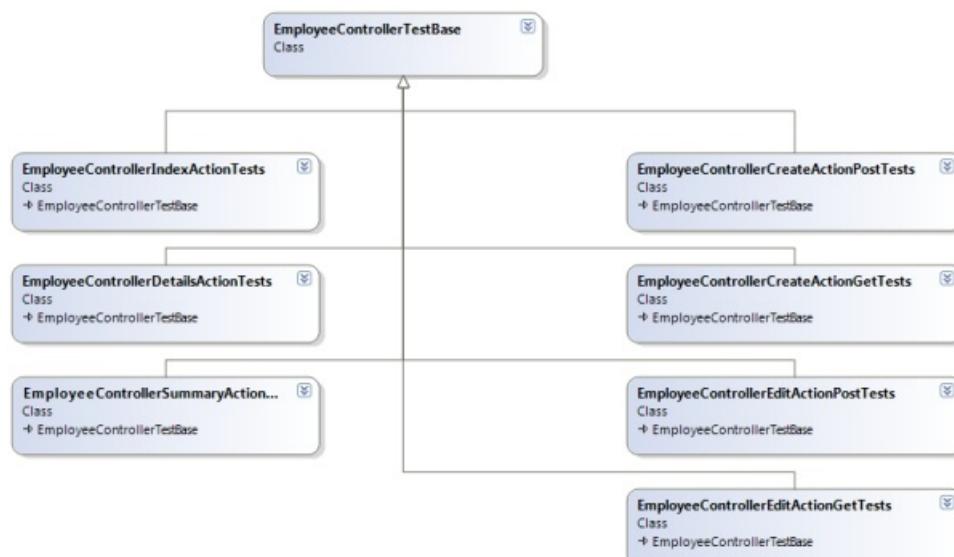
L'objet « maman » que nous utilisons dans la classe de base est un modèle commun pour la création de données de test. Un objet maman contient des méthodes de fabrique pour instancier des entités de test à utiliser sur plusieurs contextes de test.

```

public static class EmployeeObjectMother {
    public static IEnumerable<Employee> CreateEmployees() {
        yield return new Employee() {
            Id = 1, Name = "Scott", HireDate=new DateTime(2002, 1, 1)
        };
        yield return new Employee() {
            Id = 2, Name = "Poonam", HireDate=new DateTime(2001, 1, 1)
        };
        yield return new Employee() {
            Id = 3, Name = "Simon", HireDate=new DateTime(2008, 1, 1)
        };
    }
    // ... more fake data for different scenarios
}

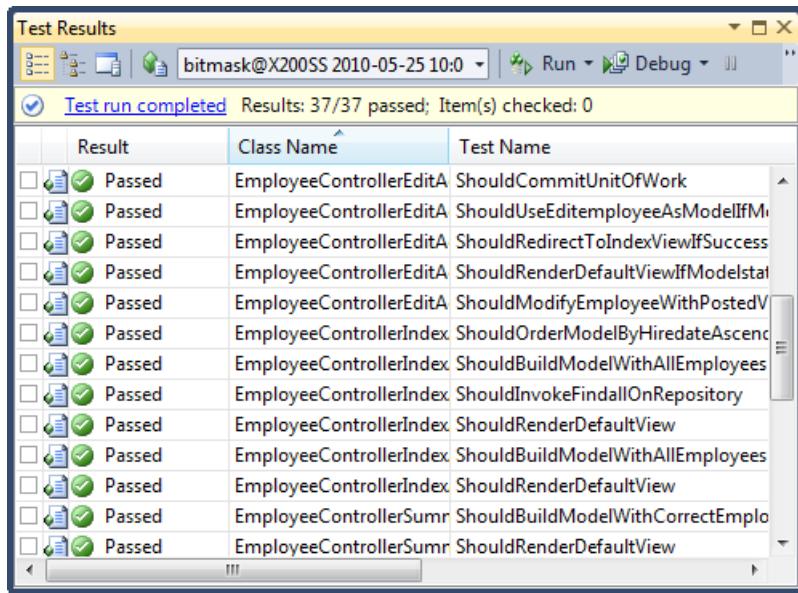
```

Nous pouvons utiliser `EmployeeControllerTestBase` comme classe de base pour un certain nombre de contextes de test (voir la figure 3). Chaque contexte de test teste une action de contrôleur spécifique. Par exemple, un seul contexte de test se concentrera sur le test de l'action de création utilisée lors d'une requête HTTP. (pour afficher la vue de la création d'un employé) et un autre contexte se concentrera sur l'action de création utilisée dans une requête HTTP `Après` (pour prendre les informations envoyées par le utilisateur pour créer un employé). Chaque classe dérivée est uniquement responsable de l'installation nécessaire dans son contexte spécifique, et pour fournir les assertions nécessaires pour vérifier les résultats pour son contexte de test spécifique.



### Figure 3

La Convention d'affectation de noms et le style de test présentés ici ne sont pas requis pour le code testable, il s'agit d'une seule approche. La figure 4 montre les tests en cours d'exécution dans le plug-in Test Runner de jet cerveau pour Visual Studio 2010.



### Figure 4

Avec une classe de base pour gérer le code d'installation partagé, les tests unitaires pour chaque action du contrôleur sont petits et faciles à écrire. Les tests s'exécuteront rapidement (puisque nous effectuons des opérations en mémoire) et ne devraient pas échouer en raison d'une infrastructure ou de problèmes environnementaux non liés (car nous avons isolé l'unité testée).

```
[TestClass]
public class EmployeeControllerCreateActionPostTests
    : EmployeeControllerTestBase {
    [TestMethod]
    public void ShouldAddNewEmployeeToRepository() {
        _controller.Create(_newEmployee);
        Assert.IsTrue(_repository.Contains(_newEmployee));
    }
    [TestMethod]
    public void ShouldCommitUnitOfWork() {
        _controller.Create(_newEmployee);
        Assert.IsTrue(_unitOfWork.Committed);
    }
    // ... more tests

    Employee _newEmployee = new Employee() {
        Name = "NEW EMPLOYEE",
        HireDate = new System.DateTime(2010, 1, 1)
    };
}
```

Dans ces tests, la classe de base effectue la plupart des tâches d'installation. Souvenez-vous que le constructeur de classe de base crée le référentiel en mémoire, une unité de travail factice et une instance de la classe EmployeeController. La classe de test dérive de cette classe de base et se concentre sur les spécificités du test de la méthode Create. Dans ce cas, les spécificités s'appliquent aux étapes « arrange, Act et Assert » que vous verrez dans toutes les procédures de test unitaire :

- Créez un objet nouvel employé pour simuler des données entrantes.
- Appelez l'action Create du EmployeeController et transmettez nouvel employé.

- Vérifiez que l'action créer produit les résultats attendus (l'employé apparaît dans le référentiel).

Ce que nous avons créé nous permet de tester n'importe quelle action EmployeeController. Par exemple, lorsque nous écrivons des tests pour l'action d'index du contrôleur Employee Controller, nous pouvons hériter de la classe de base test pour établir la même configuration de base pour nos tests. Là encore, la classe de base crée le référentiel en mémoire, l'unité de travail factice et une instance de EmployeeController. Les tests de l'action d'index doivent uniquement se concentrer sur l'appel de l'action d'index et le test des qualités du modèle retourné par l'action.

```
[TestClass]
public class EmployeeControllerIndexActionTests
    : EmployeeControllerTestBase {
    [TestMethod]
    public void ShouldBuildModelWithAllEmployees() {
        var result = _controller.Index();
        var model = result.ViewData.Model
            as IEnumerable<Employee>;
        Assert.IsTrue(model.Count() == _employeeData.Count);
    }
    [TestMethod]
    public void ShouldOrderModelByHiredateAscending() {
        var result = _controller.Index();
        var model = result.ViewData.Model
            as IEnumerable<Employee>;
        Assert.IsTrue(model.SequenceEqual(
            _employeeData.OrderBy(e => e.HireDate)));
    }
    // ...
}
```

Les tests que nous créons avec les substituts en mémoire sont orientés vers le test de l'*État* du logiciel. Par exemple, lors du test de l'action de création, nous souhaitons inspecter l'état du dépôt après l'exécution de l'action de création : le référentiel conserve-t-il le nouvel employé ?

```
[TestMethod]
public void ShouldAddNewEmployeeToRepository() {
    _controller.Create(_newEmployee);
    Assert.IsTrue(_repository.Contains(_newEmployee));
}
```

Nous examinerons ultérieurement les tests basés sur l'interaction. Le test basé sur l'interaction vous demande si le code testé a appelé les méthodes appropriées sur nos objets et a passé les paramètres corrects. Pour le moment, nous allons passer à la couverture d'un autre modèle de conception : le chargement différé.

## Chargement hâtif et chargement différé

À un moment donné dans l'application Web MVC ASP.NET, nous pouvons souhaiter afficher les informations d'un employé et inclure les cartes de point associées de l'employé. Par exemple, il peut y avoir un affichage de résumé de la carte de temps qui indique le nom de l'employé et le nombre total de cartes de temps dans le système. Il existe plusieurs approches pour implémenter cette fonctionnalité.

### Projection

Une approche simple pour créer le résumé consiste à construire un modèle dédié aux informations que nous souhaitons afficher dans la vue. Dans ce scénario, le modèle peut ressembler à ce qui suit.

```

public class EmployeeSummaryViewModel {
    public string Name { get; set; }
    public int TotalTimeCards { get; set; }
}

```

Notez que le EmployeeSummaryViewModel n'est pas une entité. en d'autres termes, il ne doit pas être conservé dans la base de données. Nous allons uniquement utiliser cette classe pour mélanger des données dans la vue d'une manière fortement typée. Le modèle de vue est semblable à un objet de transfert de données (DTO), car il ne contient aucun comportement (aucune méthode), uniquement des propriétés. Les propriétés contiendront les données que vous devez déplacer. Il est facile d'instancier ce modèle de vue à l'aide de l'opérateur de projection standard de LINQ (l'opérateur SELECT).

```

public ViewResult Summary(int id) {
    var model = _unitOfWork.Employees
        .Where(e => e.Id == id)
        .Select(e => new EmployeeSummaryViewModel
        {
            Name = e.Name,
            TotalTimeCards = e.TimeCards.Count()
        })
        .Single();
    return View(model);
}

```

Il existe deux fonctionnalités notables pour le code ci-dessus. Tout d'abord, le code est facile à tester, car il est toujours facile à observer et à isoler. L'opérateur SELECT fonctionne tout aussi bien sur nos substituts en mémoire que sur l'unité de travail réelle.

```

[TestClass]
public class EmployeeControllerSummaryActionTests
    : EmployeeControllerTestBase {
    [TestMethod]
    public void ShouldBuildModelWithCorrectEmployeeSummary() {
        var id = 1;
        var result = _controller.Summary(id);
        var model = result.ViewData.Model as EmployeeSummaryViewModel;
        Assert.IsTrue(model.TotalTimeCards == 3);
    }
    // ...
}

```

La deuxième fonctionnalité notable est la manière dont le code permet à EF4 de générer une requête unique et efficace pour assembler les informations relatives aux employés et aux cartes de temps. Nous avons chargé les informations sur les employés et les informations de carte de temps dans le même objet sans utiliser d'API spéciales. Le code a simplement exprimé les informations dont il a besoin à l'aide d'opérateurs LINQ standard qui fonctionnent avec les sources de données en mémoire, ainsi que les sources de données distantes. EF4 a pu traduire les arborescences d'expressions générées par la requête LINQ et le compilateur C# en une requête T-SQL unique et efficace.

```

SELECT
    [Limit1].[Id] AS [Id],
    [Limit1].[Name] AS [Name],
    [Limit1].[C1] AS [C1]
FROM (SELECT TOP (2)
    [Project1].[Id] AS [Id],
    [Project1].[Name] AS [Name],
    [Project1].[C1] AS [C1]
    FROM (SELECT
        [Extent1].[Id] AS [Id],
        [Extent1].[Name] AS [Name],
        (SELECT COUNT(1) AS [A1]
        FROM [dbo].[TimeCards] AS [Extent2]
        WHERE [Extent1].[Id] =
            [Extent2].[EmployeeTimeCard_TimeCard_Id]) AS [C1]
        FROM [dbo].[Employees] AS [Extent1]
        WHERE [Extent1].[Id] = @p_linq_0
    ) AS [Project1]
) AS [Limit1]

```

Il y a d'autres fois que nous ne souhaitons pas travailler avec un modèle de vue ou un objet DTO, mais avec des entités réelles. Lorsque nous savons que nous avons besoin d'un employé *et* des cartes de l'employé, nous pouvons charger les données associées de manière discrète et efficace.

### Chargement hâtif explicite

Lorsque nous souhaitons charger des informations d'entité connexes, nous avons besoin d'un mécanisme pour la logique métier (ou dans ce scénario, la logique d'action du contrôleur) pour exprimer sa volonté dans le référentiel. La classe EF4 ObjectQuery<T> définit une méthode `Include` pour spécifier les objets connexes à récupérer au cours d'une requête. N'oubliez pas que EF4 ObjectContext expose des entités via la classe concrète ObjectSet<T> qui hérite de ObjectQuery<T>. Si nous utilisions ObjectSet<T> des références dans notre action de contrôleur, nous pourrions écrire le code suivant pour spécifier un chargement hâtif d'informations de carte de temps pour chaque employé.

```

_employees.Include("TimeCards")
    .Where(e => e.HireDate.Year > 2009);

```

Toutefois, étant donné que nous essayons de garder notre code testable, nous n'exposez pas ObjectSet<T> en dehors de la classe de l'unité de travail réelle. Au lieu de cela, nous nous appuyons sur l'interface IObjectSet<T> qui est plus facile à falsifier, mais IObjectSet<T> ne définit pas de méthode `Include`. La beauté de LINQ est que nous pouvons créer notre propre opérateur `include`.

```

public static class QueryableExtensions {
    public static IQueryables<T> Include<T>
        (this IQueryables<T> sequence, string path) {
        var objectQuery = sequence as ObjectQuery<T>;
        if(objectQuery != null)
        {
            return objectQuery.Include(path);
        }
        return sequence;
    }
}

```

Notez que cet opérateur `include` est défini en tant que méthode d'extension pour `IQueryables<T>` au lieu de `IObjectSet<T>`. Cela nous donne la possibilité d'utiliser la méthode avec un plus grand nombre de types possibles, notamment `IQueryables<T>`, `IObjectSet<T>`, `ObjectQuery<T>` et `ObjectSet<T>`. Dans le cas où la séquence sous-jacente n'est pas une authentique EF4 `ObjectQuery<T>`, il n'y a pas de préjudice effectué et l'opérateur `include` est

une absence d'opération. Si la séquence sous-jacente est un ObjectQuery<t> (ou dérivé de ObjectQuery<t>), EF4 verra notre exigence concernant des données supplémentaires et formulera la requête SQL appropriée.

Avec ce nouvel opérateur en place, nous pouvons demander explicitement un chargement hâtif d'informations de carte de temps dans le référentiel.

```
public ViewResult Index() {
    var model = _unitOfWork.Employees
        .Include("TimeCards")
        .OrderBy(e => e.HireDate);
    return View(model);
}
```

Lorsqu'il est exécuté sur un ObjectContext réel, le code génère la requête unique suivante. La requête rassemble suffisamment d'informations de la base de données en un seul voyage pour matérialiser les objets des employés et remplir entièrement leur propriété de la table de temps.

```
SELECT
[Project1].[Id] AS [Id],
[Project1].[Name] AS [Name],
[Project1].[HireDate] AS [HireDate],
[Project1].[C1] AS [C1],
[Project1].[Id1] AS [Id1],
[Project1].[Hours] AS [Hours],
[Project1].[EffectiveDate] AS [EffectiveDate],
[Project1].[EmployeeTimeCard_TimeCard_Id] AS [EmployeeTimeCard_TimeCard_Id]
FROM ( SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[HireDate] AS [HireDate],
    [Extent2].[Id] AS [Id1],
    [Extent2].[Hours] AS [Hours],
    [Extent2].[EffectiveDate] AS [EffectiveDate],
    [Extent2].[EmployeeTimeCard_TimeCard_Id] AS
        [EmployeeTimeCard_TimeCard_Id],
    CASE WHEN ([Extent2].[Id] IS NULL) THEN CAST(NULL AS int)
    ELSE 1 END AS [C1]
    FROM [dbo].[Employees] AS [Extent1]
    LEFT OUTER JOIN [dbo].[TimeCards] AS [Extent2] ON [Extent1].[Id] = [Extent2].
[EmployeeTimeCard_TimeCard_Id]
) AS [Project1]
ORDER BY [Project1].[HireDate] ASC,
[Project1].[Id] ASC, [Project1].[C1] ASC
```

La bonne nouvelle, c'est que le code à l'intérieur de la méthode d'action reste entièrement testable. Nous n'avons pas besoin de fournir des fonctionnalités supplémentaires pour notre substitut pour prendre en charge l'opérateur include. La mauvaise nouvelle, c'est que nous devions utiliser l'opérateur include à l'intérieur du code, nous souhaitons conserver l'ignore de la persistance. Il s'agit d'un excellent exemple du type de compromis que vous devrez évaluer lors de la création de code testable. Dans certains cas, vous devez laisser des problèmes de persistance en dehors de l'abstraction du référentiel pour atteindre les objectifs de performances.

L'alternative au chargement hâtif est le chargement différé. Le chargement différé signifie que nous n'avons *pas* besoin de notre code d'entreprise pour annoncer explicitement la nécessité de données associées. Au lieu de cela, nous utilisons nos entités dans l'application et si des données supplémentaires sont nécessaires Entity Framework chargera les données à la demande.

### **Chargement différé**

Il est facile d'imaginer un scénario dans lequel nous ne savons pas quelles données une partie de la logique métier aura besoin. Nous savons peut-être que la logique a besoin d'un objet Employee, mais nous pouvons créer des branches dans différents chemins d'exécution où certains de ces chemins nécessitent des informations de carte de

l'employé et d'autres non. Les scénarios de ce type sont parfaits pour le chargement différé implicite, car les données s'affichent par magie en fonction des besoins.

Le chargement différé, également connu sous le nom de chargement différé, place certaines exigences sur nos objets d'entité. Les POCO avec l'ignorance de la persistance réelle ne représenteront aucune exigence de la couche de persistance, mais l'ignorance de la persistance est pratiquement impossible à atteindre. Au lieu de cela, nous mesurons l'ignorance de la persistance en degrés relatifs. Cela serait malheureux si nous avions besoin d'hériter d'une classe de base orientée persistance ou d'utiliser une collection spécialisée pour atteindre un chargement différé dans les POCO. Heureusement, EF4 a une solution moins intrusive.

### Pratiquement indétectables

Lors de l'utilisation d'objets POCO, EF4 peut générer dynamiquement des proxies d'exécution pour les entités. Ces proxies encapsulent de façon invisible les POCO matérialisés et fournissent des services supplémentaires en interceptant chaque propriété obtenir et définir l'opération pour effectuer un travail supplémentaire. L'un de ces services est la fonctionnalité de chargement différé que nous recherchons. Un autre service est un mécanisme de suivi des modifications efficace qui peut enregistrer lorsque le programme modifie les valeurs de propriété d'une entité. La liste des modifications est utilisée par `ObjectContext` pendant la méthode `SaveChanges` pour rendre persistantes toutes les entités modifiées à l'aide de commandes `UPDATE`.

Toutefois, pour que ces proxies fonctionnent, ils ont besoin d'un moyen de se raccorder à des opérations d'extraction et de définition de propriétés sur une entité, et les proxys atteignent cet objectif en remplaçant les membres virtuels. Par conséquent, si nous souhaitons avoir un chargement différé implicite et un suivi efficace des modifications, nous devons revenir à nos définitions de classe POCO et marquer les propriétés comme étant virtuelles.

```
public class Employee {
    public virtual int Id { get; set; }
    public virtual string Name { get; set; }
    public virtual DateTime HireDate { get; set; }
    public virtual ICollection<TimeCard> TimeCards { get; set; }
}
```

Nous pouvons toujours indiquer que l'entité `Employee` est principalement ignorant la persistance. La seule exigence consiste à utiliser des membres virtuels et cela n'a aucun impact sur la testabilité du code. Nous n'avons pas besoin de dériver d'une classe de base spéciale, ni même d'utiliser une collection spéciale dédiée au chargement différé. Comme le montre le code, toute classe qui implémente `ICollection<T>` est disponible pour contenir les entités associées.

Il y a également une modification mineure que nous devons faire au sein de notre unité de travail. Le chargement différé est désactivé par défaut quand vous travaillez directement avec un objet `ObjectContext`. Il existe une propriété que nous pouvons définir sur la propriété `ContextOptions` pour activer le chargement différé, et nous pouvons définir cette propriété à l'intérieur de notre véritable unité de travail si vous souhaitez activer le chargement différé partout.

```
public class SqlUnitOfWork : IUnitOfWork {
    public SqlUnitOfWork() {
        // ...
        _context = new ObjectContext(connectionString);
        _context.ContextOptions.LazyLoadingEnabled = true;
    }
    // ...
}
```

Si le chargement différé implicite est activé, le code de l'application peut utiliser un employé et les cartes de temps associées de l'employé, tout en restant tranquillement ne connaissant pas le travail nécessaire à EF pour charger les

données supplémentaires.

```
var employee = _unitOfWork.Employees
    .Single(e => e.Id == id);
foreach (var card in employee.TimeCards) {
    // ...
}
```

Le chargement différé rend le code de l'application plus facile à écrire et, avec la magie du proxy, le code reste entièrement testable. Les substituts en mémoire de l'unité de travail peuvent simplement précharger des entités factices avec des données associées lorsque cela est nécessaire pendant un test.

À ce stade, nous nous pencherons sur la création de référentiels à l'aide de `IObjectSet<T>` et nous examinerons des abstractions pour masquer tous les signes de l'infrastructure de persistance.

## Référentiels personnalisés

Lorsque nous avons présenté le modèle de conception d'unité de travail dans cet article, nous avons fourni un exemple de code pour ce à quoi peut ressembler l'unité de travail. Nous allons représenter cette idée originale à l'aide du scénario Employee et Employee Time Card que nous travaillons.

```
public interface IUnitOfWork {
    IRepository<Employee> Employees { get; }
    IRepository<TimeCard> TimeCards { get; }
    void Commit();
}
```

La principale différence entre cette unité de travail et l'unité de travail que nous avons créée dans la dernière section est la manière dont cette unité de travail n'utilise pas d'abstractions de l'infrastructure EF4 (il n'y a pas de `IObjectSet<T>`). `IObjectSet<T>` fonctionne bien comme une interface de référentiel, mais l'API qu'il expose peut ne pas être parfaitement adaptée aux besoins de l'application. Dans cette approche à venir, nous allons représenter les référentiels à l'aide d'une personnalisée `IRepository<T>` abstraction.

De nombreux développeurs qui suivent la conception pilotée par test, la conception pilotée par le comportement et les méthodologies pilotées par domaine préfèrent l'approche `IRepository<T>` pour plusieurs raisons. Tout d'abord, l'interface `IRepository<T>` représente une couche « anti-dommages ». Comme décrit par Eric Evans dans son livre de conception piloté par domaine, une couche de lutte contre la corruption permet d'éloigner votre code de domaine des API d'infrastructure, comme une API de persistance. Deuxièmement, les développeurs peuvent créer des méthodes dans le référentiel qui répondent aux besoins exacts d'une application (telle qu'elle a été détectée pendant l'écriture de tests). Par exemple, il se peut que nous ayons souvent besoin de localiser une seule entité à l'aide d'une valeur d'ID. nous pouvons donc ajouter une méthode `FindById` à l'interface du référentiel. Notre définition de `IRepository<T>` se présente comme suit.

```
public interface IRepository<T>
    where T : class, IEntity {
    IQueryable<T> FindAll();
    IQueryable<T> FindWhere(Expression<Func<T, bool>> predicate);
    T FindById(int id);
    void Add(T newEntity);
    void Remove(T entity);
}
```

Notez que nous allons revenir à l'utilisation d'une interface `IQueryable<T>` pour exposer des collections d'entités. `IQueryable<T>` permet aux arborescences d'expression LINQ de circuler dans le fournisseur EF4 et de fournir au fournisseur une vue holistique de la requête. Une seconde option consiste à retourner `IEnumerable<T>`, ce qui

signifie que le fournisseur LINQ EF4 verra uniquement les expressions générées dans le référentiel. Les regroupements, ordonnancements et projection effectués en dehors du référentiel ne sont pas composés de la commande SQL envoyée à la base de données, ce qui peut nuire aux performances. En revanche, un référentiel renvoyant uniquement `IEnumerable<T>` résultats ne vous sera jamais surpris par une nouvelle commande SQL. Les deux approches fonctionnent et les deux approches restent testables.

Il est facile de fournir une implémentation unique de l'interface `IRepository<T>` à l'aide de génériques et de l'API `ObjectContext` EF4.

```
public class SqlRepository<T> : IRepository<T>
{
    where T : class, IEntity {
        public SqlRepository(ObjectContext context) {
            _objectSet = context.CreateObjectSet<T>();
        }
        public IQueryable<T> FindAll() {
            return _objectSet;
        }
        public IQueryable<T> FindWhere(
            Expression<Func<T, bool>> predicate) {
            return _objectSet.Where(predicate);
        }
        public T FindById(int id) {
            return _objectSet.Single(o => o.Id == id);
        }
        public void Add(T newEntity) {
            _objectSet.AddObject(newEntity);
        }
        public void Remove(T entity) {
            _objectSet.DeleteObject(entity);
        }
        protected ObjectSet<T> _objectSet;
    }
}
```

L'approche `IRepository<T>` nous donne un contrôle supplémentaire sur nos requêtes, car un client doit appeler une méthode pour accéder à une entité. À l'intérieur de la méthode, nous pourrions fournir des contrôles supplémentaires et des opérateurs LINQ pour appliquer des contraintes d'application. Notez que l'interface a deux contraintes sur le paramètre de type générique. La première contrainte est la classe `Entity` requise par `ObjectSet<T>`, et la deuxième contrainte force nos entités à implémenter `IEntity` – une abstraction créée pour l'application. L'interface `IEntity` force les entités à avoir une propriété `ID` lisible, et nous pouvons ensuite utiliser cette propriété dans la méthode `FindById`. `IEntity` est défini avec le code suivant.

```
public interface IEntity {
    int Id { get; }
}
```

`IEntity` peut être considéré comme une faible violation de l'ignorance de la persistance, car nos entités sont requises pour implémenter cette interface. Rappelez-vous que l'ignorance de la persistance concerne les compromis, et que de nombreuses fonctionnalités `FindById` compenseront la contrainte imposée par l'interface. L'interface n'a aucun impact sur la testabilité.

L'instanciation d'un <code>en IRepository<T></code> nécessite un `ObjectContext` EF4, de sorte qu'une implémentation de l'unité de travail concrète doit gérer l'instanciation.

```

public class SqlUnitOfWork : IUnitOfWork {
    public SqlUnitOfWork() {
        var connectionString =
            ConfigurationManager
                .ConnectionStrings[ConnectionStringName]
                .ConnectionString;

        _context = new ObjectContext(connectionString);
        _context.ContextOptions.LazyLoadingEnabled = true;
    }

    public IRepository<Employee> Employees {
        get {
            if (_employees == null) {
                _employees = new SqlRepository<Employee>(_context);
            }
            return _employees;
        }
    }

    public IRepository<TimeCard> TimeCards {
        get {
            if (_timeCards == null) {
                _timeCards = new SqlRepository<TimeCard>(_context);
            }
            return _timeCards;
        }
    }

    public void Commit() {
        _context.SaveChanges();
    }

    SqlRepository<Employee> _employees = null;
    SqlRepository<TimeCard> _timeCards = null;
    readonly ObjectContext _context;
    const string ConnectionStringName = "EmployeeDataModelContainer";
}

```

## Utilisation du référentiel personnalisé

L'utilisation de notre référentiel personnalisé n'est pas très différente de celle d'un référentiel basé sur `IObjectSet<T>`. Au lieu d'appliquer des opérateurs LINQ directement à une propriété, nous devons d'abord appeler une des méthodes du référentiel pour récupérer une référence `IQueryable<T>`.

```

public ViewResult Index() {
    var model = _repository.FindAll()
        .Include("TimeCards")
        .OrderBy(e => e.HireDate);
    return View(model);
}

```

Notez que l'opérateur `include` personnalisé que nous avons implémenté précédemment fonctionnera sans modification. La méthode `FindById` du référentiel supprime la logique dupliquée des actions tentant de récupérer une entité unique.

```

public ViewResult Details(int id) {
    var model = _repository.FindById(id);
    return View(model);
}

```

Il n'existe aucune différence significative dans la testabilité des deux approches que nous avons examinées. Nous

pourrions fournir des implémentations factices de IRepository<T> en générant des classes concrètes sauvegardées par HashSet<Employee>, tout comme nous l'avons fait dans la dernière section. Toutefois, certains développeurs préfèrent utiliser des objets factices et des infrastructures d'objets factices au lieu de générer des substituts. Nous allons examiner l'utilisation de simulacres pour tester notre implémentation et discuter des différences entre les simulacres et les simulations dans la section suivante.

## Test avec des simulacres

Il existe différentes approches pour créer ce que Martin Fowler appelle un « test double ». Un double de test (comme un film stunt double) est un objet que vous créez pour « mettre en attente » pour les objets de production réels pendant les tests. Les dépôts en mémoire que nous avons créés sont des doubles de test pour les dépôts qui communiquent avec SQL Server. Nous avons vu comment utiliser ces doubles de test lors des tests unitaires pour isoler le code et faire en sorte que les tests s'exécutent rapidement.

Les doubles de test que nous avons créés ont des implémentations de travail réelles. En arrière-plan, chacun d'entre eux stocke une collection concrète d'objets, et ceux-ci ajoutent et suppriment des objets de cette collection au fur et à mesure que nous manipulons le référentiel pendant un test. Certains développeurs aiment créer leurs doubles de test de cette façon, avec un code réel et des implémentations de travail. Ces doubles de test sont ce que nous appelons les *substituts*. Ils ont des implémentations opérationnelles, mais ils ne sont pas assez réels pour une utilisation en production. Le référentiel factice n'écrit pas réellement dans la base de données. Le serveur SMTP factice n'envoie pas en fait un message électronique sur le réseau.

## Simulacres et substituts

Il existe un autre type de test double connu sous le nom de *fictif*. Alors que les substituts ont des implémentations opérationnelles, les simulacres ne sont pas implémentés. Avec l'aide d'une infrastructure d'objets factices, nous construisons ces objets factices au moment de l'exécution et les utilisons en tant que double de test. Dans cette section, nous allons utiliser l'infrastructure de simulation open source MOQ. Voici un exemple simple d'utilisation de MOQ pour créer dynamiquement un double de test pour un dépôt d'employés.

```
Mock<IRepository<Employee>> mock =
    new Mock<IRepository<Employee>>();
IRepository<Employee> repository = mock.Object;
repository.Add(new Employee());
var employee = repository.FindById(1);
```

Nous demandons à MOQ la mise en œuvre d'un IRepository<employé> et une implémentation dynamique. Nous pouvons accéder à l'objet qui implémente IRepository<Employee> en accédant à la propriété Object de l'objet factice<T>. Il s'agit de cet objet interne que nous pouvons transmettre à nos contrôleurs, et ils ne savent pas s'il s'agit d'un double de test ou d'un référentiel réel. Nous pouvons appeler des méthodes sur l'objet de la même façon que nous appellerons des méthodes sur un objet avec une implémentation réelle.

Vous devez vous demander ce que fera le dépôt fictif quand nous invoquons la méthode Add. Étant donné qu'il n'y a pas d'implémentation derrière l'objet factice, Add n'a aucun effet. Il n'y a pas de collection concrète en arrière-plan, comme nous l'avons fait avec les substituts que nous avons écrits, l'employé est donc ignoré. Qu'en est-il de la valeur de retour de FindById ? Dans ce cas, l'objet factice fait la seule chose qu'il peut faire, qui retourne une valeur par défaut. Étant donné que nous retournons un type référence (un employé), la valeur de retour est une valeur null.

Les simulacres peuvent ne pas avoir de bruit ; Toutefois, nous n'avons pas parlé de deux autres fonctionnalités factives. Tout d'abord, l'infrastructure MOQ enregistre tous les appels effectués sur l'objet factice. Plus tard dans le code, nous pouvons demander à MOQ si quelqu'un a appelé la méthode Add ou si quelqu'un a appelé la méthode FindById. Nous verrons plus tard comment nous pouvons utiliser cette fonctionnalité d'enregistrement « boîte noire » dans les tests.

La deuxième fonctionnalité intéressante est la façon dont nous pouvons utiliser MOQ pour programmer un objet fictif avec des *attentes*. Une attente indique à l'objet factice comment répondre à une interaction donnée. Par

exemple, nous pouvons programmer une attente dans notre simulacre et lui demander de retourner un objet employé lorsqu'un utilisateur appelle `FindById`. L'infrastructure MOQ utilise une API d'installation et des expressions lambda pour programmer ces attentes.

```
[TestMethod]
public void MockSample() {
    Mock< IRepository<Employee>> mock =
        new Mock< IRepository<Employee>>();
    mock.Setup(m => m.FindById(5))
        .Returns(new Employee { Id = 5 });
    IRepository<Employee> repository = mock.Object;
    var employee = repository.FindById(5);
    Assert.IsTrue(employee.Id == 5);
}
```

Dans cet exemple, nous demandons à MOQ de créer dynamiquement un référentiel, puis de programmer le dépôt dans une attente. L'attente indique à l'objet factice de retourner un nouvel objet `Employee` avec une valeur d'ID de 5 lorsqu'un utilisateur appelle la méthode `FindById` en passant une valeur de 5. Ce test réussit et nous n'avons pas besoin de créer une implémentation complète pour falsifier `IRepository<T>`.

Nous allons revoir les tests que nous avons écrits précédemment et les réutiliser pour utiliser des simulacres au lieu de simulations. Comme précédemment, nous utilisons une classe de base pour configurer les éléments d'infrastructure courants dont nous avons besoin pour tous les tests du contrôleur.

```
public class EmployeeControllerTestBase {
    public EmployeeControllerTestBase() {
        _employeeData = EmployeeObjectMother.CreateEmployees()
            .AsQueryable();
        _repository = new Mock< IRepository<Employee>>();
        _unitOfWork = new Mock< IUnitOfWork >();
        _unitOfWork.Setup(u => u.Employees)
            .Returns(_repository.Object);
        _controller = new EmployeeController(_unitOfWork.Object);
    }

    protected IQueryable<Employee> _employeeData;
    protected Mock< IUnitOfWork > _unitOfWork;
    protected EmployeeController _controller;
    protected Mock< IRepository<Employee>> _repository;
}
```

Le code d'installation reste quasiment le même. Au lieu d'utiliser des substituts, nous allons utiliser MOQ pour construire des objets factices. La classe de base fait en sorte que l'unité factice de travail retourne un référentiel fictif lorsque le code appelle la propriété `Employees`. Le reste de l'installation factice aura lieu à l'intérieur des contextes de test dédiés à chaque scénario spécifique. Par exemple, le contexte de test de l'action `d'index` configurera le référentiel fictif pour retourner une liste d'employés quand l'action appelle la méthode `FindAll` du référentiel fictif.

```

[TestClass]
public class EmployeeControllerIndexActionTests
    : EmployeeControllerTestBase {
    public EmployeeControllerIndexActionTests() {
        _repository.Setup(r => r.FindAll())
            .Returns(_employeeData);
    }
    // ... tests
    [TestMethod]
    public void ShouldBuildModelWithAllEmployees() {
        var result = _controller.Index();
        var model = result.ViewData.Model
            as IEnumerable<Employee>;
        Assert.IsTrue(model.Count() == _employeeData.Count());
    }
    // ... and more tests
}

```

À l'exception des attentes, nos tests sont similaires aux tests que nous avions auparavant. Toutefois, avec la capacité d'enregistrement d'un Framework fictif, nous pouvons aborder les tests à partir d'un angle différent. Nous allons examiner cette nouvelle perspective dans la section suivante.

### État et tests d'interaction

Vous pouvez utiliser différentes techniques pour tester des logiciels avec des objets fictifs. Une approche consiste à utiliser des tests basés sur l'État, ce que nous avons fait dans ce document jusqu'à présent. Le test basé sur les États fait des assertions sur l'état du logiciel. Dans le dernier test, nous avons appelé une méthode d'action sur le contrôleur et effectué une assertion sur le modèle à générer. Voici d'autres exemples d'état de test :

- Vérifiez que le référentiel contient le nouvel objet d'employé après l'exécution de Create.
- Vérifiez que le modèle contient une liste de tous les employés après l'exécution de l'index.
- Vérifiez que le dépôt ne contient pas d'employé donné après l'exécution de la suppression.

Une autre approche que vous verrez avec les objets factices consiste à vérifier les *interactions*. Bien que le test basé sur l'État fasse des assertions sur l'état des objets, le test basé sur l'interaction fait des assertions sur la manière dont les objets interagissent. Exemple :

- Vérifiez que le contrôleur appelle la méthode Add du référentiel lorsque Create s'exécute.
- Vérifiez que le contrôleur appelle la méthode FindAll du référentiel lorsque l'index s'exécute.
- Vérifiez que le contrôleur appelle la méthode Commit de l'unité de travail pour enregistrer les modifications lorsque la modification est exécutée.

Le test d'interaction requiert souvent moins de données de test, car il n'est pas possible de les percer à l'intérieur des collections et de vérifier les nombres. Par exemple, si nous savons que l'action Details appelle la méthode FindById d'un référentiel avec la valeur correcte, l'action se comporte probablement correctement. Nous pouvons vérifier ce comportement sans configurer les données de test à retourner à partir de FindById.

```

[TestClass]
public class EmployeeControllerDetailsActionTests
    : EmployeeControllerTestBase {
    // ...
    [TestMethod]
    public void ShouldInvokeRepositoryToFindEmployee() {
        var result = _controller.Details(_detailsId);
        _repository.Verify(r => r.FindById(_detailsId));
    }
    int _detailsId = 1;
}

```

La seule configuration requise dans le contexte de test ci-dessus est celle fournie par la classe de base. Lorsque nous invoquons l'action du contrôleur, MOQ enregistre les interactions avec le référentiel fictif. À l'aide de l'API Verify de MOQ, nous pouvons demander à MOQ si le contrôleur a appelé FindById avec la valeur d'ID appropriée. Si le contrôleur n'a pas appelé la méthode ou a appelé la méthode avec une valeur de paramètre inattendue, la méthode Verify lèvera une exception et le test échouera.

Voici un autre exemple pour vérifier que l'action Create appelle commit sur l'unité de travail actuelle.

```
[TestMethod]
public void ShouldCommitUnitOfWork() {
    _controller.Create(_newEmployee);
    _unitOfWork.Verify(u => u.Commit());
}
```

L'un des risques avec les tests d'interaction est la tendance à définir des interactions. La capacité de l'objet factice à enregistrer et à vérifier chaque interaction avec l'objet factice ne signifie pas que le test doit essayer de vérifier chaque interaction. Certaines interactions sont des détails d'implémentation et vous devez uniquement vérifier les interactions *requises* pour répondre au test actuel.

Le choix entre les simulacres ou les substituts dépend en grande partie du système que vous testez et de vos préférences personnelles (ou de votre équipe). Les objets factices peuvent réduire considérablement la quantité de code dont vous avez besoin pour implémenter les doubles de test, mais il n'est pas tout à fait facile de programmer les attentes en matière de programmation et de vérification des interactions.

## Conclusions

Dans ce document, nous avons présenté plusieurs approches pour créer du code testable tout en utilisant le Entity Framework ADO.NET pour la persistance des données. Nous pouvons tirer parti des abstractions intégrées telles que IObjectSet<T> ou créer vos propres abstractions comme IRepository<T>. Dans les deux cas, la prise en charge de POCO dans le ADO.NET Entity Framework 4,0 permet aux consommateurs de ces abstractions de rester persistants et d'être facilement testables. Des fonctionnalités EF4 supplémentaires telles que le chargement différé implicite permettent au code de service d'application et d'entreprise de fonctionner sans se soucier des détails d'une banque de données relationnelle. Enfin, les abstractions que nous créons sont faciles à imiter ou factices dans les tests unitaires, et nous pouvons utiliser ces doubles de test pour exécuter des tests rapides, très isolés et fiables.

### Ressources supplémentaires

- Robert C. Martin, « [le principe de responsabilité unique](#)»
- Martin Fowler, [catalogue des modèles](#) de l' *architecture des applications d'entreprise*
- Griffin Caprio, " [injection de dépendances](#)"
- Blog sur la programmabilité des données, « [procédure pas à pas : développement piloté par les tests avec le Entity Framework 4,0](#)».
- Blog sur la programmabilité des données, « [utilisation du référentiel et des modèles d'unité de travail avec Entity Framework 4,0](#)»
- Aaron Jensen, « [Présentation des spécifications d'ordinateur](#)»
- Eric Lee, « [BDD avec MSTest](#)»
- Eric Evans, « [conception pilotée par domaine](#)»
- Martin Fowler, « les [simulacres ne sont pas des stubs](#)»
- Martin Fowler, « [test double](#)»
- [Moq](#)

### Biographie

Scott Allen est membre du personnel technique de Pluralsight et du fondateur de OdeToCode.com. Dans 15 ans de développement logiciel commercial, Scott a travaillé sur des solutions pour tous les éléments, des appareils

embarqués 8 bits aux applications Web ASP.NET hautement évolutives. Vous pouvez contacter Scott sur son blog sur OdeToCode ou sur Twitter à <https://twitter.com/OdeToCode>.

# Création d'un modèle

11/10/2019 • 5 minutes to read

Un modèle EF stocke la façon dont sont mappées les classes d'application et les propriétés aux colonnes et aux tables de base de données. Il existe deux façons principales de créer un modèle EF :

- **Utilisation de Code First** : Le développeur écrit du code pour spécifier le modèle. EF génère les modèles et les mappages au moment de l'exécution en fonction des classes d'entité et de la configuration de modèle supplémentaire fournie par le développeur.
- **Utilisation du concepteur EF** : Le développeur dessine des zones et des lignes pour spécifier le modèle à l'aide d'EF Designer. Le modèle qui en résulte est stocké au format XML dans un fichier avec l'extension EDMX. De manière générale, les objets de domaine de l'application sont générés automatiquement à partir du modèle conceptuel.

## Flux de travail EF

Ces deux approches peuvent servir à cibler une base de données existante ou créer une base de données, ce qui génère 4 flux de travail différents. Découvrez celui qui vous convient le mieux :

|  | JE VEUX SIMPLEMENT ÉCRIRE DU CODE...  | JE VEUX UTILISER UN CONCEPTEUR...   |
|--|---|---|
| <b>Je crée une base de données</b>                           | Utilisez <b>Code First</b> pour définir votre modèle dans le code et générer une base de données.       | Utilisez <b>Model First</b> pour définir votre modèle à l'aide de zones et de lignes, puis générer une base de données. |
| <b>J'ai besoin d'accéder à une base de données existante</b> | Utilisez <b>Code First</b> pour créer un modèle basé sur le code mappé à une base de données existante. | Utilisez <b>Database First</b> pour créer un modèle de zones et de lignes mappé à une base de données existante.        |

### Regardez la vidéo : Quel workflow EF utiliser ?

Cette courte vidéo explique les différences entre les flux de travail et vous guide dans votre choix.

Présentée par : Rowan Miller



[WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

Si, après avoir regardé la vidéo, vous n'arrivez toujours pas à vous décider entre EF Designer ou Code First, apprenez à utiliser les deux !

## Rouages du système

Que vous utilisez Code First ou EF Designer, un modèle EF a toujours plusieurs composants :

- Les objets de domaine de l'application ou les types d'entité eux-mêmes. Il s'agit de ce qu'on appelle généralement la couche objet
- Un modèle conceptuel comprenant des types d'entité et des relations propres à un domaine, décrits à l'aide d'**Entity Data Model**. Cette couche est souvent désignée par la lettre « C » pour *conceptuelle*.
- Un modèle de stockage représentant des tables, des colonnes et des relations comme défini dans la base de

données. Cette couche est souvent désignée par la lettre « S » pour *stockage*.

- Un mappage entre le modèle conceptuel et le schéma de base de données. Ce mappage est souvent appelé mappage « C-S ».

Le moteur de mappage d'Entity Framework s'appuie sur le mappage « C-S » pour transformer les opérations sur les entités (par ex., créer, lire, mettre à jour et supprimer) en opérations équivalentes sur les tables dans la base de données.

Le mappage entre le modèle conceptuel et les objets de l'application est souvent appelé mappage « O-C ». Par rapport au mappage « C-S », le mappage « O-C » est implicite et établit des relations un à un : les entités, propriétés et relations définies dans le modèle conceptuel doivent correspondre aux formes et types des objets .NET. À partir d'EF4 et version ultérieure, la couche objet peut contenir des objets simples avec des propriétés sans dépendance sur EF. Ces objets simples sont généralement désignés comme des objets CLR traditionnels (OCT) et le mappage des types et des propriétés est effectué en fonction des conventions de correspondance de nom. Auparavant, dans EF 3.5, des restrictions spécifiques s'appliquaient à la couche objet, par exemple, les entités devaient dériver de la classe EntityObject et porter des attributs EF pour implémenter le mappage « O-C ».

# Code First à une nouvelle base de données

23/11/2019 • 20 minutes to read

Cette vidéo et la procédure pas à pas suivante fournissent une introduction au développement Code First ciblant une base de données. Ce scénario inclut une base de données cible qui n'existe pas et que Code First va créer, ou une base de données vide où Code First ajoutera des tables. Code First va tout d'abord vous permettre de définir votre modèle à l'aide de classes C# ou VB.Net. Une configuration supplémentaire peut éventuellement être effectuée à l'aide des attributs dans vos classes et propriétés ou à l'aide d'une API Fluent.

## Regarder la vidéo

Cette vidéo fournit une introduction au développement Code First ciblant une base de données. Ce scénario inclut une base de données cible qui n'existe pas et que Code First va créer, ou une base de données vide où Code First ajoutera des tables. Code First va tout d'abord vous permettre de définir votre modèle à l'aide de classes C# ou VB.Net. Une configuration supplémentaire peut éventuellement être effectuée à l'aide des attributs dans vos classes et propriétés ou à l'aide d'une API Fluent.

**Présentée par:** [Rowan Miller](#)

**Vidéo:** [wmv](#) | [MP4](#) | [WMV \(zip\)](#)

## Conditions préalables

Vous devez avoir au moins Visual Studio 2010 ou Visual Studio 2012 installé pour effectuer cette procédure pas à pas.

Si vous utilisez Visual Studio 2010, [NuGet](#) doit également être installé.

## 1. créer l'application

Pour simplifier les choses, nous allons créer une application console de base qui utilise Code First pour effectuer l'accès aux données.

- Ouvrez Visual Studio
- **Fichier> nouveau>...**
- Sélectionnez **Windows** dans le menu de gauche et dans l'**application console** .
- Entrez **CodeFirstNewDatabaseSample** comme nom
- Sélectionnez **OK**.

## 2. créer le modèle

Nous allons définir un modèle très simple à l'aide de classes. Nous allons simplement les définir dans le fichier Program.cs, mais dans une application réelle, vous fractionnez vos classes en fichiers distincts et éventuellement dans un projet distinct.

Sous la définition de classe de programme dans Program.cs, ajoutez les deux classes suivantes.

```

public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public virtual Blog Blog { get; set; }
}

```

Vous remarquerez que les deux propriétés de navigation (blog. publications et post. blog) sont virtuelles. Cela active la fonctionnalité de chargement différé de Entity Framework. Le chargement différé signifie que le contenu de ces propriétés est chargé automatiquement à partir de la base de données lorsque vous tentez d'y accéder.

### 3. créer un contexte

À présent, il est temps de définir un contexte dérivé, qui représente une session avec la base de données, ce qui nous permet d'interroger et d'enregistrer des données. Nous définissons un contexte qui dérive de System. Data. Entity. DbContext et expose un DbSet de type <tente de> typé pour chaque classe dans notre modèle.

Nous commençons à utiliser les types de la Entity Framework, donc nous devons ajouter le package NuGet EntityFramework.

- **Projet-> gérer les packages NuGet...** Remarque : Si vous n'avez pas la **gestion des packages NuGet...** option vous devez installer la [dernière version de NuGet](#)
- Sélectionner l'onglet **en ligne**
- Sélectionner le package **EntityFramework**
- Cliquez sur **installer**

Ajoutez une instruction using pour System. Data. Entity en haut de Program.cs.

```
using System.Data.Entity;
```

Sous la classe de publication dans Program.cs, ajoutez le contexte dérivé suivant.

```

public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}

```

Voici une liste complète de ce que Program.cs doit maintenant contenir.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.Entity;

namespace CodeFirstNewDatabaseSample
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }

        public virtual List<Post> Posts { get; set; }
    }

    public class Post
    {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }

        public int BlogId { get; set; }
        public virtual Blog Blog { get; set; }
    }

    public class BloggingContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
        public DbSet<Post> Posts { get; set; }
    }
}

```

C'est tout le code dont nous avons besoin pour commencer à stocker et à récupérer les données. Évidemment, il y a beaucoup de choses en coulisses et nous examinerons cela dans un moment, mais commençons par le voir en action.

## 4. lecture & écriture de données

Implémentez la méthode main dans Program.cs comme indiqué ci-dessous. Ce code crée une nouvelle instance de notre contexte, puis l'utilise pour insérer un nouveau blog. Il utilise ensuite une requête LINQ pour récupérer tous les blogs de la base de données classée par ordre alphabétique par titre.

```

class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}

```

Vous pouvez maintenant exécuter l'application et la tester.

```

Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
ADO.NET Blog
Press any key to exit...

```

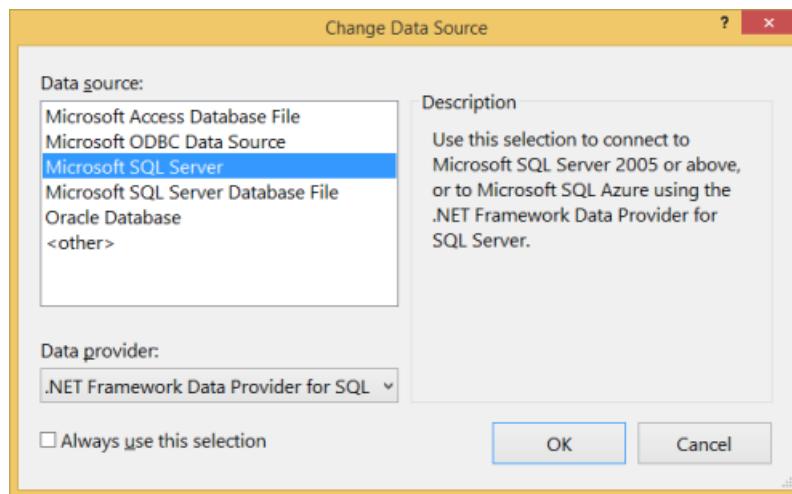
## Où sont mes données ?

Par la Convention DbContext a créé une base de données pour vous.

- Si une instance locale de SQL Express est disponible (installée par défaut avec Visual Studio 2010), Code First a créé la base de données sur cette instance.
- Si SQL Express n'est pas disponible, Code First essaiera et utilisera la [base de données locale \(installée par défaut avec Visual Studio 2012\)](#)
- La base de données est nommée après le nom qualifié complet du contexte dérivé, dans notre cas **CodeFirstNewDatabaseSample. BloggingContext**

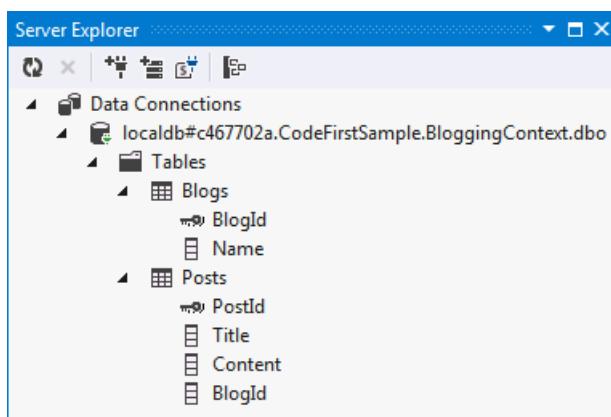
Il s'agit uniquement des conventions par défaut et il existe différentes façons de modifier la base de données que Code First utilise, plus d'informations sont disponibles dans la rubrique **How DbContext Découvre le modèle et la connexion à la base de données**. Vous pouvez vous connecter à cette base de données à l'aide de Explorateur de serveurs dans Visual Studio

- **Vue-> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **connexions de données**, puis sélectionnez **Ajouter une connexion...**
- Si vous n'êtes pas connecté à une base de données à partir de Explorateur de serveurs avant de devoir sélectionner Microsoft SQL Server comme source de données



- Connectez-vous à la base de données locale ou SQL Express, en fonction de celle que vous avez installée

Nous pouvons maintenant inspecter le schéma créé par Code First.



DbContext a utilisé les classes à inclure dans le modèle en examinant les propriétés DbSet que nous avons définies. Il utilise ensuite l'ensemble de conventions d'Code First par défaut pour déterminer les noms de tables et de colonnes, déterminer les types de données, rechercher les clés primaires, etc. Plus loin dans cette procédure pas à pas, nous allons voir comment vous pouvez remplacer ces conventions.

## 5. traitement des modifications de modèle

À présent, il est temps d'apporter des modifications à notre modèle, lorsque nous effectuons ces modifications, nous devons également mettre à jour le schéma de base de données. Pour ce faire, nous allons utiliser une fonctionnalité appelée Migrations Code First, ou migrations pour Short.

Les migrations nous permettent d'avoir un ensemble ordonné d'étapes qui décrivent comment mettre à niveau (et rétrograder) notre schéma de base de données. Chacune de ces étapes, appelée migration, contient du code qui décrit les modifications à appliquer.

La première étape consiste à activer Migrations Code First pour notre BloggingContext.

- **Outils-gestionnaire de package de la bibliothèque>-> console du gestionnaire de package**
- Exécutez la commande **Enable-Migrations** dans la Console du Gestionnaire de Package
- Un nouveau dossier migrations a été ajouté à notre projet qui contient deux éléments :
  - **Configuration.cs** : ce fichier contient les paramètres que les migrations vont utiliser pour la migration de BloggingContext. Nous n'avons pas besoin de modifier quoi que ce soit pour cette procédure pas à pas, mais ici, vous pouvez spécifier des données de départ, inscrire des fournisseurs pour d'autres bases de données, modifier l'espace de noms dans lequel les migrations sont générées, etc.

- <timestamp>\_InitialCreate.cs – il s'agit de votre première migration, il représente les modifications qui ont déjà été appliquées à la base de données pour la faire passer d'une base de données vide à une base de données qui comprend les blogs et les tables des publications. Bien que nous puissions laisser Code First créer automatiquement ces tables pour nous, maintenant que nous avons choisi des migrations, elles ont été converties en migration. Code First a également enregistré dans notre base de données locale que cette migration a déjà été appliquée. L'horodateur sur le nom de fichier est utilisé à des fins de classement.

Nous allons maintenant apporter une modification à notre modèle, ajouter une propriété URL à la classe de blog :

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public virtual List<Post> Posts { get; set; }
}
```

- Exécutez la commande **Add-migration AddUrl** dans la console du gestionnaire de package. La commande Add-migration vérifie les modifications apportées depuis votre dernière migration et génère une nouvelle migration avec les modifications détectées. Nous pouvons attribuer un nom à la migration. dans ce cas, nous appelons la migration « AddUrl ». Le code de génération de modèles automatique indique que nous devons ajouter une colonne d'URL, qui peut contenir des données de chaîne, au dbo.Table blogs. Si nécessaire, nous pourrions modifier le code de génération de modèles automatique, mais cela n'est pas nécessaire dans ce cas.

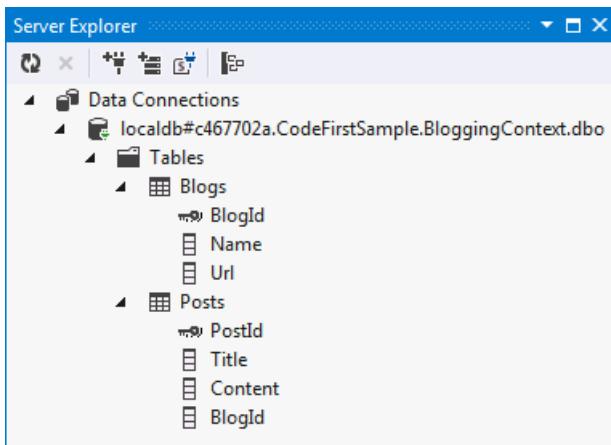
```
namespace CodeFirstNewDatabaseSample.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddUrl : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}
```

- Exécutez la commande **Update-Database** dans la console du gestionnaire de package. Cette commande applique toutes les migrations en attente à la base de données. Notre migration InitialCreate a déjà été appliquée afin que les migrations appliquent simplement notre nouvelle migration AddUrl. Conseil : vous pouvez utiliser le commutateur – **Verbose** lors de l'appel de Update-Database pour voir le SQL en cours d'exécution sur la base de données.

La nouvelle colonne URL est maintenant ajoutée à la table blogs dans la base de données :



## 6. Annotations de données

Jusqu'à présent, nous permettons à EF de découvrir le modèle à l'aide de ses conventions par défaut, mais il y a des moments où nos classes ne suivent pas les conventions et nous devons être en mesure d'effectuer une configuration supplémentaire. Il existe deux options pour cela ; Nous allons examiner les annotations de données dans cette section, puis l'API Fluent dans la section suivante.

- Nous allons ajouter une classe d'utilisateur à notre modèle

```
public class User
{
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

- Nous devons également ajouter un ensemble à notre contexte dérivé

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }
}
```

- Si nous avons essayé d'ajouter une migration, nous obtenons une erreur indiquant «*EntityType 'User' n'a aucune clé définie. Définissez la clé pour cet EntityType.*» comme EF n'a aucun moyen de savoir que le nom d'utilisateur doit être la clé primaire de l'utilisateur.
- Nous allons utiliser des annotations de données maintenant afin d'ajouter une instruction using en haut de Program.cs

```
using System.ComponentModel.DataAnnotations;
```

- Annotez maintenant la propriété UserName pour identifier qu'il s'agit de la clé primaire

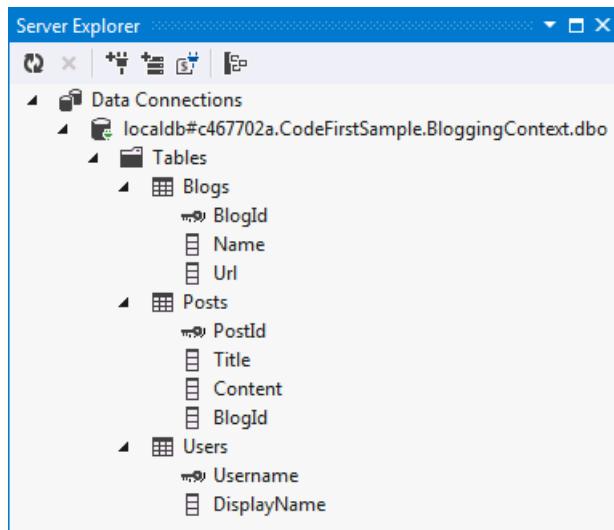
```
public class User
{
    [Key]
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

- Utilisez la commande **Add-migration adduser** pour générer automatiquement la structure d'une migration

et appliquer ces modifications à la base de données.

- Exécutez la commande **Update-Database** pour appliquer la nouvelle migration à la base de données.

La nouvelle table est maintenant ajoutée à la base de données :



La liste complète des annotations prises en charge par EF est la suivante :

- [KeyAttribute](#)
- [StringLengthAttribute](#)
- [MaxLengthAttribute](#)
- [ConcurrencyCheckAttribute](#)
- [RequiredAttribute](#)
- [TimestampAttribute](#)
- [ComplexTypeAttribute](#)
- [ColumnAttribute](#)
- [TableAttribute](#)
- [InversePropertyAttribute](#)
- [ForeignKeyAttribute](#)
- [DatabaseGeneratedAttribute](#)
- [NotMappedAttribute](#)

## 7. API Fluent

Dans la section précédente, nous avons vu comment utiliser des annotations de données pour compléter ou remplacer ce qui a été détecté par Convention. L'autre façon de configurer le modèle consiste à utiliser l'API Fluent Code First.

La plupart des configurations de modèle peuvent être effectuées à l'aide d'annotations de données simples. L'API Fluent est un moyen plus avancé de spécifier une configuration de modèle qui couvre tout ce que les annotations de données peuvent faire en plus d'une configuration plus avancée qui n'est pas possible avec les annotations de données. Les annotations de données et l'API Fluent peuvent être utilisées ensemble.

Pour accéder à l'API Fluent, vous devez substituer la méthode `OnModelCreating` dans `DbContext`. Supposons que nous voulions renommer la colonne dans laquelle `User`.`DisplayName` est stocké pour afficher\_`nom`.

- Remplacez la méthode `OnModelCreating` sur `BloggingContext` par le code suivant :

```

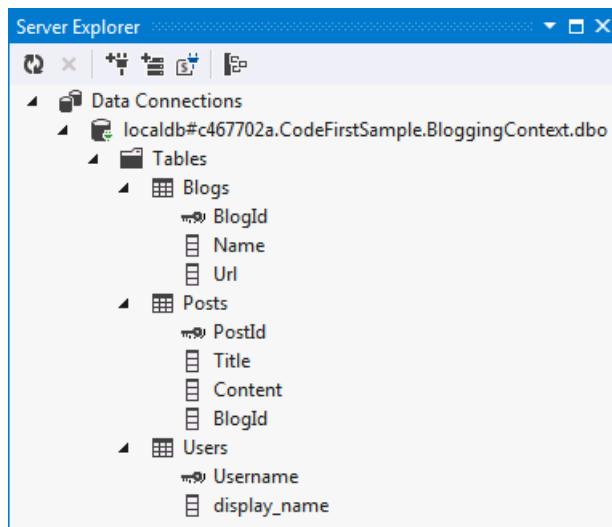
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<User> Users { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<User>()
            .Property(u => u.DisplayName)
            .HasColumnName("display_name");
    }
}

```

- Utilisez la commande **Add-migration ChangeDisplayName** pour générer automatiquement une migration pour appliquer ces modifications à la base de données.
- Exécutez la commande **Update-Database** pour appliquer la nouvelle migration à la base de données.

La colonne DisplayName est maintenant renommée pour afficher\_nom :



## Résumé

Dans cette procédure pas à pas, nous avons examiné Code First développement à l'aide d'une nouvelle base de données. Nous avons défini un modèle à l'aide de classes, puis j'ai utilisé ce modèle pour créer une base de données et stocker et récupérer des données. Une fois la base de données créée, nous avons utilisé Migrations Code First pour modifier le schéma à mesure de l'évolution du modèle. Nous avons également vu comment configurer un modèle à l'aide d'annotations de données et de l'API Fluent.

# Code First à une base de données existante

23/11/2019 • 10 minutes to read

Cette vidéo et la procédure pas à pas fournissent une introduction au développement Code First ciblant une base de données existante. Code First va tout d'abord vous permettre de définir votre modèle à l'aide de classes C# ou VB.Net. Éventuellement, une configuration supplémentaire peut être effectuée à l'aide d'attributs sur vos classes et propriétés ou à l'aide d'une API Fluent.

## Regarder la vidéo

Cette vidéo est [désormais disponible sur Channel 9](#).

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous devez avoir installé **Visual Studio 2012** ou **Visual Studio 2013**.

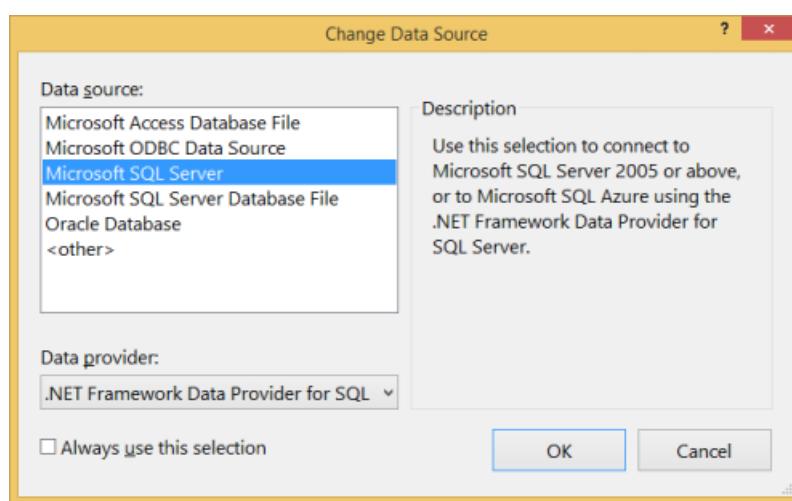
Vous aurez également besoin de la version **6.1** (ou ultérieure) du **Entity Framework Tools pour Visual Studio**. Consultez [obtenir des Entity Framework](#) pour plus d'informations sur l'installation de la dernière version du Entity Framework Tools.

## 1. créer une base de données existante

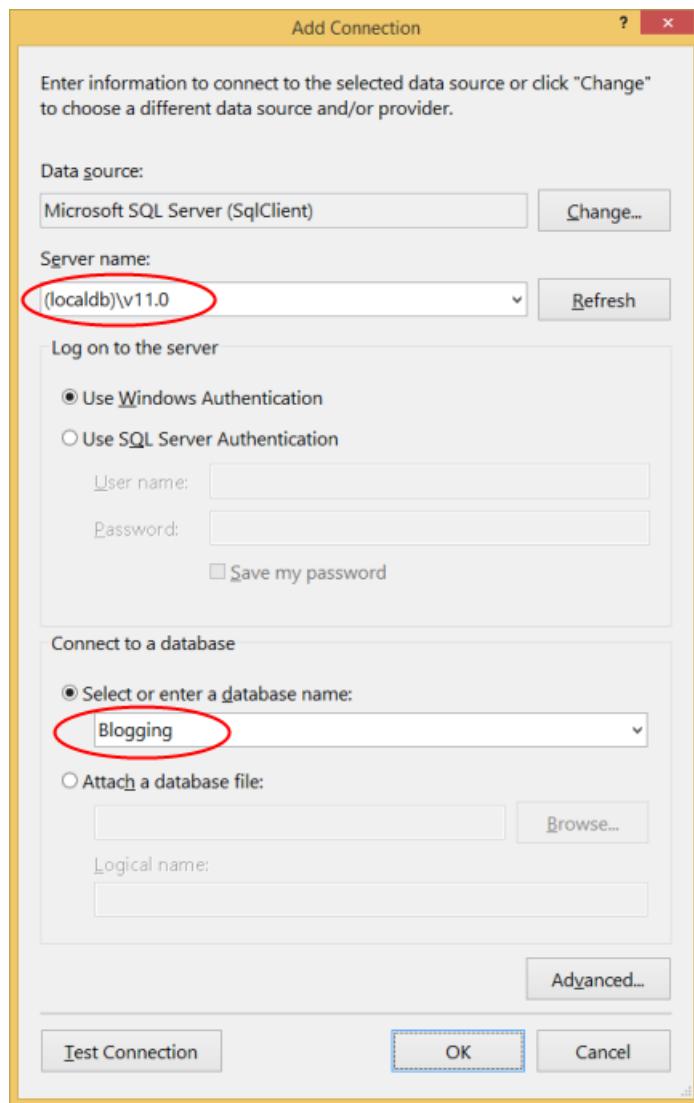
En général, lorsque vous ciblez une base de données existante, elle est déjà créée, mais pour cette procédure pas à pas, nous devons créer une base de données à laquelle accéder.

Commençons par générer la base de données.

- Ouvrez Visual Studio
- **Vue-> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **connexions de données-> ajouter une connexion...**
- Si vous n'êtes pas connecté à une base de données à partir de **Explorateur de serveurs** avant de devoir sélectionner **Microsoft SQL Server** comme source de données



- Connectez-vous à votre instance de base de données locale et entrez les **blogs** comme nom de base de données



- Sélectionnez **OK**. vous serez invité à créer une nouvelle base de données, sélectionnez **Oui** .



- La nouvelle base de données s'affiche alors dans Explorateur de serveurs, cliquez dessus avec le bouton droit et sélectionnez **nouvelle requête** .
- Copiez le code SQL suivant dans la nouvelle requête, cliquez avec le bouton droit sur la requête et sélectionnez **exécuter** .

```

CREATE TABLE [dbo].[Blogs] (
    [BlogId] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url] NVARCHAR (200) NULL,
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [dbo].[Posts] (
    [PostId] INT IDENTITY (1, 1) NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [dbo].[Blogs] ([BlogId]) ON
DELETE CASCADE
);

INSERT INTO [dbo].[Blogs] ([Name],[Url])
VALUES ('The Visual Studio Blog', 'http://blogs.msdn.com/visualstudio/')

INSERT INTO [dbo].[Blogs] ([Name],[Url])
VALUES ('.NET Framework Blog', 'http://blogs.msdn.com/dotnet/')

```

## 2. créer l'application

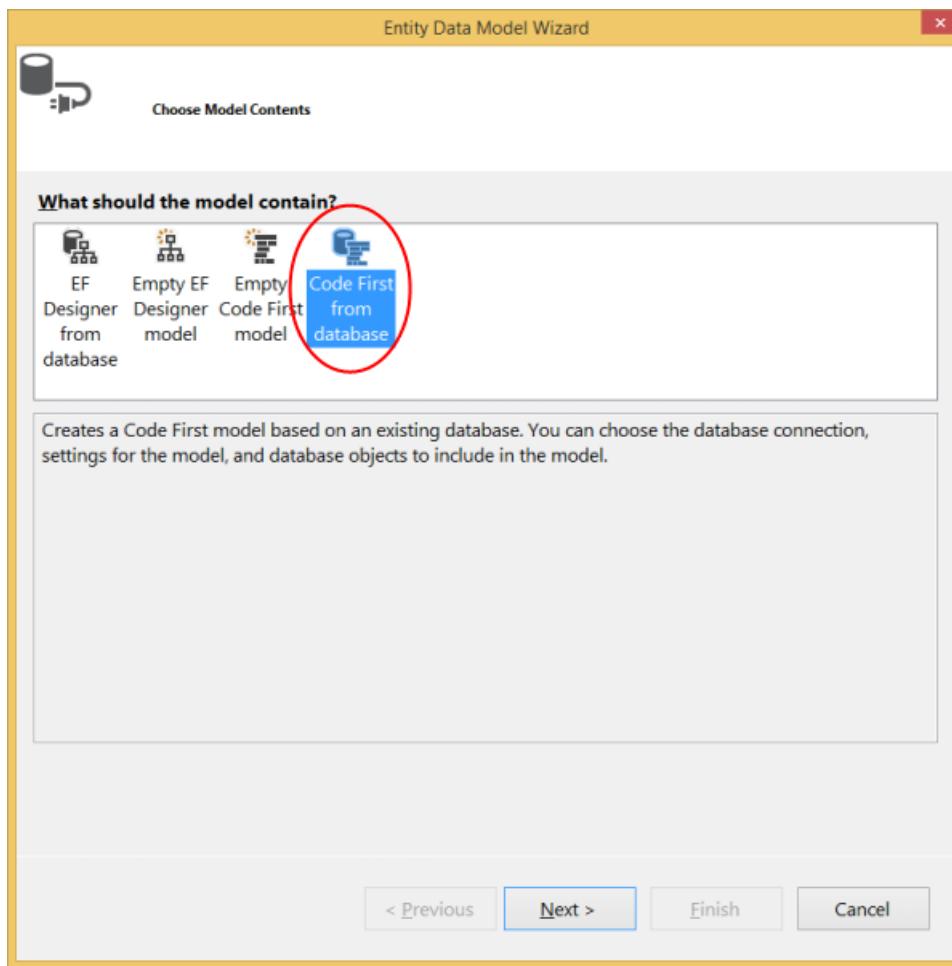
Pour simplifier les choses, nous allons créer une application console de base qui utilise Code First pour effectuer l'accès aux données :

- Ouvrez Visual Studio
- **Fichier> nouveau>...**
- Sélectionnez **Windows** dans le menu de gauche et dans l'**application console** .
- Entrez **CodeFirstExistingDatabaseSample** comme nom
- Sélectionnez **OK**.

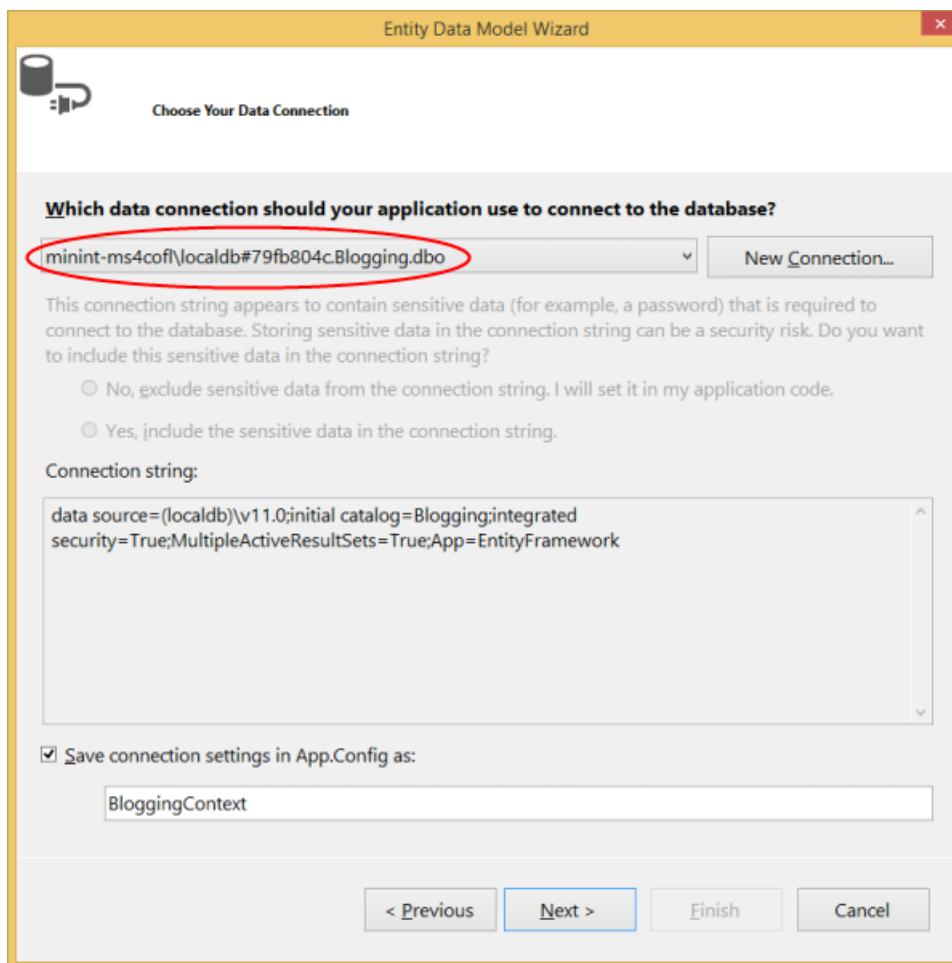
## 3. modèle d'ingénierie à rebours

Nous allons utiliser le Entity Framework Tools pour Visual Studio pour nous aider à générer du code initial à mapper à la base de données. Ces outils génèrent simplement du code que vous pouvez également taper manuellement si vous préférez.

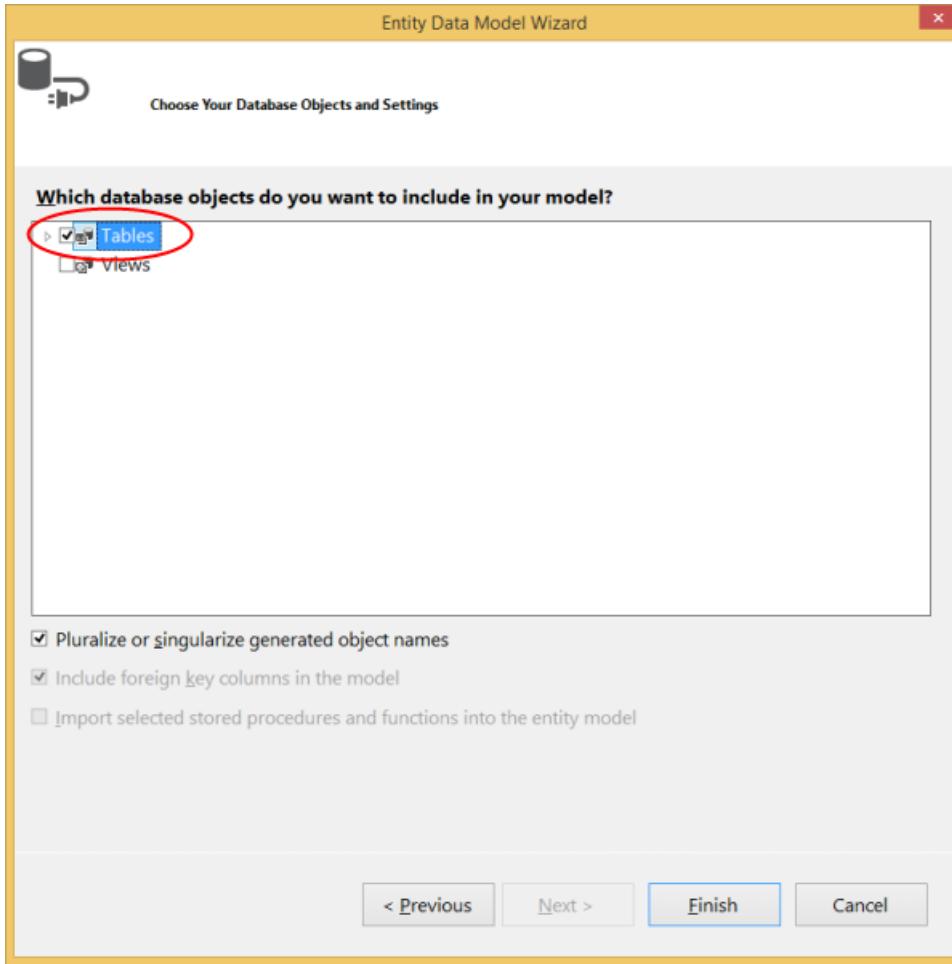
- **Projet-> ajouter un nouvel élément...**
- Sélectionnez **données** dans le menu de gauche, puis **ADO.NET Entity Data Model**
- Entrez **BloggingContext** comme nom et cliquez sur **OK** .
- Cela lance l' **assistant Entity Data Model**
- Sélectionnez **Code First dans la base de données** , puis cliquez sur **suivant** .



- Sélectionnez la connexion à la base de données que vous avez créée dans la première section, puis cliquez sur **suivant**.



- Cochez la case en regard de **tables** pour importer toutes les tables, puis cliquez sur **Terminer**.



Une fois que le processus d'ingénierie à rebours est terminé, un certain nombre d'éléments ont été ajoutés au projet, examinons ce qui a été ajouté.

### fichier de configuration

Un fichier app.config a été ajouté au projet, ce fichier contient la chaîne de connexion à la base de données existante.

```
<connectionStrings>
  <add
    name="BloggingContext"
    connectionString="data source=(localdb)\mssqllocaldb;initial catalog=Blogging;integrated
    security=True;MultipleActiveResultSets=True;App=EntityFramework"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

*Vous remarquerez également d'autres paramètres dans le fichier de configuration. il s'agit de paramètres EF par défaut qui indiquent Code First où créer les bases de données. Étant donné que nous allons mapper à une base de données existante, ces paramètres seront ignorés dans notre application.*

### Contexte dérivé

Une classe **BloggingContext** a été ajoutée au projet. Le contexte représente une session avec la base de données, ce qui nous permet d'interroger et d'enregistrer des données. Le contexte expose un **DbSet<tente>** pour chaque type dans notre modèle. Vous remarquerez également que le constructeur par défaut appelle un constructeur de base à l'aide de la syntaxe **Name = .** Cela indique Code First que la chaîne de connexion à utiliser pour ce contexte doit être chargée à partir du fichier de configuration.

```

public partial class BloggingContext : DbContext
{
    public BloggingContext()
        : base("name=BloggingContext")
    {
    }

    public virtual DbSet<Blog> Blogs { get; set; }
    public virtual DbSet<Post> Posts { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
    }
}

```

*Vous devez toujours utiliser la syntaxe **Name** = lorsque vous utilisez une chaîne de connexion dans le fichier de configuration. Cela garantit que si la chaîne de connexion n'est pas présente, Entity Framework lève une exception au lieu de créer une nouvelle base de données par Convention.*

### Classes de modèle

Enfin, un **blog** et une classe de **publication** ont également été ajoutés au projet. Il s'agit des classes de domaine qui composent notre modèle. Vous verrez des annotations de données appliquées aux classes pour spécifier la configuration dans laquelle les conventions d'Code First ne sont pas alignées sur la structure de la base de données existante. Par exemple, vous verrez l'annotation **StringLength** sur **blog.Name** et **blog.URL**, car ils ont une longueur maximale de **200** dans la base de données (la code First par défaut est d'utiliser la longueur maximale prise en charge par le fournisseur de base de données- **nvarchar (max)** dans SQL Server).

```

public partial class Blog
{
    public Blog()
    {
        Posts = new HashSet<Post>();
    }

    public int BlogId { get; set; }

    [StringLength(200)]
    public string Name { get; set; }

    [StringLength(200)]
    public string Url { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}

```

## 4. lecture & écriture de données

Maintenant que nous avons un modèle, il est temps de l'utiliser pour accéder à certaines données. Implémentez la méthode **main** dans **Program.cs** comme indiqué ci-dessous. Ce code crée une nouvelle instance de notre contexte, puis l'utilise pour insérer un nouveau **blog**. Il utilise ensuite une requête LINQ pour récupérer tous les **blogs** de la base de données classée par ordre alphabétique par **titre**.

```

class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}

```

Vous pouvez maintenant exécuter l'application et la tester.

```

Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
.NET Framework Blog
ADO.NET Blog
The Visual Studio Blog
Press any key to exit...

```

## Que se passe-t-il si ma base de données change ?

L'Assistant Code First à la base de données est conçu pour générer un ensemble de classes de point de départ que vous pouvez ensuite ajuster et modifier. Si votre schéma de base de données change, vous pouvez modifier manuellement les classes ou exécuter un autre ingénieur inverse pour remplacer les classes.

## Utilisation de Migrations Code First à une base de données existante

Si vous souhaitez utiliser Migrations Code First avec une base de données existante, consultez [migrations code First à une base de données existante](#).

## Résumé

Dans cette procédure pas à pas, nous avons examiné Code First développement à l'aide d'une base de données existante. Nous avons utilisé le Entity Framework Tools pour Visual Studio pour rétroconcevoir un ensemble de classes mappées à la base de données et pouvant être utilisé pour stocker et récupérer des données.

# Annotations de données Code First

17/07/2019 • 30 minutes to read

## NOTE

**EF4.1 et versions ultérieures uniquement** -les fonctionnalités, API, etc. abordés dans cette page ont été introduits dans Entity Framework 4.1. Si vous utilisez une version antérieure, tout ou partie de ces informations ne s'applique pas.

Le contenu de cette page est adapté à partir d'un article écrit à l'origine par Julie Lerman (<<http://thedatafarm.com>>).

Entity Framework Code First vous permet d'utiliser vos propres classes de domaine pour représenter le modèle EF s'appuie sur pour exécuter des requêtes sur, modifier, de suivi et la mise à jour des fonctions. Code profite tout d'abord un modèle de programmation appelé « convention sur configuration ». Code suppose tout d'abord que vos classes respectent les conventions d'Entity Framework et dans ce cas, fonctionneront automatiquement comment effectuer son travail. Toutefois, si vos classes ne suivent pas ces conventions, vous avez la possibilité d'ajouter des configurations à vos classes afin de fournir d'EF avec les informations requises.

Tout d'abord les code vous offre deux façons d'ajouter ces configurations à vos classes. Une utilise des attributs simples appelés DataAnnotations, et le second est à l'aide API du Code First Fluent, qui vous offre un moyen de décrire des configurations de manière impérative dans le code.

Cet article se concentrera sur l'utilisation de DataAnnotations (dans l'espace de noms System.ComponentModel.DataAnnotations) pour configurer vos classes – les configurations plus fréquemment requises de mise en surveillance. DataAnnotations sont également comprises par un nombre d'applications .NET, telles qu'ASP.NET MVC qui autorise ces applications d'exploiter les mêmes annotations de validations côté client.

## Le modèle

Je vais vous montrer le Code de première DataAnnotations avec une simple paire de classes : Blog et Post.

```
public class Blog
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime DateCreated { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    public ICollection<Comment> Comments { get; set; }
}
```

Lorsqu'ils sont, les classes de Blog et Post facilement suivent la convention de premier code et ne nécessitent aucun ajustement de l'activer sur la compatibilité EF. Toutefois, vous pouvez également utiliser les annotations pour fournissent des informations supplémentaires sur les classes et de la base de données auxquelles elles sont

mappées à EF.

## Clé

Entity Framework s'appuie sur chaque entité ayant une valeur de clé qui est utilisée pour l'entité de suivi. Une convention de Code First est les propriétés de clé implicites ; Code tout d'abord recherchera une propriété nommée « Id » ou une combinaison de nom de classe et « Id », tels que « BlogId ». Cette propriété doit être mappée à une colonne de clé primaire dans la base de données.

Les classes Blog et Post suivent cette convention. Que se passe-t-il si ce n'était pas ? Que se passe-t-il si Blog utilisé le nom *PrimaryTrackingKey* au lieu de cela, ou même *foo*? Si le code tout d'abord ne trouve pas d'une propriété qui correspond à cette convention lève une exception en raison de l'exigence d'Entity Framework que vous devez disposer d'une propriété de clé. Vous pouvez utiliser l'annotation de clé pour spécifier quelle propriété doit être utilisé comme valeur EntityKey.

```
public class Blog
{
    [Key]
    public int PrimaryTrackingKey { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public virtual ICollection<Post> Posts { get; set; }
}
```

Si vous êtes tout d'abord à l'aide de code est fonction de génération de base de données, la table de Blog aura une colonne clé primaire nommée *PrimaryTrackingKey*, qui est également défini en tant qu'identité par défaut.

```
dbo.Blogs
    Columns
        PrimaryTrackingKey (PK, int, not null)
        Title (nvarchar(128), null)
        BloggerName (nvarchar(128), null)
```

## Clés composites

Entity Framework prend en charge les clés composites - clés primaires qui se composent de plusieurs propriétés. Par exemple, vous pourriez avoir une classe de Passport dont la clé primaire est une combinaison de *PassportNumber* et *IssuingCountry*.

```
public class Passport
{
    [Key]
    public int PassportNumber { get; set; }
    [Key]
    public string IssuingCountry { get; set; }
    public DateTime Issued { get; set; }
    public DateTime Expires { get; set; }
}
```

Tentative d'utilisation de la classe ci-dessus dans votre modèle EF entraînerait une `InvalidOperationException` :

*Impossible de déterminer le composite primaire tri par clé pour le type « Passport ». Utilisez la ColumnAttribute ou la méthode HasKey pour spécifier l'ordre des clés primaires composites.*

Pour pouvoir utiliser les clés composites, Entity Framework, vous devez définir un ordre pour les propriétés de clé. Pour cela, à l'aide de l'annotation de colonne pour spécifier un ordre.

### NOTE

La valeur d'ordre est relative (plutôt qu'un index est basé) pour toutes les valeurs puissent être utilisées. Par exemple, 100 et 200 serait acceptable à la place de 1 et 2.

```
public class Passport
{
    [Key]
    [Column(Order=1)]
    public int PassportNumber { get; set; }
    [Key]
    [Column(Order = 2)]
    public string IssuingCountry { get; set; }
    public DateTime Issued { get; set; }
    public DateTime Expires { get; set; }
}
```

Si vous avez des entités avec des clés étrangères composites, vous devez spécifier la même colonne de classement que vous avez utilisé pour les propriétés de clé primaire correspondantes.

Uniquement l'ordre relatif dans les propriétés de clé étrangères doit être le même, les valeurs exactes affectés à **ordre** n'avez pas besoin de correspondre. Par exemple, dans la classe suivante, 3 et 4 peut servir à la place de 1 et 2.

```
public class PassportStamp
{
    [Key]
    public int StampId { get; set; }
    public DateTime Stamped { get; set; }
    public string StampingCountry { get; set; }

    [ForeignKey("Passport")]
    [Column(Order = 1)]
    public int PassportNumber { get; set; }

    [ForeignKey("Passport")]
    [Column(Order = 2)]
    public string IssuingCountry { get; set; }

    public Passport Passport { get; set; }
}
```

## Obligatoire

L'annotation requise indique à EF qu'une propriété particulière est requise.

Ajout nécessaire pour la propriété Title forcera EF (et MVC) pour vous assurer que la propriété comporte des données.

```
[Required]
public string Title { get; set; }
```

Sans code supplémentaire ni modifications de balisage dans l'application, une application MVC effectue la validation côté client, création même dynamique d'un message en utilisant les noms de propriété et d'annotation.

## Create

Blog

Title  
 The Title field is required.

BloggerName  
 Julie

L'attribut Required affecte également la base de données générée en effectuant la propriété mappée non nullable. Vérifiez que le champ titre est passé à « not null ».

### NOTE

Dans certains cas il ne peut pas être possible pour la colonne dans la base de données soit non nullable même si la propriété est requise. Par exemple, quand à l'aide de données de stratégie de l'héritage TPH pour plusieurs types est stockée dans une table unique. Si un type dérivé inclut une propriété obligatoire, que la colonne ne peut pas être rendue non nullable comme pas tous les types dans la hiérarchie ont cette propriété.

|  |  |  |
|--|--|--|
|  |  | dbo.Blogs                              |
|  |  | Columns                                |
|  |  | PrimaryTrackingKey (PK, int, not null) |
|  |  | Title (nvarchar(128), not null)        |
|  |  | BloggerName (nvarchar(128), null)      |

## MaxLength et MinLength

Les attributs MaxLength et MinLength autorisent vous permettent de spécifier des validations de propriété supplémentaires, comme vous le faisiez avec requis.

Voici le BloggerName avec les exigences de longueur. L'exemple montre également comment combiner des attributs.

```
[MaxLength(10),MinLength(5)]  
public string BloggerName { get; set; }
```

L'annotation MaxLength aura un impact sur la base de données en définissant la longueur de la propriété à 10.

|  |  |  |
|--|--|--|
|  |  | Columns                                |
|  |  | PrimaryTrackingKey (PK, int, not null) |
|  |  | Title (nvarchar(128), not null)        |
|  |  | BloggerName (nvarchar(10), null)       |

Annotation de côté client MVC et annotation de côté serveur EF 4.1 seront honorer cette validation, création à nouveau dynamique d'un message d'erreur : « Le champ BloggerName doit être un type de chaîne ou tableau avec une longueur maximale de « 10 ». » Ce message est un peu long. Nombre illimité d'annotations vous permettre de spécifier un message d'erreur avec l'attribut de message d'erreur.

```
[MaxLength(10, ErrorMessage="BloggerName must be 10 characters or less"),MinLength(5)]
public string BloggerName { get; set; }
```

Vous pouvez également spécifier le message d'erreur dans l'annotation requise.

## Create

Blog

Title

BloggerName  
 BloggerName must be 10 characters or less

## NotMapped

Convention de premier code impose que chaque propriété d'un type de données pris en charge est représentée dans la base de données. Mais ce n'est pas toujours le cas dans vos applications. Par exemple, vous pouvez avoir une propriété dans la classe de Blog qui crée un code basé sur les champs titre et BloggerName. Cette propriété peut être créée dynamiquement et ne doit pas être stocké. Vous pouvez marquer toutes les propriétés qui ne correspondent pas à la base de données avec l'annotation NotMapped telle que cette propriété BlogCode.

```
[NotMapped]
public string BlogCode
{
    get
    {
        return Title.Substring(0, 1) + ":" + BloggerName.Substring(0, 1);
    }
}
```

## ComplexType

Il n'est pas rare de décrire vos entités de domaine sur un ensemble de classes et de couche ensuite ces classes pour décrire une entité complète. Par exemple, vous pouvez ajouter une classe appelée BlogDetails à votre modèle.

```
public class BlogDetails
{
    public DateTime? DateCreated { get; set; }

    [MaxLength(250)]
    public string Description { get; set; }
}
```

Notez que BlogDetails n'a pas de n'importe quel type de propriété de clé. Dans la conception conduite par domaine, BlogDetails est appelé pour un objet de valeur. Entity Framework fait référence aux objets de valeur comme des types complexes. Les types complexes ne peuvent pas être suivies sur leurs propres.

Toutefois en tant que propriété dans la classe de Blog, BlogDetails il sera suivie dans le cadre d'un objet de Blog.

Dans l'ordre pour code first pour reconnaître de cela, vous devez marquer la classe BlogDetails comme un ComplexType.

```
[ComplexType]
public class BlogDetails
{
    public DateTime? DateCreated { get; set; }

    [MaxLength(250)]
    public string Description { get; set; }
}
```

Vous pouvez maintenant ajouter une propriété dans la classe de Blog pour représenter le BlogDetails pour ce blog.

```
public BlogDetails BlogDetail { get; set; }
```

Dans la base de données, la table de Blog contiendra toutes les propriétés du blog, y compris les propriétés contenues dans sa propriété BlogDetail. Par défaut, chacun d'eux est précédé par le nom du type complexe, BlogDetail.

```
dbo.Blogs
└─ Columns
    PrimaryTrackingKey (PK, int, not null)
    Title (nvarchar(128), not null)
    BloggerName (nvarchar(10), null)
    BlogDetail_DateCreated (datetime, null)
    BlogDetail_Description (nvarchar(250), null)
```

## ConcurrencyCheck

L'annotation ConcurrencyCheck vous permet de marquer une ou plusieurs propriétés à utiliser pour l'accès concurrentiel dans la base de données lorsqu'un utilisateur modifie ou supprime une entité. Si vous avez travaillé avec le Concepteur EF, celui-ci s'aligne définition ConcurrencyMode d'une propriété sur Fixed.

Nous allons voir comment ConcurrencyCheck fonctionne en l'ajoutant à la propriété BloggerName.

```
[ConcurrencyCheck, MaxLength(10, ErrorMessage="BloggerName must be 10 characters or less"), MinLength(5)]
public string BloggerName { get; set; }
```

Lorsque SaveChanges est appelée, en raison de l'annotation ConcurrencyCheck sur le champ BloggerName, la valeur d'origine de cette propriété sera utilisée dans la mise à jour. La commande tente de localiser la ligne correcte par filtrage non seulement sur la valeur de clé, mais également sur la valeur d'origine de BloggerName. Voici les parties critiques de la commande de mise à jour envoyées à la base de données, où vous pouvez voir la commande met à jour la ligne qui a un PrimaryTrackingKey est 1 et un BloggerName de « Julie » qui était la valeur d'origine quand ce blog a été récupéré à partir de la base de données.

```
where ([PrimaryTrackingKey] = @4) and ([BloggerName] = @5)
@4=1,@5=N'Julie'
```

Si quelqu'un a modifié le nom de blogueur pour ce blog en attendant, cette mise à jour échoue et vous obtiendrez une DbUpdateConcurrencyException dont vous aurez besoin pour gérer.

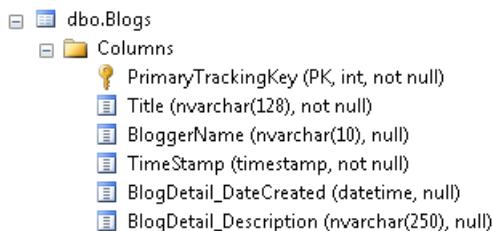
## TimeStamp

Il est plus courant d'utiliser des champs rowversion ou timestamp pour le contrôle d'accès concurrentiel. Mais au lieu d'utiliser l'annotation ConcurrencyCheck, vous pouvez utiliser l'annotation d'horodateur plus spécifique, que le type de la propriété est le tableau d'octets. Code tout d'abord traite les propriétés horodatage exactement en tant que propriétés de ConcurrencyCheck, mais il garantit également que le champ de base de données qui génère d'abord code est non nullable. Vous ne pouvez avoir qu'une seule propriété d'horodatage dans une classe donnée.

Ajout de la propriété suivante à la classe de Blog :

```
[Timestamp]  
public Byte[] TimeStamp { get; set; }
```

résultats dans le code créant d'abord une colonne timestamp non null dans la table de base de données.



```
dbo.Blogs  
Columns  
PrimaryTrackingKey (PK, int, not null)  
Title (nvarchar(128), not null)  
BloggerName (nvarchar(10), null)  
TimeStamp (timestamp, not null)  
BlogDetail_DateCreated (datetime, null)  
BlogDetail_Description (nvarchar(250), null)
```

## Table et colonne

Si vous permettez à Code First créer la base de données, vous souhaiterez modifier le nom des tables et des colonnes en cours de création. Vous pouvez également utiliser Code First avec une base de données existante. Mais il n'est pas toujours le cas que les noms des classes et des propriétés dans votre domaine correspondent aux noms des tables et des colonnes dans votre base de données.

Ma classe est nommée Blog et par convention, code tout d'abord part du principe que cela permet de mapper vers une table nommée Blogs. Si tel n'est pas le cas, vous pouvez spécifier le nom de la table avec l'attribut de la Table. Ici, par exemple, l'annotation Spécifie que le nom de table est InternalBlogs.

```
[Table("InternalBlogs")]  
public class Blog
```

L'annotation de la colonne est un adepte plus en spécifiant les attributs d'une colonne mappée. Vous pouvez stipuler un nom, type de données ou même l'ordre dans lequel une colonne apparaît dans la table. Voici un exemple de l'attribut de colonne.

```
[Column("BlogDescription", TypeName="ntext")]  
public String Description {get;set;}
```

Ne confondez pas attribut TypeName de la colonne avec le DataType DataAnnotation. Type de données est une annotation utilisée pour l'interface utilisateur et est ignorée par Code First.

Voici la table une fois qu'il est régénéré. Le nom de la table a changé à InternalBlogs et colonne de Description à partir du type complexe est désormais BlogDescription. Étant donné que le nom a été spécifié dans l'annotation, code tout d'abord n'utilise pas la convention de commencer le nom de colonne avec le nom du type complexe.

|    |    |  |
|----|----|--|
| └─ | └─ | dbo.InternalBlogs                          |
| └─ | └─ | └─ Columns                                 |
| └─ | └─ | └─ PrimaryTrackingKey (PK, int, not null)  |
| └─ | └─ | └─ Title (nvarchar(128), not null)         |
| └─ | └─ | └─ BloggerName (nvarchar(10), null)        |
| └─ | └─ | └─ TimeStamp (timestamp, not null)         |
| └─ | └─ | └─ BlogDetail_DateCreated (datetime, null) |
| └─ | └─ | └─ BlogDescription (ntext, null)           |

## DatabaseGenerated

Une fonctionnalités de base de données important est la possibilité d'avoir les propriétés calculées. Si vous mappez vos classes de Code First pour les tables qui contiennent des colonnes calculées, vous ne voulez Entity Framework pour tenter de mettre à jour ces colonnes. Mais vous ne souhaitez pas que EF pour renvoyer ces valeurs à partir de la base de données une fois que vous avez inséré ou mis à jour des données. Vous pouvez utiliser l'annotation DatabaseGenerated pour signaler les propriétés dans votre classe, ainsi que l'énumération calculé. Autres énumérations sont None et d'identité.

```
[DatabaseGenerated(DatabaseGeneratedOption.Computed)]
public DateTime DateCreated { get; set; }
```

Vous pouvez utiliser la base de données générée sur les colonnes byte ou timestamp lorsque code génère tout d'abord la base de données, sinon vous devez uniquement l'utiliser en pointant sur les bases de données existantes, car le code ne sont pas tout d'abord être en mesure de déterminer la formule pour la colonne calculée.

Vous lisez supérieur à celui par défaut, une propriété de clé est un entier deviendra une clé d'identité dans la base de données. Qui serait le même que l'affectation DatabaseGenerated DatabaseGeneratedOption.Identity. Si vous ne souhaitez pas qu'il soit une clé d'identité, vous pouvez définir la valeur à DatabaseGeneratedOption.None.

## Index

### NOTE

**EF6.1 et versions ultérieures uniquement** -attribut de l'Index a été introduite dans Entity Framework 6.1. Si vous utilisez une version antérieure les informations contenues dans cette section ne s'applique pas.

Vous pouvez créer un index sur une ou plusieurs colonnes à l'aide de la **IndexAttribute**. Ajout de l'attribut à une ou plusieurs propriétés sera provoquer d'EF créer l'index correspondant dans la base de données lorsqu'il crée la base de données ou structurer le correspondantes **CreateIndex** appelle si vous utilisez Code First Migrations.

Par exemple, le code suivant entraîne un index créé sur le **évaluation** colonne de la **billets** table dans la base de données.

```

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    [Index]
    public int Rating { get; set; }
    public int BlogId { get; set; }
}

```

Par défaut, l'index sera nommé **IX\_<nom de la propriété>** (IX\_Rating dans l'exemple ci-dessus). Vous pouvez également spécifier un nom pour l'index cependant. L'exemple suivant spécifie que l'index doit être nommé **PostRatingIndex**.

```

[Index("PostRatingIndex")]
public int Rating { get; set; }

```

Par défaut, les index sont non uniques, mais vous pouvez utiliser la **IsUnique** nommé de paramètre pour spécifier qu'un index doit être unique. L'exemple suivant présente un index unique sur une **utilisateur** du nom de connexion.

```

public class User
{
    public int UserId { get; set; }

    [Index(IsUnique = true)]
    [StringLength(200)]
    public string Username { get; set; }

    public string DisplayName { get; set; }
}

```

### Index de plusieurs colonnes

Les index qui s'étendent sur plusieurs colonnes sont spécifiés à l'aide du même nom dans plusieurs annotations d'Index pour une table donnée. Lorsque vous créez des index multi-colonnes, vous devez spécifier un ordre pour les colonnes dans l'index. Par exemple, le code suivant crée un index multicolonnes sur **évaluation** et **BlogId** appelé **IX\_BlogIdAndRating**. **BlogId** est la première colonne dans l'index et **évaluation** est le deuxième.

```

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    [Index("IX_BlogIdAndRating", 2)]
    public int Rating { get; set; }
    [Index("IX_BlogIdAndRating", 1)]
    public int BlogId { get; set; }
}

```

## Attributs de relation : InverseProperty et ForeignKey

## NOTE

Cette page fournit des informations sur la configuration des relations dans votre modèle Code First à l'aide des Annotations de données. Pour obtenir des informations générales sur les relations dans Entity Framework et comment accéder à et manipuler des données à l'aide de relations, consultez [relations & Propriétés de Navigation.](#) \*

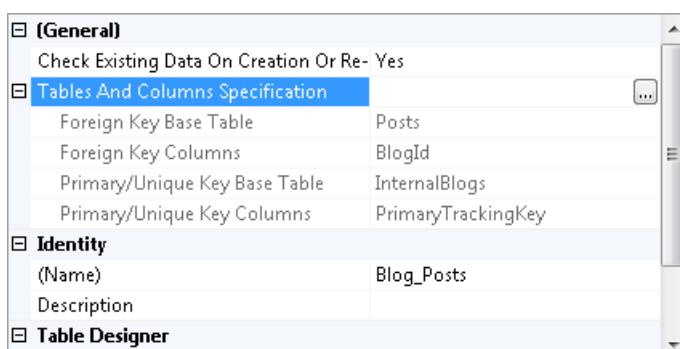
Convention de premier code s'occupe des relations plus courantes dans votre modèle, mais il existe quelques cas où il a besoin d'aide.

Modification du nom de la propriété de clé dans la classe Blog crée un problème avec sa relation à Post.

Lors de la génération de la base de données, code tout d'abord voit la propriété BlogId dans la classe de publication et la reconnaît, par la convention qu'elle correspond à un nom de classe et un « Id », comme une clé étrangère à la classe de Blog. Mais il n'existe aucune propriété BlogId dans la classe de blog. La solution consiste à créer une propriété de navigation dans le billet d'utiliser le DataAnnotation étrangère pour aider à code tout d'abord comprendre comment créer la relation entre les deux classes, à l'aide de la propriété Post.BlogId, ainsi que la façon de spécifier des contraintes dans la base de données.

```
public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime DateCreated { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    [ForeignKey("BlogId")]
    public Blog Blog { get; set; }
    public ICollection<Comment> Comments { get; set; }
}
```

La contrainte dans la base de données montre une relation entre InternalBlogs.PrimaryTrackingKey et Posts.BlogId.



Le InverseProperty est utilisé lorsque vous avez plusieurs relations entre les classes.

Dans la classe Post, voulez-vous effectuer le suivi de qui l'a écrit un billet de blog, ainsi que qui a été modifié. Voici deux nouvelles propriétés de navigation pour la classe de publication.

```
public Person CreatedBy { get; set; }
public Person UpdatedBy { get; set; }
```

Vous devrez également ajouter dans la classe Person référencée par ces propriétés. La classe Person a les propriétés de navigation à la publication, un pour tous les messages écrits par la personne et un pour tous les billets de mise à jour par cette personne.

```

public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }
    public List<Post> PostsWritten { get; set; }
    public List<Post> PostsUpdated { get; set; }
}

```

Tout d'abord code n'est pas capable de faire correspondre les propriétés dans les deux classes sur son propre. La table de base de données pour des publications doit avoir une clé étrangère pour la personne CreatedBy et un pour la personne UpdatedBy mais le code sera tout d'abord créer quatre propriétés de clé étrangère : Personne\_Id, personne\_Id1, CreatedBy\_Id et UpdatedBy\_ID.

```

[+] dbo.Posts
  [+] Columns
    Id (PK, int, not null)
    Title (nvarchar(128), null)
    DateCreated (datetime, not null)
    Content (nvarchar(128), null)
    BlogId (FK, int, not null)
    Person_Id (FK, int, null)
    Person_Id1 (FK, int, null)
    CreatedBy_Id (FK, int, null)
    UpdatedBy_Id (FK, int, null)

```

Pour résoudre ces problèmes, vous pouvez utiliser l'annotation `InverseProperty` pour spécifier l'alignement des propriétés.

```

[InverseProperty("CreatedBy")]
public List<Post> PostsWritten { get; set; }

[InverseProperty("UpdatedBy")]
public List<Post> PostsUpdated { get; set; }

```

Étant donné que la propriété `PostsWritten` en personne sait que cela fait référence au type de publication, il générera la relation à `Post.CreatedBy`. De même, `PostsUpdated` sera connecté à `Post.UpdatedBy`. Et tout d'abord code ne crée pas les clés étrangères supplémentaires.

```

[+] dbo.Posts
  [+] Columns
    Id (PK, int, not null)
    Title (nvarchar(128), null)
    DateCreated (datetime, not null)
    Content (nvarchar(128), null)
    BlogId (FK, int, not null)
    CreatedBy_Id (FK, int, null)
    UpdatedBy_Id (FK, int, null)

```

## Récapitulatif

DataAnnotations vous permettent non seulement de décrire la validation côté client et serveur dans vos classes de premier code, mais ils vous permettent également d'améliorer et même corriger les hypothèses code va d'abord faire sur vos classes selon ses conventions. Avec DataAnnotations, vous pouvez optimiser pas uniquement la génération de schéma de base de données, mais vous pouvez également mapper vos classes de premier code pour une base de données existante.

Pendant qu'elles sont très flexibles, n'oubliez pas que DataAnnotations fournir uniquement le meilleur parti généralement les modifications de configuration que vous pouvez effectuer sur vos classes de premier code

nécessaires. Pour configurer vos classes pour certains cas edge, vous devriez rechercher au mécanisme de configuration de remplacement API du Code First Fluent.

# Définition de DbSets

13/09/2018 • 3 minutes to read

Lors du développement avec le workflow de Code First que vous définissez un DbContext dérivé qui représente votre session avec la base de données et expose un DbSet pour chaque type dans votre modèle. Cette rubrique décrit les différentes méthodes que vous pouvez définir les propriétés DbSet.

## DbContext avec des propriétés DbSet

Le cas courant indiqué dans les exemples de Code First consiste à avoir un DbContext avec les propriétés publiques DbSet automatique pour les types d'entités de votre modèle. Exemple :

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
}
```

Lorsqu'il est utilisé en mode Code First, cette opération configure des Blogs et des publications en tant que types d'entité, comme la configuration est accessible à partir de ces autres types de. En outre, DbContext appelle automatiquement la méthode setter pour chacune de ces propriétés pour définir une instance de la DbSet appropriée.

## DbContext avec des propriétés IDbSet

Il existe des situations, par exemple créer des objets fictifs ou fakes, où il est plus utile déclarer des propriétés de votre jeu à l'aide d'une interface. Dans ce cas le IDbSet interface peut être utilisée à la place de DbSet. Exemple :

```
public class BloggingContext : DbContext
{
    public IDbSet<Blog> Blogs { get; set; }
    public IDbSet<Post> Posts { get; set; }
}
```

Ce contexte fonctionne dans exactement la même façon que le contexte qui utilise la classe DbSet pour ses propriétés de jeu.

## DbContext les propriétés définies en lecture seule

Si vous ne souhaitez pas exposer des accesseurs Set publics pour vos propriétés DbSet ou IDbSet, vous pouvez créer des propriétés en lecture seule et créer les instances vous-même. Exemple :

```
public class BloggingContext : DbContext
{
    public DbSet<Blog> Blogs
    {
        get { return Set<Blog>(); }
    }

    public DbSet<Post> Posts
    {
        get { return Set<Post>(); }
    }
}
```

Notez que DbContext met en cache l'instance de DbSet retourné à partir de la méthode Set afin que chacune de ces propriétés retournera la même instance chaque fois qu'elle est appelée.

Découverte de types d'entités pour Code First fonctionne de la même façon ici car il fait pour les propriétés avec accesseurs Get publique et Set.

# Prise en charge de l'enum - Code First

27/09/2018 • 8 minutes to read

## NOTE

**EF5 et versions ultérieures uniquement** -les fonctionnalités, API, etc. abordés dans cette page ont été introduits dans Entity Framework 5. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Cette procédure pas à pas vidéo et pas à pas montre comment utiliser les types enum avec Entity Framework Code First. Il montre également comment utiliser les énumérations dans une requête LINQ.

Cette procédure pas à pas utilise Code First pour créer une nouvelle base de données, mais vous pouvez également utiliser [Code First pour mapper à une base de données existante](#).

Prise en charge de l'énumération a été introduite dans Entity Framework 5. Pour utiliser les nouvelles fonctionnalités telles que les énumérations, les types de données spatiales et les fonctions table, vous devez cibler .NET Framework 4.5. Visual Studio 2012 cible .NET 4.5 par défaut.

Dans Entity Framework, une énumération peut avoir des types sous-jacents suivants : **octets**, **Int16**, **Int32**, **Int64**, ou **SByte**.

## Regardez la vidéo

Cette vidéo montre comment utiliser les types enum avec Entity Framework Code First. Il montre également comment utiliser les énumérations dans une requête LINQ.

**Présenté par:** Julia Kornich

**Vidéo:** [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

## Conditions préalables

Vous devez disposer de Visual Studio 2012, Ultimate, Premium, Professionnel ou Web Express edition est installé pour terminer cette procédure pas à pas.

## Configurer le projet

1. Ouvrez Visual Studio 2012
2. Sur le **fichier** menu, pointez sur **New**, puis cliquez sur **projet**
3. Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le **Console** modèle
4. Entrez **EnumCodeFirst** en tant que le nom du projet et cliquez sur **OK**

## Définir un nouveau modèle à l'aide de Code First

Lors de l'utilisation du développement Code First vous commencez généralement par écriture de classes .NET Framework qui définissent votre modèle conceptuel (domaine). Le code suivant définit la classe de service.

Le code définit également l'énumération DepartmentNames. Par défaut, l'énumération est de **int** type. La propriété nom de la classe de service est du type DepartmentNames.

Ouvrez le fichier Program.cs et collez les définitions de classe suivantes.

```

public enum DepartmentNames
{
    English,
    Math,
    Economics
}

public partial class Department
{
    public int DepartmentID { get; set; }
    public DepartmentNames Name { get; set; }
    public decimal Budget { get; set; }
}

```

## Définir le DbContext de Type dérivé

Outre la définition des entités, vous devez définir une classe qui dérive de `DbContext` et expose `DbSet< TEntity >` propriétés. Le `DbSet< TEntity >` propriétés permettent le contexte de connaître les types que vous souhaitez inclure dans le modèle.

Une instance du type `DbContext` dérivée gère les objets d'entité pendant l'exécution, ce qui inclut le remplissage des objets avec des données à partir d'une base de données, modifier le suivi et la persistance des données à la base de données.

Les types `DbContext` et `DbSet` sont définis dans l'assembly `EntityFramework`. Nous allons ajouter une référence à cette DLL à l'aide du package `EntityFramework NuGet`.

1. Dans l'Explorateur de solutions, cliquez sur le nom du projet.
2. Sélectionnez **gérer les Packages NuGet...**
3. Dans la boîte de dialogue Gérer les Packages NuGet, sélectionnez le **Online** onglet et sélectionnez le **EntityFramework** package.
4. Cliquez sur **installer**

Notez que l'assembly `EntityFramework`, en plus des références aux assemblies `System.ComponentModel.DataAnnotations` et `System.Data.Entity` sont également ajoutés.

En haut du fichier `Program.cs`, ajoutez le code suivant à l'aide d'instruction :

```
using System.Data.Entity;
```

Dans `Program.cs`, ajoutez la définition du contexte.

```

public partial class EnumTestContext : DbContext
{
    public DbSet<Department> Departments { get; set; }
}

```

## Conserver et récupérer des données

Ouvrez le fichier `Program.cs` dans lequel la méthode `Main` est définie. Ajoutez le code suivant dans la fonction `Main`. Le code ajoute un nouvel objet de service pour le contexte. Ensuite, il enregistre les données. Le code exécute également une requête LINQ qui retourne un département où le nom est `DepartmentNames.English`.

```

using (var context = new EnumTestContext())
{
    context.Departments.Add(new Department { Name = DepartmentNames.English });

    context.SaveChanges();

    var department = (from d in context.Departments
                      where d.Name == DepartmentNames.English
                      select d).FirstOrDefault();

    Console.WriteLine(
        "DepartmentID: {0} Name: {1}",
        department.DepartmentID,
        department.Name);
}

```

Compilez et exécutez l'application. Le programme génère la sortie suivante :

```
DepartmentID: 1 Name: English
```

## Afficher la base de données générée

Lorsque vous exécutez l'application la première fois, Entity Framework crée une base de données pour vous. Étant donné que nous Visual Studio 2012 est installé, la base de données est créée sur l'instance de base de données locale. Par défaut, Entity Framework nomme la base de données après le nom qualifié complet de contexte dérivé (pour cet exemple est **EnumCodeFirst.EnumTestContext**). Les fois suivantes, que la base de données existante sera utilisée.

Notez que si vous apportez des modifications à votre modèle une fois que la base de données a été créée, vous devez utiliser des Migrations Code First pour mettre à jour le schéma de base de données. Consultez [Code First pour une base de données](#) pour obtenir un exemple d'utilisation de Migrations.

Pour afficher la base de données et les données, procédez comme suit :

1. Dans le menu principal de Visual Studio 2012, sélectionnez **vue -> Explorateur d'objets SQL Server**.
2. Si la base de données locale n'est pas dans la liste des serveurs, cliquez sur le bouton droit de la souris sur **SQL Server** et sélectionnez **ajouter SQL Server** utiliser la valeur par défaut **l'authentification Windows** pour se connecter à la Instance de base de données locale
3. Développez le nœud de base de données locale
4. Déroulez la **bases de données** dossier pour voir la nouvelle base de données et accédez à la **département**  
Note de Code First ne crée pas une table qui mappe au type énumération de table
5. Pour afficher les données, avec le bouton droit sur la table et sélectionnez **afficher les données**

## Récapitulatif

Dans cette procédure pas à pas, nous avons vu comment utiliser les types enum avec Entity Framework Code First.

# Code First spatial

23/11/2019 • 9 minutes to read

## NOTE

**EF5 uniquement** : les fonctionnalités, les API, etc. présentées dans cette page ont été introduites dans Entity Framework 5. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

La vidéo et la procédure pas à pas montrent comment mapper des types spatiaux avec Entity Framework Code First. Il montre également comment utiliser une requête LINQ pour rechercher une distance entre deux emplacements.

Cette procédure pas à pas utilise Code First pour créer une base de données, mais vous pouvez également utiliser des [Code First à une base de données existante](#).

La prise en charge des types spatiaux a été introduite dans Entity Framework 5. Notez que pour utiliser les nouvelles fonctionnalités telles que le type spatial, les énumérations et les fonctions table, vous devez cibler .NET Framework 4,5. Visual Studio 2012 cible .NET 4,5 par défaut.

Pour utiliser des types de données spatiales, vous devez également utiliser un fournisseur de Entity Framework qui a une prise en charge spatiale. Pour plus d'informations, consultez [prise en charge des fournisseurs pour les types spatiaux](#).

Il existe deux types de données spatiales principales : Geography et Geometry. Le type de données geography stocke des données ellipsoïdal (par exemple, des coordonnées de latitude et de longitude GPS). Le type de données geometry représente le système de coordonnées euclidienne (Flat).

## Regarder la vidéo

Cette vidéo montre comment mapper des types spatiaux avec Entity Framework Code First. Il montre également comment utiliser une requête LINQ pour rechercher une distance entre deux emplacements.

**Présenté par:** Julia Kornich

**Vidéo:** [wmv](#) | [MP4](#) | [WMV \(zip\)](#)

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous devez avoir installé Visual Studio 2012, Ultimate, Premium, Professional ou Web Express Edition.

## Configurer le projet

1. Ouvrir Visual Studio 2012
2. Dans le menu **fichier**, pointez sur **nouveau**, puis cliquez sur **projet**.
3. Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le modèle **console**.
4. Entrez **SpatialCodeFirst** comme nom du projet, puis cliquez sur **OK**.

## Définir un nouveau modèle à l'aide de Code First

Lors de l'utilisation de Code First développement, vous commencez généralement par écrire des classes .NET

Framework qui définissent votre modèle conceptuel (domaine). Le code ci-dessous définit la classe University.

La propriété Location de l'Université est du type DbGeography. Pour utiliser le type DbGeography, vous devez ajouter une référence à l'assembly System. Data. Entity et ajouter également l'instruction using System. Data. spatial.

Ouvrez le fichier Program.cs et collez les instructions using suivantes en haut du fichier :

```
using System.Data.Spatial;
```

Ajoutez la définition de classe University suivante au fichier Program.cs.

```
public class University
{
    public int UniversityID { get; set; }
    public string Name { get; set; }
    public DbGeography Location { get; set; }
}
```

## Définir le type dérivé DbContext

En plus de définir des entités, vous devez définir une classe qui dérive de DbContext et expose les propriétés DbSet< TEntity >. Les propriétés DbSet< TEntity > permettent au contexte de savoir quels types vous souhaitez inclure dans le modèle.

Une instance du type dérivé DbContext gère les objets d'entité au moment de l'exécution, ce qui comprend le remplissage des objets avec les données d'une base de données, le suivi des modifications et la persistance des données dans la base de données.

Les types DbContext et DbSet sont définis dans l'assembly EntityFramework. Nous allons ajouter une référence à cette DLL à l'aide du package NuGet EntityFramework.

1. Dans Explorateur de solutions, cliquez avec le bouton droit sur le nom du projet.
2. Sélectionnez **gérer les packages NuGet...**
3. Dans la boîte de dialogue gérer les packages NuGet, sélectionnez l'onglet **en ligne** et choisissez le package **EntityFramework**.
4. Cliquez sur **installer**

Notez qu'en plus de l'assembly EntityFramework, une référence à l'assembly System. ComponentModel. DataAnnotations est également ajoutée.

En haut du fichier Program.cs, ajoutez l'instruction using suivante :

```
using System.Data.Entity;
```

Dans Program.cs, ajoutez la définition de contexte.

```
public partial class UniversityContext : DbContext
{
    public DbSet<University> Universities { get; set; }
}
```

## Conserver et récupérer des données

Ouvrez le fichier Program.cs dans lequel la méthode main est définie. Ajoutez le code suivant à la fonction main.

Le code ajoute deux nouveaux objets University au contexte. Les propriétés spatiales sont initialisées à l'aide de la méthode DbGeography. FromText. Le point géographique représenté en tant que WellKnownText est passé à la méthode. Le code enregistre ensuite les données. Ensuite, la requête LINQ qui retourne un objet University dans lequel son emplacement est le plus proche de l'emplacement spécifié est construite et exécutée.

```
using (var context = new UniversityContext ())
{
    context.Universities.Add(new University()
    {
        Name = "Graphic Design Institute",
        Location = DbGeography.FromText("POINT(-122.336106 47.605049)"),
    });

    context.Universities.Add(new University()
    {
        Name = "School of Fine Art",
        Location = DbGeography.FromText("POINT(-122.335197 47.646711)"),
    });

    context.SaveChanges();

    var myLocation = DbGeography.FromText("POINT(-122.296623 47.640405)");

    var university = (from u in context.Universities
                      orderby u.Location.Distance(myLocation)
                      select u).FirstOrDefault();

    Console.WriteLine(
        "The closest University to you is: {0}.",
        university.Name);
}
```

Compilez et exécutez l'application. Le programme génère la sortie suivante :

```
The closest University to you is: School of Fine Art.
```

## Afficher la base de données générée

Lorsque vous exécutez l'application la première fois, la Entity Framework crée une base de données pour vous. Étant donné que Visual Studio 2012 est installé, la base de données sera créée sur l'instance de base de données locale. Par défaut, le Entity Framework nomme la base de données après le nom qualifié complet du contexte dérivé (dans cet exemple, il s'agit de **SpatialCodeFirst. UniversityContext**). La prochaine fois que la base de données existante sera utilisée.

Notez que si vous apportez des modifications à votre modèle après la création de la base de données, vous devez utiliser Migrations Code First pour mettre à jour le schéma de base de données. Pour obtenir un exemple d'utilisation de migrations, consultez [Code First à une nouvelle base de données](#).

Pour afficher la base de données et les données, procédez comme suit :

1. Dans le menu principal de Visual Studio 2012, sélectionnez **afficher -> Explorateur d'objets SQL Server**.
2. Si la base de données locale ne figure pas dans la liste des serveurs, cliquez sur le bouton droit de la souris sur **SQL Server** et sélectionnez **ajouter SQL Server** utiliser l' **authentification Windows** par défaut pour vous connecter à l'instance de base de données locale.
3. Développez le nœud de base de données locale
4. Dérouler le dossier **bases de données** pour afficher la nouvelle base de données et accéder à la table

## **universités**

5. Pour afficher les données, cliquez avec le bouton droit sur la table et sélectionnez **afficher les données**.

## Résumé

Dans cette procédure pas à pas, nous avons vu comment utiliser des types spatiaux avec Entity Framework Code First.

# Conventions Code First

06/05/2019 • 10 minutes to read

Code First vous permet de décrire un modèle à l'aide de classes C# ou Visual Basic .NET. La forme de base du modèle est détectée à l'aide des conventions. Conventions des ensembles de règles qui sont utilisés pour configurer automatiquement un modèle conceptuel basé sur les définitions de classe lorsque vous travaillez avec Code First. Les conventions sont définies dans l'espace de noms `System.Data.Entity.ModelConfiguration.Conventions`.

Vous pouvez configurer davantage de votre modèle à l'aide des annotations de données ou l'API fluent. La priorité est donnée à la configuration via l'API fluent, suivi de conventions et des annotations de données. Pour plus d'informations, consultez [Annotations de données](#), [API Fluent - relations](#), [API Fluent - Types de & propriétés](#) et [API Fluent avec VB.NET](#).

Une liste détaillée des conventions Code First est disponible dans le [Documentation de l'API](#). Cette rubrique fournit une vue d'ensemble des conventions utilisées par Code First.

## Découverte de type

Lors de l'utilisation du développement Code First vous commencez généralement par écriture de classes .NET Framework qui définissent votre modèle conceptuel (domaine). En plus de définir les classes, vous devez également communiquer à **DbContext** connaître les types que vous souhaitez inclure dans le modèle. Pour ce faire, vous définissez une classe de contexte qui dérive de **DbContext** et expose **DbSet** propriétés pour les types que vous souhaitez faire partie du modèle. Code inclut tout d'abord ces types et également permet d'extraire tous les types référencés, même si les types référencés sont définis dans un autre assembly.

Si vos types de participent à une hiérarchie d'héritage, il suffit de définir un **DbSet** propriété pour la classe de base et les types dérivés qui sera incluse automatiquement, si elles se trouvent dans le même assembly que la classe de base.

Dans l'exemple suivant, il existe un seul **DbSet** propriété définie sur le **SchoolEntities** classe (**départements**). Code utilise cette propriété pour découvrir et d'extraire tous les types référencés.

```

public class SchoolEntities : DbContext
{
    public DbSet<Department> Departments { get; set; }
}

public class Department
{
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; set; }
}

public class Course
{
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
}

public partial class OnlineCourse : Course
{
    public string URL { get; set; }
}

public partial class OnsiteCourse : Course
{
    public string Location { get; set; }
    public string Days { get; set; }
    public System.DateTime Time { get; set; }
}

```

Si vous souhaitez exclure un type à partir du modèle, utilisez le **NotMapped** attribut ou le **DbModelBuilder.Ignore** API fluent.

```
modelBuilder.Ignore<Department>();
```

## Convention de clé primaire

Code déduit tout d'abord qu'une propriété est une clé primaire si une propriété sur une classe est nommée « ID » (non sensible à la casse), ou le nom de classe suivie de « ID ». Si le type de la propriété de clé primaire est numérique ou le GUID sera configuré comme une colonne d'identité.

```

public class Department
{
    // Primary key
    public int DepartmentID { get; set; }

    . . .

}

```

## Convention de relation

Dans Entity Framework, les propriétés de navigation permettent de naviguer d'une relation entre deux types d'entités. Chaque objet peut avoir une propriété de navigation pour chaque relation à laquelle il participe. Propriétés de navigation permettent de parcourir et de gérer les relations dans les deux directions, en retournant un objet de la référence (si la multiplicité est soit un ou zéro-ou-un) ou une collection (si la multiplicité est nombreuses). Code déduit tout d'abord les relations basées sur les propriétés de navigation définies sur vos types.

En plus des propriétés de navigation, nous vous recommandons d'inclure des propriétés de clé étrangère sur les types qui représentent des objets dépendants. N'importe quelle propriété avec le même type de données en tant que propriété de clé primaire principal et avec un nom qui suit l'un des formats suivants représente une clé étrangère pour la relation : «<nom de propriété de navigation><principal nom de propriété de clé primaire>','<nom de la classe principal><nom de propriété de clé primaire>», ou «<nom de propriété de clé primaire principal>». Si plusieurs correspondances sont trouvées priorité est donnée dans l'ordre indiqué ci-dessus. La détection de clé étrangère n'est pas sensible à la casse. Lorsqu'une propriété de clé étrangère est détectée, Code First déduit la multiplicité de la relation en fonction de la possibilité de valeur de la clé étrangère. Si la propriété est nullable, la relation est enregistrée comme étant facultatifs ; Sinon, la relation est inscrit en fonction des besoins.

Si une clé étrangère sur l'entité dépendante n'est pas nullable, Code First définit ensuite suppression en cascade sur la relation. Si une clé étrangère sur l'entité dépendante est nullable, Code First ne définit pas suppression en cascade sur la relation, et lorsque le principal est supprimé la clé étrangère sera définie sur null. La multiplicité et cascade delete comportement détecté par convention peut être substituée à l'aide de l'API fluent.

Dans l'exemple suivant, les propriétés de navigation et une clé étrangère sont utilisés pour définir la relation entre les classes de service et un cours.

```
public class Department
{
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; set; }
}

public class Course
{
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
}
```

#### NOTE

Si vous avez plusieurs relations entre les mêmes types (par exemple, supposons que vous définissiez le **personne** et **livre** classes, où le **personne** classe contient le **ReviewedBooks** et **AuthoredBooks** propriétés de navigation et la **livre** classe contient le **auteur** et **Réviseur** propriétés de navigation) vous devez configurer manuellement les relations à l'aide des Annotations de données ou l'API fluent. Pour plus d'informations, consultez [Annotations de données - relations](#) et [API Fluent - relations](#).

## Convention de Types complexes

Lorsque Code First détecte une définition de classe où une clé primaire ne peut pas être déduite, et aucune clé primaire n'est inscrit par le biais des annotations de données ou l'API fluent, puis le type est automatiquement enregistré comme un type complexe. Détection de type complexe requiert également que le type n'a pas de propriétés qui réfèrent des types d'entités et ne sont pas référencées à partir d'une propriété de collection sur un autre type. Compte tenu des définitions de classe suivant Code First serait déduire que **détails** est un type complexe, car il n'a aucune clé primaire.

```
public partial class OnsiteCourse : Course
{
    public OnsiteCourse()
    {
        Details = new Details();
    }

    public Details Details { get; set; }
}

public class Details
{
    public System.DateTime Time { get; set; }
    public string Location { get; set; }
    public string Days { get; set; }
}
```

## Convention de chaîne de connexion

Pour en savoir plus sur les conventions que DbContext utilise pour détecter la connexion à utiliser voir [connexions et des modèles](#).

## Suppression des Conventions

Vous pouvez supprimer une des conventions définies dans l'espace de noms `System.Data.Entity.ModelConfiguration.Conventions`. L'exemple suivant supprime **PluralizingTableNameConvention**.

```
public class SchoolEntities : DbContext
{
    . .

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure Code First to ignore PluralizingTableName convention
        // If you keep this convention, the generated tables
        // will have pluralized names.
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```

## Conventions personnalisées

Conventions personnalisées sont prises en charge dans EF6 et versions ultérieures. Pour plus d'informations, consultez [Conventions Code First personnalisées](#).

# Conventions personnalisées Code First

13/09/2018 • 21 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Lorsque vous utilisez Code First, votre modèle est calculé à partir de vos classes à l'aide d'un ensemble de conventions. La valeur par défaut [Conventions Code First](#) déterminer des choses comme qui propriété devient la clé primaire d'une entité, le nom de la table d'une entité est mappé à, et que la précision et l'échelle une colonne decimal a par défaut.

Parfois, ces conventions par défaut ne sont pas adaptées à votre modèle, et vous devez y remédier en configurant des nombreuses entités individuelles à l'aide des Annotations de données ou l'API Fluent. Conventions Code First personnalisées vous permettent de définir vos propres conventions qui fournissent des valeurs par défaut de configuration pour votre modèle. Dans cette procédure pas à pas, nous allons examiner les différents types de conventions personnalisées et comment créer chacun d'eux.

## Conventions reposant sur un modèle

Cette page traite de l'API `DbModelBuilder` pour conventions personnalisées. Cette API doit être suffisante pour la création de la plupart des conventions personnalisées. Toutefois, il est également la capacité à créer des conventions basée sur modèle - conventions qui manipulent le modèle final qui a été créé - pour gérer des scénarios avancés. Pour plus d'informations, consultez [Conventions reposant sur le modèle](#).

## Notre modèle

Commençons par définir un modèle simple que nous pouvons utiliser avec nos conventions. Ajoutez les classes suivantes à votre projet.

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

public class ProductContext : DbContext
{
    static ProductContext()
    {
        Database.SetInitializer(new DropCreateDatabaseIfModelChanges<ProductContext>());
    }

    public DbSet<Product> Products { get; set; }
}

public class Product
{
    public int Key { get; set; }
    public string Name { get; set; }
    public decimal? Price { get; set; }
    public DateTime? ReleaseDate { get; set; }
    public ProductCategory Category { get; set; }
}

public class ProductCategory
{
    public int Key { get; set; }
    public string Name { get; set; }
    public List<Product> Products { get; set; }
}

```

## Présentation de Conventions personnalisées

Nous allons écrire une convention qui configure de n'importe quelle propriété de clé qui sera la clé primaire pour son type d'entité nommée.

Conventions sont activées sur le Générateur de modèles, qui est accessible en substituant OnModelCreating dans le contexte. Mettre à jour la classe ProductContext comme suit :

```

public class ProductContext : DbContext
{
    static ProductContext()
    {
        Database.SetInitializer(new DropCreateDatabaseIfModelChanges<ProductContext>());
    }

    public DbSet<Product> Products { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Properties()
            .Where(p => p.Name == "Key")
            .Configure(p => p.IsKey());
    }
}

```

Maintenant, n'importe quelle propriété dans notre modèle nommé clé sera configuré en tant que la clé primaire de toute entité sa partie de.

Nous pourrions également souhaiter que nos conventions plus spécifique en filtrant sur le type de propriété que nous allons configurer :

```

modelBuilder.Properties<int>()
    .Where(p => p.Name == "Key")
    .Configure(p => p.IsKey());

```

Cette opération configure toutes les propriétés appelées clé primaire de clé de l'entité, mais uniquement si elles sont un entier.

Une fonctionnalité intéressante de la méthode IsKey est qu'il est additif. Ce qui signifie que si vous appelez IsKey sur plusieurs propriétés et ils deviennent tous partie d'une clé composite. Pour cela l'inconvénient est que lorsque vous spécifiez plusieurs propriétés pour une clé vous devez également spécifier un ordre de ces propriétés. Pour cela, en appelant la HasColumnOrder méthode comme ci-dessous :

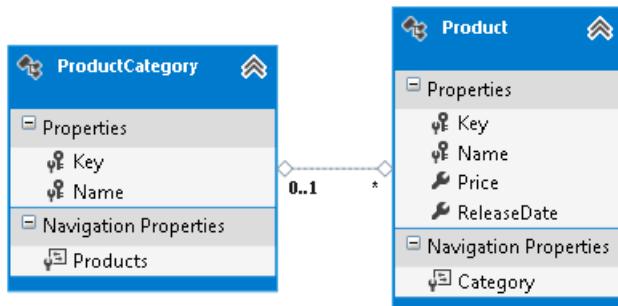
```

modelBuilder.Properties<int>()
    .Where(x => x.Name == "Key")
    .Configure(x => x.IsKey().HasColumnOrder(1));

modelBuilder.Properties()
    .Where(x => x.Name == "Name")
    .Configure(x => x.IsKey().HasColumnOrder(2));

```

Ce code configurera les types dans notre modèle pour avoir une clé composite constituée de la colonne de clé de type int et la colonne de nom de chaîne. Si nous permet d'afficher le modèle dans le concepteur, elle ressemblerait à ceci :



Un autre exemple de conventions de propriété consiste à configurer toutes les propriétés de date/heure dans mon modèle pour mapper au type datetime2 dans SQL Server au lieu de date/heure. Vous pouvez y parvenir avec les éléments suivants :

```

modelBuilder.Properties<DateTime>()
    .Configure(c => c.HasColumnType("datetime2"));

```

## Classes de convention

Une autre façon de définir les conventions consiste à utiliser une Convention de classe pour encapsuler votre convention. Lorsque vous utilisez une classe de Convention puis vous créez un type qui hérite de la classe Convention dans l'espace de noms System.Data.Entity.ModelConfiguration.Conventions.

Nous pouvons créer une classe de Convention avec la convention datetime2 que nous avons montré précédemment en procédant comme suit :

```

public class DateTime2Convention : Convention
{
    public DateTime2Convention()
    {
        this.Properties<DateTime>()
            .Configure(c => c.HasColumnType("datetime2"));
    }
}

```

Pour demander à EF d'utiliser cette convention ajoutez-le à la collection de Conventions dans OnModelCreating, qui si vous avez suivi la procédure pas à pas se présentera comme suit :

```

protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Properties<int>()
        .Where(p => p.Name.EndsWith("Key"))
        .Configure(p => p.IsKey());

    modelBuilder.Conventions.Add(new DateTime2Convention());
}

```

Comme vous pouvez le voir nous ajoutons une instance de notre convention à la collection de conventions. Héritage à partir de la Convention d'offre un moyen pratique de regroupement et de partager des conventions entre équipes ou quelques projets. Vous pouvez, par exemple, avoir une bibliothèque de classes avec un ensemble commun de conventions que toutes vos organisations projets utilisent.

## Attributs personnalisés

Une autre utilisation des conventions consiste à activer les nouveaux attributs à utiliser lors de la configuration d'un modèle. Pour illustrer cela, nous allons créer un attribut que nous pouvons utiliser pour marquer les propriétés de chaîne en tant que non-Unicode.

```

[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
public class NonUnicode : Attribute
{
}

```

Maintenant, nous allons créer une convention pour appliquer cet attribut à notre modèle :

```

modelBuilder.Properties()
    .Where(x => x.GetCustomAttributes(false).OfType<NonUnicode>().Any())
    .Configure(c => cIsUnicode(false));

```

Avec cette convention, nous pouvons ajouter l'attribut de type non-Unicode à un de nos propriétés de chaîne, ce qui signifie que la colonne dans la base de données est stockée en tant que varchar au lieu de nvarchar.

Une chose à noter concernant cette convention est que si vous placez l'attribut de type non-Unicode sur autre chose qu'une propriété de chaîne, puis elle lève une exception. Pour cela, car vous ne pouvez pas configurerIsUnicode sur n'importe quel type autre qu'une chaîne. Si cela se produit, alors vous pouvez rendre votre convention plus spécifiques, afin qu'il exclut tout ce qui n'est pas une chaîne.

Bien que la convention ci-dessus fonctionne pour la définition des attributs personnalisés, il existe une autre API peut être beaucoup plus facile à utiliser, en particulier lorsque vous souhaitez utiliser les propriétés de la classe d'attributs.

Pour cet exemple, nous allons mettre à jour notre attribut et remplacez-le par un attributIsUnicode, afin qu'il

ressemble à ceci :

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false)]
internal class IsUnicode : Attribute
{
    public bool Unicode { get; set; }

    public IsUnicode(bool isUnicode)
    {
        Unicode = isUnicode;
    }
}
```

Une fois que nous avons ceci, nous pouvons définir une valeur booléenne sur notre attribut pour indiquer à la convention ou non une propriété doit être Unicode. Nous pourrions effectuer cette opération dans la convention que nous avons déjà en accédant à la ClrProperty de la classe de configuration comme suit :

```
modelBuilder.Properties()
    .Where(x => x.GetCustomAttributes(false).OfType<IsUnicode>().Any())
    .Configure(c => c.IsUnicode(c.Clr PropertyInfo.GetCustomAttribute<IsUnicode>().Unicode));
```

Cela est assez simple, mais il existe un moyen plus succinct de parvenir à l'aide de la méthode de l'API de conventions. L'avoir de méthode a un paramètre de type Func<PropertyInfo, T> qui accepte le PropertyInfo identique à l'emplacement où la méthode, mais est censée renvoyer un objet. Si l'objet retourné est null, alors la propriété ne sera pas configurée, ce qui signifie que vous pouvez filtrer les propriétés avec lui comme Where, mais il est différent, il sera également capturer l'objet retourné et passez-le à la méthode Configure. Cela fonctionne comme suit :

```
modelBuilder.Properties()
    .Having(x =>x.GetCustomAttributes(false).OfType<IsUnicode>().FirstOrDefault())
    .Configure((config, att) => config.IsUnicode(att.Unicode));
```

Attributs personnalisés ne sont pas la seule raison d'utiliser la méthode, il est utile en tout lieu dont vous avez besoin de raisonner sur quelque chose que vous filtrez sur lors de la configuration de vos types ou les propriétés.

## Configuration des Types

Jusqu'à présent, toutes nos conventions ont été pour les propriétés, mais il existe une autre zone de l'API de conventions pour la configuration des types dans votre modèle. L'expérience est semblable aux conventions que nous avons vus jusqu'à présent, mais les options de configuration à l'intérieur sera à l'entité au lieu de la propriété niveau.

Une des choses que les conventions de niveau de Type peuvent être très utiles pour changer la convention de nommage table pour mapper à un schéma existant qui diffère de la valeur par défaut d'EF ou pour créer une nouvelle base de données avec une autre convention d'affectation de noms. Pour ce faire, nous devons tout d'abord une méthode qui peut accepter la TypeInfo pour un type dans notre modèle et retourner ce que le nom de table pour ce type doit être :

```
private string GetTableName(Type type)
{
    var result = Regex.Replace(type.Name, "[A-Z]", m => m.Value[0] + "_" + m.Value[1]);

    return result.ToLower();
}
```

Cette méthode prend un type et retourne une chaîne qui utilise des minuscules avec des traits de soulignement au lieu de la casse mixte. Dans notre modèle, cela signifie que la classe de ProductCategory sera être mappée à une table appelée produit\_catégorie au lieu de ProductCategories.

Une fois que nous avons cette méthode, nous pouvons l'appeler dans une convention comme suit :

```
modelBuilder.Types()
    .Configure(c => c.ToTable(GetTableName(c.ClrType)));
```

Cette convention configure chaque type dans notre modèle pour mapper au nom de table qui est retourné à partir de notre méthode GetTableName. Cette convention est l'équivalent à l'appel de la méthode ToTable pour chaque entité dans le modèle à l'aide de l'API Fluent.

Une chose à noter à ce sujet est que lorsque vous appelez EF ToTable prendra la chaîne que vous fournissez le nom de table exact, sans les pluralisation qui serait normalement le cas lors de la détermination des noms de table. C'est pourquoi le nom de table à partir de notre convention est produit\_catégorie au lieu de produit\_catégories. Nous pouvons résoudre que dans notre convention en effectuant un appel au service de pluralisation nous-mêmes.

Dans le code suivant, nous utiliserons le [résolution des dépendances](#) fonctionnalité ajoutée dans EF6 pour récupérer le service de pluralisation EF aurait utilisé et pluraliser notre nom de la table.

```
private string GetTableName(Type type)
{
    var pluralizationService = DbConfiguration.DependencyResolver.GetService<IPluralizationService>();

    var result = pluralizationService.Pluralize(type.Name);

    result = Regex.Replace(result, "[A-Z]", m => m.Value[0] + "_" + m.Value[1]);

    return result.ToLower();
}
```

#### NOTE

La version générique de GetService est une méthode d'extension dans l'espace de noms System.Data.Entity.Infrastructure.DependencyResolution, vous devez ajouter une à l'aide de l'instruction à votre contexte pour pouvoir l'utiliser.

#### ToTable et héritage

Un autre aspect important de ToTable est que si vous mappez explicitement un type à une table donnée, vous pouvez modifier la stratégie de mappage qui seront utilisés par EF. Si vous appelez ToTable pour chaque type dans une hiérarchie d'héritage, en passant le nom de type en tant que le nom de la table, comme nous l'avons fait ci-dessus, puis vous allez modifier la stratégie de mappage de Table par hiérarchie (TPH) par défaut pour la Table par Type (TPT). La meilleure façon de décrire Ceci est un exemple concret de whith :

```

public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
}

public class Manager : Employee
{
    public string SectionManaged { get; set; }
}

```

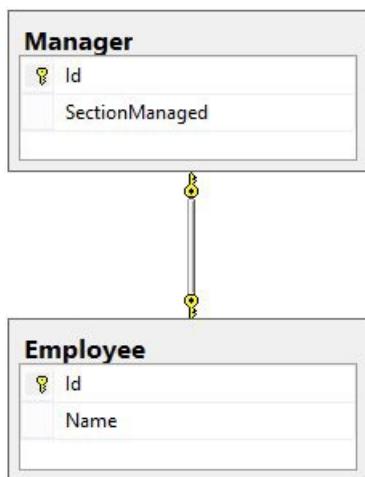
Par défaut employé et manager sont mappés à la même table (employés) dans la base de données. La table contiendra les employés et aux responsables d'une colonne de discriminateur qui vous indiquera à quel type d'instance est stocké dans chaque ligne. Il s'agit de mappage TPH qu'il existe une seule table pour la hiérarchie. Toutefois, si vous appelez ToTable sur les deux classes puis chaque type sera plutôt être mappée à sa propre table, également appelé TPT, car chaque type possède sa propre table.

```

modelBuilder.Types()
    .Configure(c=>c.ToTable(cClrType.Name));

```

Le code ci-dessus mappera vers une structure de table qui ressemble à ceci :



Vous pouvez éviter ce problème et mettre à jour le mappage TPH par défaut, de différentes manières :

1. Appelez ToTable portant le même nom de table pour chaque type dans la hiérarchie.
2. Appelez ToTable uniquement sur la classe de base de la hiérarchie, dans notre exemple serait employé.

## Ordre d'exécution

Conventions de fonctionnement de manière dernière wins, le même que l'API Fluent. Cela signifie que si vous écrivez deux conventions qui configurent la même option de la même propriété, puis il pour exécuter wins. Par exemple, dans le code ci-dessous, la longueur maximale de toutes les chaînes est définie sur 500, mais nous ensuite configurer toutes les propriétés nom dans le modèle pour avoir une longueur maximale de 250.

```

modelBuilder.Properties<string>()
    .Configure(c => c.HasMaxLength(500));

modelBuilder.Properties<string>()
    .Where(x => x.Name == "Name")
    .Configure(c => c.HasMaxLength(250));

```

Étant donné que la convention pour définir la longueur maximale de 250 est après celle qui définit toutes les chaînes à 500, toutes les propriétés appelées nom dans notre modèle aura un MaxLength de 250 lors de toutes les autres chaînes, telles que des descriptions, serait de 500. À l'aide des conventions de cette façon signifie que vous pouvez fournir une convention générale pour les types ou les propriétés dans votre modèle, puis override pour les sous-ensembles sont différents.

L'API Fluent et les Annotations de données peuvent également être utilisées pour remplacer une convention dans des cas spécifiques. Dans l'exemple ci-dessus si nous avions utilisé l'API Fluent pour définir la longueur maximale d'une propriété puis nous pouvons également avoir ajouter il avant ou après la convention, étant donné que l'API Fluent plus spécifiques emporte sur la Convention de Configuration plus générales.

## Conventions intégrées

Étant donné que les conventions personnalisées pourraient être affectées par les conventions Code First par défaut, il peut être utile d'ajouter des conventions à exécuter avant ou après une autre convention. Pour ce faire, vous pouvez utiliser les méthodes AddBefore et AddAfter de la collection de Conventions sur votre DbContext dérivée. Le code suivant ajouteriez la classe convention que nous avons créé précédemment afin qu'elle sera exécutée avant la génération de la convention de découverte des clés.

```
modelBuilder.Conventions.AddBefore<IdKeyDiscoveryConvention>(new DateTime2Convention());
```

Cela va être les plus utiles lors de l'ajout des conventions qui doivent s'exécuter avant ou après les conventions intégrées, une liste des conventions intégrées est disponible ici :

[System.Data.Entity.ModelConfiguration.Conventions Namespace](#) .

Vous pouvez également supprimer les conventions que vous ne souhaitez pas appliquées à votre modèle. Pour supprimer une convention, utilisez la méthode Remove. Voici un exemple de la suppression de la PluralizingTableNameConvention.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
}
```

# Conventions basées sur les modèles

07/11/2019 • 9 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement :** Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Les conventions basées sur les modèles sont une méthode avancée de configuration de modèle basée sur une convention. Pour la plupart des scénarios, l' [API de convention d'code First personnalisée sur DbModelBuilder](#) doit être utilisée. Il est recommandé d'avoir une compréhension de l'API DbModelBuilder pour les conventions avant d'utiliser des conventions basées sur les modèles.

Les conventions basées sur les modèles permettent de créer des conventions qui affectent les propriétés et les tables qui ne sont pas configurables par le biais de conventions standard. Voici des exemples de colonnes de discriminateur dans les modèles de table par hiérarchie et les colonnes d'association indépendantes.

## Création d'une convention

La première étape de la création d'une Convention basée sur un modèle consiste à choisir le moment auquel la Convention doit être appliquée au modèle. Il existe deux types de conventions de modèle, conceptuel (C-Space) et Store (espace S). Une convention d'espace C est appliquée au modèle généré par l'application, tandis qu'une convention d'espacement est appliquée à la version du modèle qui représente la base de données et contrôle les éléments tels que le nom des colonnes générées automatiquement.

Une convention de modèle est une classe qui s'étend à partir de `IConceptualModelConvention` ou `IStoreModelConvention`. Ces interfaces acceptent un type générique qui peut être de type `MetadataItem`, qui est utilisé pour filtrer le type de données auquel la Convention s'applique.

## Ajout d'une convention

Les conventions de modèle sont ajoutées de la même façon que les classes de conventions normales. Dans la méthode `OnModelCreating`, ajoutez la Convention à la liste des conventions pour un modèle.

```
using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;

public class BlogContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Add<MyModelBasedConvention>();
    }
}
```

Une convention peut également être ajoutée par rapport à une autre convention à l'aide de conventions `AddBefore<>` ou `conventions.AddAfter<>` méthodes. Pour plus d'informations sur les conventions que Entity

Framework applique, consultez la section Remarques.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Conventions.AddAfter<IdKeyDiscoveryConvention>(new MyModelBasedConvention());
}
```

## Exemple : Convention de modèle de discriminateur

Le changement de nom des colonnes générées par EF est un exemple de ce que vous ne pouvez pas faire avec les autres API conventions. Il s'agit d'une situation dans laquelle l'utilisation des conventions de modèle est la seule option.

Un exemple d'utilisation d'une Convention basée sur un modèle pour configurer des colonnes générées consiste à personnaliser la manière dont les colonnes de discriminateur sont nommées. Voici un exemple de Convention basée sur un modèle simple qui renomme chaque colonne du modèle nommé « discriminateur » en « EntityType ». Cela comprend les colonnes que le développeur a simplement nommé « discriminateur ». Étant donné que la colonne « discriminateur » est une colonne générée, celle-ci doit s'exécuter dans un espace S.

```
using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;

class DiscriminatorRenamingConvention : IStoreModelConvention<EdmProperty>
{
    public void Apply(EdmProperty property, DbModel model)
    {
        if (property.Name == "Discriminator")
        {
            property.Name = "EntityType";
        }
    }
}
```

## Exemple : Convention de renommage générale d'IA

Un autre exemple plus complexe de conventions basées sur les modèles en action consiste à configurer la façon dont les associations indépendantes (IAs) sont nommées. Il s'agit d'une situation dans laquelle les conventions de modèle sont applicables, car IAs est généré par EF et n'est pas présent dans le modèle auquel l'API DbModelBuilder peut accéder.

Quand EF génère une IA, il crée une colonne nommée EntityType\_KeyName. Par exemple, pour une association nommée Customer avec une colonne clé nommée CustomerId, elle générera une colonne nommée Customer\_CustomerId. La Convention suivante supprime le caractère «\_» du nom de colonne généré pour l'IA.

```

using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;

// Provides a convention for fixing the independent association (IA) foreign key column names.
public class ForeignKeyNamingConvention : IStoreModelConvention<AssociationType>
{
    public void Apply(AssociationType association, DbModel model)
    {
        // Identify ForeignKey properties (including IAs)
        if (association.IsForeignKey)
        {
            // rename FK columns
            var constraint = association.Constraint;
            if (DoPropertiesHaveDefaultNames(constraint.FromProperties, constraint.ToRole.Name,
                constraint.ToProperties))
            {
                NormalizeForeignKeyProperties(constraint.FromProperties);
            }
            if (DoPropertiesHaveDefaultNames(constraint.ToProperties, constraint.FromRole.Name,
                constraint.FromProperties))
            {
                NormalizeForeignKeyProperties(constraint.ToProperties);
            }
        }
    }

    private bool DoPropertiesHaveDefaultNames(ReadOnlyMetadataCollection<EdmProperty> properties, string
        roleName, ReadOnlyMetadataCollection<EdmProperty> otherEndProperties)
    {
        if (properties.Count != otherEndProperties.Count)
        {
            return false;
        }

        for (int i = 0; i < properties.Count; ++i)
        {
            if (!properties[i].Name.EndsWith("_" + otherEndProperties[i].Name))
            {
                return false;
            }
        }
        return true;
    }

    private void NormalizeForeignKeyProperties(ReadOnlyMetadataCollection<EdmProperty> properties)
    {
        for (int i = 0; i < properties.Count; ++i)
        {
            int underscoreIndex = properties[i].Name.IndexOf('_');
            if (underscoreIndex > 0)
            {
                properties[i].Name = properties[i].Name.Remove(underscoreIndex, 1);
            }
        }
    }
}

```

## Extension des conventions existantes

Si vous devez écrire une convention qui est semblable à l'une des conventions que Entity Framework applique déjà à votre modèle, vous pouvez toujours étendre cette Convention pour éviter de devoir la réécrire à partir de zéro. Par exemple, vous pouvez remplacer la Convention de correspondance d'ID existante par une convention

personnalisée. L'un des avantages supplémentaires de la substitution de la Convention de clé est que la méthode substituée sera appelée uniquement si aucune clé n'a été détectée ou explicitement configurée. La liste des conventions utilisées par Entity Framework est disponible ici :

<http://msdn.microsoft.com/library/system.data.entity.modelconfiguration.conventions.aspx>.

```
using System.Data.Entity;
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Linq;

// Convention to detect primary key properties.
// Recognized naming patterns in order of precedence are:
// 1. 'Key'
// 2. [type name]Key
// Primary key detection is case insensitive.
public class CustomKeyDiscoveryConvention : KeyDiscoveryConvention
{
    private const string Id = "Key";

    protected override IEnumerable<EdmProperty> MatchKeyProperty(
        EntityType entityType, IEnumerable<EdmProperty> primitiveProperties)
    {
        Debug.Assert(entityType != null);
        Debug.Assert(primitiveProperties != null);

        var matches = primitiveProperties
            .Where(p => Id.Equals(p.Name, StringComparison.OrdinalIgnoreCase));

        if (!matches.Any())
        {
            matches = primitiveProperties
                .Where(p => (entityType.Name + Id).Equals(p.Name, StringComparison.OrdinalIgnoreCase));
        }

        // If the number of matches is more than one, then multiple properties matched differing only by
        // case--for example, "Key" and "key".
        if (matches.Count() > 1)
        {
            throw new InvalidOperationException("Multiple properties match the key convention");
        }

        return matches;
    }
}
```

Nous devons ensuite ajouter la nouvelle convention avant la Convention de clé existante. Après avoir ajouté le CustomKeyDiscoveryConvention, nous pouvons supprimer le IdKeyDiscoveryConvention. Si nous n'avons pas supprimé le IdKeyDiscoveryConvention existant, cette Convention aura toujours priorité sur la Convention de détection des ID, car elle est exécutée en premier, mais dans le cas où aucune propriété « clé » n'est trouvée, la Convention « ID » s'exécute. Nous voyons ce comportement, car chaque convention voit le modèle tel qu'il a été mis à jour par la convention précédente (au lieu de s'en servir indépendamment et tout est combiné) de sorte que, si, par exemple, une convention précédente a mis à jour un nom de colonne pour correspondre à un nom de colonne intérêt pour votre convention personnalisée (lorsque le nom ne vous intéresse pas), elle s'applique à cette colonne.

```
public class BlogContext : DbContext
{
    public DbSet<Post> Posts { get; set; }
    public DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.AddBefore<IdKeyDiscoveryConvention>(new CustomKeyDiscoveryConvention());
        modelBuilder.Conventions.Remove<IdKeyDiscoveryConvention>();
    }
}
```

## Notes

Une liste de conventions actuellement appliquées par Entity Framework est disponible dans la documentation MSDN ici : <http://msdn.microsoft.com/library/system.data.entity.modelconfiguration.conventions.aspx>. Cette liste est extraite directement à partir de notre code source. Le code source de Entity Framework 6 est disponible sur [GitHub](#) et la plupart des conventions utilisées par Entity Framework sont de bons points de départ pour les conventions basées sur les modèles personnalisés.

# API Fluent - relations

13/09/2018 • 11 minutes to read

## NOTE

Cette page fournit des informations sur la définition des relations dans votre modèle Code First à l'aide de l'API fluent. Pour obtenir des informations générales sur les relations dans Entity Framework et comment accéder à et manipuler des données à l'aide de relations, consultez [relations & Propriétés de Navigation](#).

Lorsque vous travaillez avec Code First, vous définissez votre modèle en définissant vos classes CLR de domaine. Par défaut, Entity Framework utilise les conventions Code First pour mapper vos classes pour le schéma de base de données. Si vous utilisez les conventions d'affectation de noms de Code First, dans la plupart des cas, vous pouvez compter sur Code First pour définir des relations entre vos tables basées sur les clés étrangères et les propriétés de navigation que vous définissez sur les classes. Si vous ne suivez pas les conventions lors de la définition de vos classes, ou si vous souhaitez modifier la façon dont les conventions de fonctionnement, vous pouvez utiliser l'API fluent ou des annotations de données pour configurer vos classes de Code First pour mapper les relations entre vos tables.

## Introduction

Lorsque vous configurez une relation avec l'API fluent, vous commencez par l'instance `EntityTypeConfiguration`, puis utilisez la méthode `HasRequired`, `HasOptional` ou `HasMany` pour spécifier le type de relation de que cette entité participe. Les méthodes `HasRequired` et `HasOptional` prennent une expression lambda qui représente une propriété de navigation de référence. La méthode `HasMany` prend une expression lambda qui représente une propriété de navigation de collection. Vous pouvez ensuite configurer une propriété de navigation inverse en utilisant les méthodes `WithRequired`, `WithOptional` et `WithMany`. Ces méthodes ont des surcharges qui ne prennent pas d'arguments et peuvent être utilisées pour spécifier cardinalité avec navigation unidirectionnelle.

Vous pouvez ensuite configurer les propriétés de clé étrangère à l'aide de la méthode `HasForeignKey`. Cette méthode prend une expression lambda qui représente la propriété à utiliser comme clé étrangère.

## Configuration d'une relation requise-à-facultatif (un-à-zéro-ou-un)

L'exemple suivant configure une relation un-à-zéro-ou-un. Le `OfficeAssignment` a la propriété `InstructorID` qui est une clé primaire et une clé étrangère, étant donné que le nom de la propriété ne respecte pas la convention, que la méthode `HasKey` permet de configurer la clé primaire.

```
// Configure the primary key for the OfficeAssignment
modelBuilder.Entity<OfficeAssignment>()
    .HasKey(t => t.InstructorID);

// Map one-to-zero or one relationship
modelBuilder.Entity<OfficeAssignment>()
    .HasRequired(t => t.Instructor)
    .WithOptional(t => t.OfficeAssignment);
```

## Configuration d'une relation où les deux extrémités sont nécessaires (-à-un)

Dans la plupart des cas Entity Framework peut déduire le type dépend et qui est le principal dans une relation. Toutefois, lorsque les deux extrémités de la relation sont requises ou les deux côtés sont facultatifs Entity Framework ne peut pas identifier le principal et dépendants. Lorsque les deux extrémités de la relation sont requises, utilisez WithRequiredPrincipal ou WithRequiredDependent après la méthode HasRequired. Lorsque les deux extrémités de la relation sont facultatifs, utilisez WithOptionalPrincipal ou WithOptionalDependent après la méthode HasOptional.

```
// Configure the primary key for the OfficeAssignment
modelBuilder.Entity<OfficeAssignment>()
    .HasKey(t => t.InstructorID);

modelBuilder.Entity<Instructor>()
    .HasRequired(t => t.OfficeAssignment)
    .WithRequiredPrincipal(t => t.Instructor);
```

## Configuration d'une relation plusieurs-à-plusieurs

Le code suivant configure une relation plusieurs-à-plusieurs entre les types de Course et Instructor. Dans l'exemple suivant, les conventions Code First par défaut sont utilisées pour créer une table de jointure. Par conséquent, la table CourseInstructor est créée avec des colonnes Course\_CourseID et Instructor\_InstructorID.

```
modelBuilder.Entity<Course>()
    .HasMany(t => t.Instructors)
    .WithMany(t => t.Courses)
```

Si vous souhaitez spécifier le nom de table de jointure et les noms des colonnes de la table, vous devez effectuer une configuration supplémentaire à l'aide de la méthode de mappage. Le code suivant génère la table CourseInstructor avec des colonnes CourseID et InstructorID.

```
modelBuilder.Entity<Course>()
    .HasMany(t => t.Instructors)
    .WithMany(t => t.Courses)
    .Map(m =>
{
    m.ToTable("CourseInstructor");
    m.MapLeftKey("CourseID");
    m.MapRightKey("InstructorID");
});
```

## Configuration d'une relation avec une propriété de Navigation

Un unidirectionnelle (également appelé unidirectionnel) relation est lorsqu'une propriété de navigation est définie sur un seul des extrémités de relation et non sur les deux. Par convention, Code First interprète toujours une relation unidirectionnelle comme un-à-plusieurs. Par exemple, si vous souhaitez une relation un à un entre formateur et OfficeAssignment, où vous avez une propriété de navigation sur uniquement le type Instructor, vous devez utiliser l'API fluent pour configurer cette relation.

```
// Configure the primary Key for the OfficeAssignment
modelBuilder.Entity<OfficeAssignment>()
    .HasKey(t => t.InstructorID);

modelBuilder.Entity<Instructor>()
    .HasRequired(t => t.OfficeAssignment)
    .WithRequiredPrincipal();
```

## L'activation de suppression en Cascade

Vous pouvez configurer la suppression en cascade sur une relation à l'aide de la méthode `WillCascadeOnDelete`. Si une clé étrangère sur l'entité dépendante n'est pas nullable, Code First définit ensuite suppression en cascade sur la relation. Si une clé étrangère sur l'entité dépendante est nullable, Code First ne définit pas suppression en cascade sur la relation, et lorsque le principal est supprimé la clé étrangère sera définie sur null.

Vous pouvez supprimer ces conventions de suppression en cascade à l'aide de :

```
modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>()
modelBuilder.Conventions.Remove<ManyToManyCascadeDeleteConvention>()
```

Le code suivant configure la relation comme étant obligatoire et puis désactive la suppression en cascade.

```
modelBuilder.Entity<Course>()
    .IsRequired(t => t.Department)
    .WithMany(t => t.Courses)
    .HasForeignKey(d => d.DepartmentID)
    .WillCascadeOnDelete(false);
```

## Configuration d'une clé étrangère Composite

Si la clé primaire sur le type de service est constitué de propriétés `DepartmentID` et `Name`, vous devez configurer la clé primaire pour le service et la clé étrangère sur les types de cours comme suit :

```
// Composite primary key
modelBuilder.Entity<Department>()
    .HasKey(d => new { d.DepartmentID, d.Name });

// Composite foreign key
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(d => d.Courses)
    .HasForeignKey(d => new { d.DepartmentID, d.DepartmentName });
```

## Modification du nom d'une clé étrangère qui n'est pas définie dans le modèle

Si vous choisissez de ne pas définir une clé étrangère sur le type CLR, mais souhaitez spécifier quel nom, il doit avoir dans la base de données, procédez comme suit :

```
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(t => t.Courses)
    .Map(m => m.MapKey("ChangedDepartmentID"));
```

## Configuration d'un nom de clé étrangère qui ne respecte pas la Convention de premier Code

Si la propriété de clé étrangère sur la classe de cours était appelée `SomeDepartmentID` au lieu de `DepartmentID`, vous devez procédez comme suit pour spécifier que vous souhaitez `SomeDepartmentID` qui sera la clé étrangère :

```
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(d => d.Courses)
    .HasForeignKey(c => c.SomeDepartmentID);
```

## Modèle utilisé dans les exemples

Le modèle Code First suivant est utilisé pour les exemples de cette page.

```
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
// add a reference to System.ComponentModel.DataAnnotations DLL
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
using System;

public class SchoolEntities : DbContext
{
    public DbSet<Course> Courses { get; set; }
    public DbSet<Department> Departments { get; set; }
    public DbSet<Instructor> Instructors { get; set; }
    public DbSet<OfficeAssignment> OfficeAssignments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure Code First to ignore PluralizingTableName convention
        // If you keep this convention then the generated tables will have pluralized names.
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}

public class Department
{
    public Department()
    {
        this.Courses = new HashSet<Course>();
    }
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }
    public decimal Budget { get; set; }
    public System.DateTime StartDate { get; set; }
    public int? Administrator { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; private set; }
}

public class Course
{
    public Course()
    {
        this.Instructors = new HashSet<Instructor>();
    }
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
```

```
    public virtual ICollection<Instructor> Instructors { get; private set; }

}

public partial class OnlineCourse : Course
{
    public string URL { get; set; }
}

public partial class OnsiteCourse : Course
{
    public OnsiteCourse()
    {
        Details = new Details();
    }

    public Details Details { get; set; }
}

public class Details
{
    public System.DateTime Time { get; set; }
    public string Location { get; set; }
    public string Days { get; set; }
}

public class Instructor
{
    public Instructor()
    {
        this.Courses = new List<Course>();
    }

    // Primary key
    public int InstructorID { get; set; }
    public string LastName { get; set; }
    public string FirstName { get; set; }
    public System.DateTime HireDate { get; set; }

    // Navigation properties
    public virtual ICollection<Course> Courses { get; private set; }
}

public class OfficeAssignment
{
    // Specifying InstructorID as a primary
    [Key()]
    public Int32 InstructorID { get; set; }

    public string Location { get; set; }

    // When Entity Framework sees Timestamp attribute
    // it configures ConcurrencyCheck and DatabaseGeneratedPattern=Computed.
    [Timestamp]
    public Byte[] Timestamp { get; set; }

    // Navigation property
    public virtual Instructor Instructor { get; set; }
}
```

# API Fluent - configuration et de mappage des Types et des propriétés

28/11/2018 • 21 minutes to read

Lorsque vous travaillez avec Entity Framework Code First le comportement par défaut consiste à mapper vos classes POCO à des tables à l'aide d'un ensemble de conventions intégrées à EF. Parfois, cependant, vous ne pouvez pas ou ne souhaitez pas suivre ces conventions et devez mapper des entités sur autre chose que ce que les conventions dictent.

Il existe deux façons principales, vous pouvez configurer EF pour utiliser une valeur autre que de conventions, à savoir [annotations](#) ou l'API fluent EFs. Les annotations couvrent uniquement un sous-ensemble des fonctionnalités de l'API fluent, donc il existe des scénarios de mappage qui ne peuvent pas être obtenus à l'aide d'annotations. Cet article est conçu pour montrer comment utiliser l'API fluent pour configurer les propriétés.

L'API fluent code first est généralement accessible en substituant la méthode [OnModelCreating](#) sur votre dérivée [DbContext](#). Les exemples suivants sont conçus pour montrer comment effectuer diverses tâches avec l'api fluent et vous permettent de copier le code et le personnaliser pour l'adapter à votre modèle, si vous souhaitez voir le modèle qui ils peuvent être utilisés avec comme-est alors il est fourni à la fin de cet article.

## Paramètres de modèle à l'échelle

### Schéma par défaut (Entity Framework 6 et versions ultérieures)

En commençant par EF6, vous pouvez utiliser la méthode `HasDefaultSchema` sur `DbModelBuilder` pour spécifier le schéma de base de données à utiliser pour toutes les tables, procédures stockées, etc. Ce paramètre par défaut est remplacé pour tous les objets que vous configurez explicitement un schéma différent pour.

```
modelBuilder.HasDefaultSchema("sales");
```

### Conventions personnalisées (Entity Framework 6 et versions ultérieures)

Démarrage avec EF6, vous pouvez créer vos propres conventions en supplément de ceux inclus dans le Code First. Pour plus d'informations, consultez [Conventions Code First personnalisées](#).

## Mappage de propriétés

Le [propriété](#) méthode est utilisée pour configurer des attributs pour chaque propriété appartenant à une entité ou un type complexe. La méthode de propriété est utilisée pour obtenir un objet de configuration pour une propriété donnée. Les options de l'objet de configuration sont spécifiques au type en cours de configuration ; `IsUnicode` est disponible uniquement sur les propriétés de chaîne par exemple.

### Configuration d'une clé primaire

La convention d'Entity Framework pour les clés primaires est :

1. Votre classe définit une propriété dont le nom est « ID » ou « Id »
2. ou un nom de classe suivie de « ID » ou « Id »

Pour définir explicitement une propriété soit une clé primaire, vous pouvez utiliser la méthode `HasKey`. Dans l'exemple suivant, la méthode `HasKey` permet de configurer la clé primaire `InstructorID` sur le type `OfficeAssignment`.

```
modelBuilder.Entity<OfficeAssignment>().HasKey(t => t.InstructorID);
```

## Configuration d'une clé primaire Composite

L'exemple suivant configure les propriétés DepartmentID et le nom à la clé primaire composite du type de service.

```
modelBuilder.Entity<Department>().HasKey(t => new { t.DepartmentID, t.Name });
```

## En désactivant les identités pour les clés primaires numériques

L'exemple suivant définit la propriété DepartmentID à

System.ComponentModel.DataAnnotations.DatabaseGeneratedOption.None pour indiquer que la valeur ne sera pas générée par la base de données.

```
modelBuilder.Entity<Department>().Property(t => t.DepartmentID)
    .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);
```

## Spécifiant la longueur maximale sur une propriété

Dans l'exemple suivant, la propriété Name doit être non plus de 50 caractères. Si vous apportez à la valeur de plus de 50 caractères, vous obtiendrez un [DbEntityValidationException](#) exception. Si le Code First crée une base de données à partir de ce modèle il définit également la longueur maximale de la colonne de nom à 50 caractères.

```
modelBuilder.Entity<Department>().Property(t => t.Name).HasMaxLength(50);
```

## Configuration de la propriété comme étant obligatoire

Dans l'exemple suivant, la propriété Name est requise. Si vous ne spécifiez pas le nom, vous obtiendrez une exception [DbEntityValidationException](#). Si le Code First crée une base de données à partir de ce modèle puis la colonne utilisée pour stocker cette propriété seront généralement non nullable.

### NOTE

Dans certains cas il ne peut pas être possible pour la colonne dans la base de données soit non nullable même si la propriété est requise. Par exemple, quand à l'aide de données de stratégie de l'héritage TPH pour plusieurs types est stockée dans une table unique. Si un type dérivé inclut une propriété obligatoire, que la colonne ne peut pas être rendue non nullable comme pas tous les types dans la hiérarchie ont cette propriété.

```
modelBuilder.Entity<Department>().Property(t => t.Name).IsRequired();
```

## Configuration d'un Index sur une ou plusieurs propriétés

### NOTE

**EF6.1 et versions ultérieures uniquement** -attribut de l'Index a été introduite dans Entity Framework 6.1. Si vous utilisez une version antérieure les informations contenues dans cette section ne s'applique pas.

Création d'index n'est pas pris en charge en mode natif par l'API Fluent, mais vous pouvez faire utiliser la prise en charge pour **IndexAttribute** via l'API Fluent. Attributs d'index sont traités en incluant une annotation de modèle sur le modèle qui est ensuite activée dans un Index dans la base de données plus loin dans le pipeline. Vous pouvez ajouter manuellement ces mêmes annotations à l'aide de l'API Fluent.

Le moyen le plus simple pour ce faire consiste à créer une instance de **IndexAttribute** qui contient tous les paramètres pour le nouvel index. Vous pouvez ensuite créer une instance de **IndexAnnotation** qui est un type spécifique d'EF qui convertira le **IndexAttribute** paramètres dans une annotation de modèle qui peut être stocké sur le modèle EF. Il peuvent ensuite être transmis à la **HasColumnAnnotation** méthode sur l'API Fluent, en spécifiant le nom **Index** pour l'annotation.

```
modelBuilder
    .Entity<Department>()
    .Property(t => t.Name)
    .HasColumnAnnotation("Index", new IndexAnnotation(new IndexAttribute()));
```

Pour obtenir la liste complète des paramètres disponibles dans **IndexAttribute**, consultez le *Index* section de [Annotations de données Code First](#). Cela inclut la personnalisation du nom de l'index, la création d'index uniques et la création des index multi-colonnes.

Vous pouvez spécifier plusieurs annotations d'index sur une propriété unique en passant un tableau de **IndexAttribute** au constructeur de **IndexAnnotation**.

```
modelBuilder
    .Entity<Department>()
    .Property(t => t.Name)
    .HasColumnAnnotation(
        "Index",
        new IndexAnnotation(new[]
        {
            new IndexAttribute("Index1"),
            new IndexAttribute("Index2") { IsUnique = true }
        }));
    ));;
```

### Spécification afin de ne pas mapper une propriété CLR à une colonne dans la base de données

L'exemple suivant montre comment spécifier qu'une propriété sur un type CLR n'est pas mappée à une colonne dans la base de données.

```
modelBuilder.Entity<Department>().Ignore(t => t.Budget);
```

### Mappage d'une propriété CLR à une colonne spécifique dans la base de données

L'exemple suivant mappe la propriété nom CLR à la colonne de base de données DepartmentName.

```
modelBuilder.Entity<Department>()
    .Property(t => t.Name)
    .HasColumnName("DepartmentName");
```

### Modification du nom d'une clé étrangère qui n'est pas définie dans le modèle

Si vous choisissez de ne pas définir une clé étrangère sur un type CLR, mais souhaitez spécifier quel nom, il doit avoir dans la base de données, procédez comme suit :

```
modelBuilder.Entity<Course>()
    .IsRequired(c => c.Department)
    .WithMany(t => t.Courses)
    .Map(m => m.MapKey("ChangedDepartmentID"));
```

### Configuration si une propriété de chaîne prend en charge le contenu Unicode

Par défaut, les chaînes sont Unicode (nvarchar dans SQL Server). Vous pouvez utiliser la méthode `IsUnicode` pour

spécifier qu'une chaîne doit être de type varchar.

```
modelBuilder.Entity<Department>()
    .Property(t => t.Name)
    .IsUnicode(false);
```

### Configuration du Type de données d'une colonne de base de données

Le [HasColumnType](#) méthode active le mappage aux différentes représentations du même type de base. À l'aide de cette méthode ne vous autorise pas à effectuer une conversion des données en cours d'exécution. Notez que `IsUnicode` est la meilleure façon de définition de colonnes varchar, car il est indépendant de la base de données.

```
modelBuilder.Entity<Department>()
    .Property(p => p.Name)
    .HasColumnType("varchar");
```

### Configurer les propriétés d'un Type complexe

Il existe deux façons de configurer des propriétés scalaires sur un type complexe.

Vous pouvez appeler la propriété sur `ComplexTypeConfiguration`.

```
modelBuilder.ComplexType<Details>()
    .Property(t => t.Location)
    .HasMaxLength(20);
```

Vous pouvez également utiliser la notation par points pour accéder à une propriété d'un type complexe.

```
modelBuilder.Entity<OnsiteCourse>()
    .Property(t => t.Details.Location)
    .HasMaxLength(20);
```

### Configuration d'une propriété à utiliser comme un jeton d'accès concurrentiel optimiste

Pour spécifier qu'une propriété dans une entité représente un jeton d'accès concurrentiel, vous pouvez utiliser l'attribut `ConcurrencyCheck` ou la méthode `IsConcurrencyToken`.

```
modelBuilder.Entity<OfficeAssignment>()
    .Property(t => t.Timestamp)
    .IsConcurrencyToken();
```

Vous pouvez également utiliser la méthode `IsRowVersion` pour configurer la propriété à une version de ligne dans la base de données. Définition de la propriété soit qu'une version de ligne configure automatiquement pour qu'il soit un jeton d'accès concurrentiel optimiste.

```
modelBuilder.Entity<OfficeAssignment>()
    .Property(t => t.Timestamp)
    .IsRowVersion();
```

## Mappage de type

### En spécifiant qu'une classe est un Type complexe

Par convention, un type qui ne possède aucune clé primaire spécifiée est traité comme un type complexe. Il existe certains scénarios où `Code First` ne détecte pas un type complexe (par exemple, si vous n'avez pas une propriété

appelée ID, mais vous ne signifient pas qu'elle soit une clé primaire). Dans ce cas, vous utiliseriez l'API fluent pour spécifier explicitement qu'un type est un type complexe.

```
modelBuilder.ComplexType<Details>();
```

### Spécification afin de ne pas mapper un Type d'entité CLR à une Table dans la base de données

L'exemple suivant montre comment exclure un type CLR de qui est mappé à une table dans la base de données.

```
modelBuilder.Ignore<OnlineCourse>();
```

### Mappage d'un Type d'entité à une Table spécifique dans la base de données

Toutes les propriétés du service seront mappées aux colonnes dans une table appelée m\_ département.

```
modelBuilder.Entity<Department>()
    .ToTable("t_Department");
```

Vous pouvez également spécifier le nom de schéma comme suit :

```
modelBuilder.Entity<Department>()
    .ToTable("t_Department", "school");
```

### Mappage de l'héritage Table par hiérarchie (TPH)

Dans le scénario de mappage TPH, tous les types dans une hiérarchie d'héritage sont mappés à une seule table. Une colonne de discriminateur est utilisée pour identifier le type de chaque ligne. Lorsque vous créez votre modèle avec Code First, TPH est la stratégie par défaut pour les types qui participent à la hiérarchie d'héritage. Par défaut, la colonne de discriminateur est ajoutée à la table avec le nom « Discriminator » et le nom de type CLR de chaque type dans la hiérarchie est utilisé pour les valeurs de discriminateur. Vous pouvez modifier le comportement par défaut à l'aide de l'API fluent.

```
modelBuilder.Entity<Course>()
    .Map<Course>(m => m.Requires("Type").HasValue("Course"))
    .Map<OnsiteCourse>(m => m.Requires("Type").HasValue("OnsiteCourse"));
```

### Mappage de l'héritage Table par Type (TPT)

Dans le scénario de mappage TPT, tous les types sont mappés à des tables individuelles. Les propriétés qui appartiennent uniquement à un type de base ou à un type dérivé sont stockées dans une table qui mappe à ce type. Les tables qui mappent aux types dérivés stockent également une clé étrangère qui joint la table dérivée avec la table de base.

```
modelBuilder.Entity<Course>().ToTable("Course");
modelBuilder.Entity<OnsiteCourse>().ToTable("OnsiteCourse");
```

### Mappage de l'héritage Table-par classe concrète (TPC)

Dans le scénario de mappage TPC, tous les types non abstraits dans la hiérarchie sont mappés à des tables individuelles. Les tables qui mappent aux classes dérivées n'ont aucune relation à la table qui mappe à la classe de base dans la base de données. Toutes les propriétés d'une classe, y compris les propriétés héritées, sont mappées aux colonnes de la table correspondante.

Appelez la méthode MapInheritedProperties pour configurer chaque type dérivé. MapInheritedProperties remappe toutes les propriétés héritées de la classe de base à de nouvelles colonnes dans la table pour la classe

dérivée.

#### NOTE

Notez que, étant donné que les tables impliquées dans la hiérarchie d'héritage TPC ne partagent pas une clé primaire il sera clés d'entité en double, lors de l'insertion dans des tables qui sont mappés aux sous-classes si vous avez des valeurs de base de données générée avec la même valeur de départ d'identité. Pour résoudre ce problème, vous pouvez spécifier une valeur de départ initiale différente pour chaque table ou désactive l'identité sur la propriété de clé primaire. L'identité est la valeur par défaut pour les propriétés de clé entière lorsque vous travaillez avec Code First.

```
modelBuilder.Entity<Course>()
    .Property(c => c.CourseID)
    .HasDatabaseGeneratedOption(DatabaseGeneratedOption.None);

modelBuilder.Entity<OnsiteCourse>().Map(m =>
{
    m.MapInheritedProperties();
    m.ToTable("OnsiteCourse");
});

modelBuilder.Entity<OnlineCourse>().Map(m =>
{
    m.MapInheritedProperties();
    m.ToTable("OnlineCourse");
});
```

#### Mappage des propriétés d'un Type d'entité à plusieurs Tables dans la base de données (fractionnement d'entité)

Fractionnement d'entité permet aux propriétés d'un type d'entité d'être réparties entre plusieurs tables. Dans l'exemple suivant, l'entité Department est divisée en deux tables : département et DepartmentDetails.

Fractionnement d'entité utilise plusieurs appels à la méthode de mappage pour mapper un sous-ensemble de propriétés à une table spécifique.

```
modelBuilder.Entity<Department>()
    .Map(m =>
{
    m.Properties(t => new { t.DepartmentID, t.Name });
    m.ToTable("Department");
})
    .Map(m =>
{
    m.Properties(t => new { t.DepartmentID, t.Administrator, t.StartDate, t.Budget });
    m.ToTable("DepartmentDetails");
});
```

#### Mappage de plusieurs Types d'entités à une Table dans la base de données (fractionnement de Table)

L'exemple suivant mappe les deux types d'entités qui partagent une clé primaire pour une table.

```

modelBuilder.Entity<OfficeAssignment>()
    .HasKey(t => t.InstructorID);

modelBuilder.Entity<Instructor>()
    .IsRequired(t => t.OfficeAssignment)
    .WithRequiredPrincipal(t => t.Instructor);

modelBuilder.Entity<Instructor>().ToTable("Instructor");

modelBuilder.Entity<OfficeAssignment>().ToTable("Instructor");

```

## Mappage d'un Type d'entité aux procédures stockées Insert/Update/Delete (Entity Framework 6 et versions ultérieures)

En commençant par EF6, vous pouvez mapper une entité pour utiliser des procédures stockées pour insérer, mettre à jour et supprimer. Pour plus d'informations, consultez [Code première Insert/Update/Delete Stored Procedures](#).

## Modèle utilisé dans les exemples

Le modèle Code First suivant est utilisé pour les exemples de cette page.

```

using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
// add a reference to System.ComponentModel.DataAnnotations DLL
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
using System;

public class SchoolEntities : DbContext
{
    public DbSet<Course> Courses { get; set; }
    public DbSet<Department> Departments { get; set; }
    public DbSet<Instructor> Instructors { get; set; }
    public DbSet<OfficeAssignment> OfficeAssignments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        // Configure Code First to ignore PluralizingTableName convention
        // If you keep this convention then the generated tables will have pluralized names.
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}

public class Department
{
    public Department()
    {
        this.Courses = new HashSet<Course>();
    }
    // Primary key
    public int DepartmentID { get; set; }
    public string Name { get; set; }
    public decimal Budget { get; set; }
    public System.DateTime StartDate { get; set; }
    public int? Administrator { get; set; }

    // Navigation property
    public virtual ICollection<Course> Courses { get; private set; }
}

public class Course
{
    public Course()
    {

```

```

        this.Instructors = new HashSet<Instructor>();
    }
    // Primary key
    public int CourseID { get; set; }

    public string Title { get; set; }
    public int Credits { get; set; }

    // Foreign key
    public int DepartmentID { get; set; }

    // Navigation properties
    public virtual Department Department { get; set; }
    public virtual ICollection<Instructor> Instructors { get; private set; }
}

public partial class OnlineCourse : Course
{
    public string URL { get; set; }
}

public partial class OnsiteCourse : Course
{
    public OnsiteCourse()
    {
        Details = new Details();
    }

    public Details Details { get; set; }
}

public class Details
{
    public System.DateTime Time { get; set; }
    public string Location { get; set; }
    public string Days { get; set; }
}

public class Instructor
{
    public Instructor()
    {
        this.Courses = new List<Course>();
    }

    // Primary key
    public int InstructorID { get; set; }
    public string LastName { get; set; }
    public string FirstName { get; set; }
    public System.DateTime HireDate { get; set; }

    // Navigation properties
    public virtual ICollection<Course> Courses { get; private set; }
}

public class OfficeAssignment
{
    // Specifying InstructorID as a primary
    [Key()]
    public Int32 InstructorID { get; set; }

    public string Location { get; set; }

    // When Entity Framework sees Timestamp attribute
    // it configures ConcurrencyCheck and DatabaseGeneratedPattern=Computed.
    [Timestamp]
    public Byte[] Timestamp { get; set; }

    // Navigation properties
}

```

```
// Navigation property  
public virtual Instructor Instructor { get; set; }  
}
```

# API Fluent avec VB.NET

05/12/2019 • 11 minutes to read

Code First vous permet de définir votre modèle à l'aide de classes C# ou VB.NET. Une configuration supplémentaire peut éventuellement être effectuée à l'aide des attributs dans vos classes et propriétés ou à l'aide d'une API Fluent. Cette procédure pas à pas montre comment effectuer la configuration de l'API Fluent à l'aide de VB.NET.

Cette page suppose que vous avez une compréhension de base des Code First. Pour plus d'informations sur Code First, consultez les procédures pas à pas suivantes :

- [Code First to a New Database](#) (Création d'une nouvelle base de données avec Code First)
- [Code First à une base de données existante](#)

## Prérequis

Vous devez avoir au moins Visual Studio 2010 ou Visual Studio 2012 installé pour effectuer cette procédure pas à pas.

Si vous utilisez Visual Studio 2010, [NuGet](#) doit également être installé

## Création de l'application

Pour simplifier les choses, nous allons créer une application console de base qui utilise Code First pour effectuer l'accès aux données.

- Ouvrez Visual Studio
- **Fichier> nouveau>...**
- Sélectionnez **Windows** dans le menu de gauche et dans l'**application console** .
- Entrez **CodeFirstVBSSample** comme nom
- Sélectionnez **OK**.

## Définir le modèle

Au cours de cette étape, vous allez définir les types d'entités VB.NET POCO qui représentent le modèle conceptuel. Les classes n'ont pas besoin de dériver d'une classe de base ou d'implémenter des interfaces.

- Ajoutez une nouvelle classe au projet, entrez **SchoolModel** comme nom de classe
- Remplacez le contenu de la nouvelle classe par le code suivant :

```
Public Class Department
    Public Sub New()
        Me.Courses = New List(Of Course)()
    End Sub

    ' Primary key
    Public Property DepartmentID() As Integer
    Public Property Name() As String
    Public Property Budget() As Decimal
    Public Property StartDate() As Date
    Public Property Administrator() As Integer?
    Public Overridable Property Courses() As ICollection(Of Course)
End Class
```

```

Public Class Course
    Public Sub New()
        Me.Instructors = New HashSet(Of Instructor)()
    End Sub

    ' Primary key
    Public Property CourseID() As Integer
    Public Property Title() As String
    Public Property Credits() As Integer

    ' Foreign key that does not follow the Code First convention.
    ' The fluent API will be used to configure DepartmentID_FK to be the foreign key for this entity.
    Public Property DepartmentID_FK() As Integer

    ' Navigation properties
    Public Overridable Property Department() As Department
    Public Overridable Property Instructors() As ICollection(Of Instructor)
End Class

Public Class OnlineCourse
    Inherits Course

    Public Property URL() As String
End Class

Partial Public Class OnsiteCourse
    Inherits Course

    Public Sub New()
        Details = New OnsiteCourseDetails()
    End Sub

    Public Property Details() As OnsiteCourseDetails
End Class

' Complex type
Public Class OnsiteCourseDetails
    Public Property Time() As Date
    Public Property Location() As String
    Public Property Days() As String
End Class

Public Class Person
    ' Primary key
    Public Property PersonID() As Integer
    Public Property LastName() As String
    Public Property FirstName() As String
End Class

Public Class Instructor
    Inherits Person

    Public Sub New()
        Me.Courses = New List(Of Course)()
    End Sub

    Public Property HireDate() As Date

    ' Navigation properties
    Private privateCourses As ICollection(Of Course)
    Public Overridable Property Courses() As ICollection(Of Course)
    Public Overridable Property OfficeAssignment() As OfficeAssignment
End Class

Public Class OfficeAssignment
    ' Primary key that does not follow the Code First convention.
    ' The HasKey method is used later to configure the primary key for the entity.
    Public Property InstructorID() As Integer

```

```

Public Property Location() As String
Public Property Timestamp() As Byte()

    ' Navigation property
    Public Overridable Property Instructor() As Instructor
End Class

```

## Définir un contexte dérivé

Nous sommes sur le début de l'utilisation des types du Entity Framework donc nous devons ajouter le package NuGet EntityFramework.

- \* \* Projet-> **gérer les packages NuGet...**

### NOTE

Si vous ne disposez pas de l' **administration gérer les packages NuGet...** option vous devez installer la [dernière version de NuGet](#)

- Sélectionner l'onglet **en ligne**
- Sélectionner le package **EntityFramework**
- Cliquez sur **Installer**.

À présent, il est temps de définir un contexte dérivé, qui représente une session avec la base de données, ce qui nous permet d'interroger et d'enregistrer des données. Nous définissons un contexte qui dérive de System. Data. Entity. DbContext et expose un DbSet de type<tente de> typé pour chaque classe dans notre modèle.

- Ajoutez une nouvelle classe au projet, entrez **SchoolContext** pour le nom de la classe
- Remplacez le contenu de la nouvelle classe par le code suivant :

```

Imports System.ComponentModel.DataAnnotations
Imports System.ComponentModel.DataAnnotations.Schema
Imports System.Data.Entity
Imports System.Data.Entity.Infrastructure
Imports System.Data.Entity.ModelConfiguration.Conventions

Public Class SchoolContext
    Inherits DbContext

    Public Property OfficeAssignments() As DbSet(Of OfficeAssignment)
    Public Property Instructors() As DbSet(Of Instructor)
    Public Property Courses() As DbSet(Of Course)
    Public Property Departments() As DbSet(Of Department)

    Protected Overrides Sub OnModelCreating(ByVal modelBuilder As DbModelBuilder)
        End Sub
End Class

```

## Configuration avec l'API Fluent

Cette section montre comment utiliser les API Fluent pour configurer les types de mappage des tables, des propriétés sur les colonnes et des relations entre les tables\type dans votre modèle. L'API Fluent est exposée via le type **DbModelBuilder** et est généralement accessible en remplaçant la méthode **OnModelCreating** sur **DbContext**.

- Copiez le code suivant et ajoutez-le à la méthode **OnModelCreating** définie sur la classe **SchoolContext**. les commentaires expliquent ce que fait chaque mappage

```

' Configure Code First to ignore PluralizingTableName convention
' If you keep this convention then the generated tables
' will have pluralized names.
modelBuilder.Conventions.Remove(Of PluralizingTableNameConvention)()

' Specifying that a Class is a Complex Type

' The model defined in this topic defines a type OnsiteCourseDetails.
' By convention, a type that has no primary key specified
' is treated as a complex type.
' There are some scenarios where Code First will not
' detect a complex type (for example, if you do have a property
' called ID, but you do not mean for it to be a primary key).
' In such cases, you would use the fluent API to
' explicitly specify that a type is a complex type.
modelBuilder.ComplexType(Of OnsiteCourseDetails)()

' Mapping a CLR Entity Type to a Specific Table in the Database.

' All properties of OfficeAssignment will be mapped
' to columns in a table called t_OfficeAssignment.
modelBuilder.Entity(Of OfficeAssignment)().ToTable("t_OfficeAssignment")

' Mapping the Table-Per-Hierarchy (TPH) Inheritance

' In the TPH mapping scenario, all types in an inheritance hierarchy
' are mapped to a single table.
' A discriminator column is used to identify the type of each row.
' When creating your model with Code First,
' TPH is the default strategy for the types that
' participate in the inheritance hierarchy.
' By default, the discriminator column is added
' to the table with the name "Discriminator"
' and the CLR type name of each type in the hierarchy
' is used for the discriminator values.
' You can modify the default behavior by using the fluent API.
modelBuilder.Entity(Of Person)().
    Map(Of Person)(Function(t) t.Requires("Type").
        HasValue("Person")).
    Map(Of Instructor)(Function(t) t.Requires("Type").
        HasValue("Instructor"))

' Mapping the Table-Per-Type (TPT) Inheritance

' In the TPT mapping scenario, all types are mapped to individual tables.
' Properties that belong solely to a base type or derived type are stored
' in a table that maps to that type. Tables that map to derived types
' also store a foreign key that joins the derived table with the base table.
modelBuilder.Entity(Of Course)().ToTable("Course")
modelBuilder.Entity(Of OnsiteCourse)().ToTable("OnsiteCourse")
modelBuilder.Entity(Of OnlineCourse)().ToTable("OnlineCourse")

' Configuring a Primary Key

' If your class defines a property whose name is "ID" or "Id",
' or a class name followed by "ID" or "Id",
' the Entity Framework treats this property as a primary key by convention.
' If your property name does not follow this pattern, use the HasKey method
' to configure the primary key for the entity.
modelBuilder.Entity(Of OfficeAssignment)().
    HasKey(Function(t) t.InstructorID)

```

```

' Specifying the Maximum Length on a Property

' In the following example, the Name property
' should be no longer than 50 characters.
' If you make the value longer than 50 characters,
' you will get a DbEntityValidationException exception.
modelBuilder.Entity(Of Department)().Property(Function(t) t.Name).
    HasMaxLength(60)

' Configuring the Property to be Required

' In the following example, the Name property is required.
' If you do not specify the Name,
' you will get a DbEntityValidationException exception.
' The database column used to store this property will be non-nullable.
modelBuilder.Entity(Of Department)().Property(Function(t) t.Name).
    IsRequired()

' Switching off Identity for Numeric Primary Keys

' The following example sets the DepartmentID property to
' System.ComponentModel.DataAnnotations.DatabaseGeneratedOption.None to indicate that
' the value will not be generated by the database.
modelBuilder.Entity(Of Course)().Property(Function(t) t.CourseID).
    HasDatabaseGeneratedOption(DatabaseGeneratedOption.None)

'Specifying NOT to Map a CLR Property to a Column in the Database
modelBuilder.Entity(Of Department)().
    Ignore(Function(t) t.Administrator)

'Mapping a CLR Property to a Specific Column in the Database
modelBuilder.Entity(Of Department)().Property(Function(t) t.Budget).
    HasColumnName("DepartmentBudget")

'Configuring the Data Type of a Database Column
modelBuilder.Entity(Of Department)().Property(Function(t) t.Name).
    HasColumnType("varchar")

'Configuring Properties on a Complex Type
modelBuilder.Entity(Of OnsiteCourse)().Property(Function(t) t.Details.Days).
    HasColumnName("Days")
modelBuilder.Entity(Of OnsiteCourse)().Property(Function(t) t.Details.Location).
    HasColumnName("Location")
modelBuilder.Entity(Of OnsiteCourse)().Property(Function(t) t.Details.Time).
    HasColumnName("Time")

' Map one-to-zero or one relationship

' The OfficeAssignment has the InstructorID
' property that is a primary key and a foreign key.
modelBuilder.Entity(Of OfficeAssignment)().
    HasRequired(Function(t) t.Instructor).
    WithOptional(Function(t) t.OfficeAssignment)

' Configuring a Many-to-Many Relationship

' The following code configures a many-to-many relationship
' between the Course and Instructor types.
' In the following example, the default Code First conventions
' are used to create a join table.
' As a result the CourseInstructor table is created with
' Course_CourseID and Instructor_InstructorID columns.
modelBuilder.Entity(Of Course)().
    HasMany(Function(t) t.Instructors).
    WithMany(Function(t) t.Courses)

```

```

' Configuring a Many-to-Many Relationship and specifying the names
' of the columns in the join table

' If you want to specify the join table name
' and the names of the columns in the table
' you need to do additional configuration by using the Map method.
' The following code generates the CourseInstructor
' table with CourseID and InstructorID columns.

modelBuilder.Entity(Of Course)().
    HasMany(Function(t) t.Instructors).
    WithMany(Function(t) t.Courses).
    Map(Sub(m)
        m.ToTable("CourseInstructor")
        m.MapLeftKey("CourseID")
        m.MapRightKey("InstructorID")
    End Sub)

' Configuring a foreign key name that does not follow the Code First convention

' The foreign key property on the Course class is called DepartmentID_FK
' since that does not follow Code First conventions you need to explicitly specify
' that you want DepartmentID_FK to be the foreign key.

modelBuilder.Entity(Of Course)().
    HasRequired(Function(t) t.Department).
    WithMany(Function(t) t.Courses).
    HasForeignKey(Function(t) t.DepartmentID_FK)

' Enabling Cascade Delete

' By default, if a foreign key on the dependent entity is not nullable,
' then Code First sets cascade delete on the relationship.
' If a foreign key on the dependent entity is nullable,
' Code First does not set cascade delete on the relationship,
' and when the principal is deleted the foreign key will be set to null.
' The following code configures cascade delete on the relationship.

' You can also remove the cascade delete conventions by using:
' modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>()
' and modelBuilder.Conventions.Remove<ManyToManyCascadeDeleteConvention>()

modelBuilder.Entity(Of Course)().
    HasRequired(Function(t) t.Department).
    WithMany(Function(t) t.Courses).
    HasForeignKey(Function(d) d.DepartmentID_FK).
    WillCascadeOnDelete(False)

```

## Utilisation du modèle

Nous allons effectuer un accès aux données à l'aide de **SchoolContext** pour voir le modèle en action.

- Ouvrez le fichier Module1.vb dans lequel la fonction main est définie.
- Copiez et collez la définition Module1 suivante

```

Imports System.Data.Entity

Module Module1

Sub Main()

    Using context As New SchoolContext()

        ' Create and save a new Department.
        Console.Write("Enter a name for a new Department: ")
        Dim name = Console.ReadLine()

        Dim department = New Department With { .Name = name, .StartDate = DateTime.Now }
        context.Departments.Add(department)
        context.SaveChanges()

        ' Display all Departments from the database ordered by name
        Dim departments =
            From d In context.Departments
            Order By d.Name
            Select d

        Console.WriteLine("All Departments in the database:")
        For Each department In departments
            Console.WriteLine(department.Name)
        Next

    End Using

    Console.WriteLine("Press any key to exit...")
    Console.ReadKey()

End Sub

End Module

```

Vous pouvez maintenant exécuter l'application et la tester.

```

Enter a name for a new Department: Computing
All Departments in the database:
Computing
Press any key to exit...

```

# Code première insertion, de mettre à jour et supprimer des procédures stockées

13/09/2018 • 14 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Par défaut, Code First configure toutes les entités pour effectuer l'insertion, de mettre à jour et de supprimer des commandes à l'aide de l'accès direct à la table. En commençant dans EF6, vous pouvez configurer votre modèle Code First pour utiliser des procédures stockées pour certaines ou toutes les entités dans votre modèle.

## Mappage d'entité de base

Vous pouvez opter pour l'utilisation de procédures stockées pour insérer, mettre à jour et supprimer à l'aide de l'API Fluent.

```
modelBuilder  
    .Entity<Blog>()  
    .MapToStoredProcedures();
```

Cela provoquera Code First utiliser certaines conventions pour créer la forme attendue des procédures stockées dans la base de données.

- Trois procédures nommés **<type\_name>\_Insérer**, **<type\_name>\_mettre à jour** et **<type\_name>\_Supprimer** (par exemple, Blog\_Insert, Blog\_Update et Blog\_Delete).
- Noms de paramètres correspondent aux noms de propriété.

## NOTE

Si vous utilisez HasColumnName() ou l'attribut de colonne pour renommer la colonne pour une propriété donnée ce nom est utilisé pour les paramètres au lieu du nom de propriété.

- **La procédure stockée insert** aura un paramètre pour chaque propriété, à l'exception de celles qui sont marquées comme magasin généré (identité ou calculée). La procédure stockée doit retourner un jeu de résultats avec une colonne pour chaque propriété de base générée.
- **La mise à jour de procédure stockée** aura un paramètre pour chaque propriété, à l'exception de celles qui sont marquées avec un modèle de magasin généré de « Calculé ». Bien que certains jetons d'accès concurrentiel requièrent un paramètre pour la valeur d'origine, consultez la *jetons d'accès concurrentiel* section ci-dessous pour plus d'informations. La procédure stockée doit retourner un jeu de résultats avec une colonne pour chaque propriété calculée.
- **La suppression d'une procédure stockée** doit avoir un paramètre pour la valeur de clé de l'entité (ou plusieurs paramètres si l'entité a une clé composite). En outre, la procédure de suppression doit également avoir des paramètres pour toutes les clés étrangères association indépendante sur la table cible (les relations qui n'ont pas de propriétés de clé étrangère correspondantes déclarées dans l'entité). Bien que certains jetons d'accès concurrentiel requièrent un paramètre pour la valeur d'origine, consultez la *jetons d'accès concurrentiel* section ci-dessous pour plus d'informations.

À l'aide de la classe suivante comme exemple :

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
}
```

La valeur par défaut des procédures stockées serait :

```
CREATE PROCEDURE [dbo].[Blog_Insert]
    @Name nvarchar(max),
    @Url nvarchar(max)
AS
BEGIN
    INSERT INTO [dbo].[Blogs] ([Name], [Url])
    VALUES (@Name, @Url)

    SELECT SCOPE_IDENTITY() AS BlogId
END
CREATE PROCEDURE [dbo].[Blog_Update]
    @BlogId int,
    @Name nvarchar(max),
    @Url nvarchar(max)
AS
    UPDATE [dbo].[Blogs]
    SET [Name] = @Name, [Url] = @Url
    WHERE BlogId = @BlogId;
CREATE PROCEDURE [dbo].[Blog_Delete]
    @BlogId int
AS
    DELETE FROM [dbo].[Blogs]
    WHERE BlogId = @BlogId
```

## Substituer les valeurs par défaut

Vous pouvez remplacer tout ou partie de ce qui a été configuré par défaut.

Vous pouvez modifier le nom d'une ou plusieurs procédures stockées. Cet exemple renomme la procédure stockée update uniquement.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.HasName("modify_blog")));

```

Cet exemple renomme tous les trois procédures stockées.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.HasName("modify_blog"))
        .Delete(d => d.HasName("delete_blog"))
        .Insert(i => i.HasName("insert_blog")));

```

Dans ces exemples, les appels sont chaînées ensemble, mais vous pouvez également utiliser la syntaxe de bloc d'expression lambda.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
    {
        s.Update(u => u.HasName("modify_blog"));
        s.Delete(d => d.HasName("delete_blog"));
        s.Insert(i => i.HasName("insert_blog"));
    });

```

Cet exemple renomme le paramètre pour la propriété BlogId sur la procédure stockée update.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.Parameter(b => b.BlogId, "blog_id")));

```

Ces appels sont enchaînées et composable. Voici un exemple qui renomme les trois procédures stockées et leurs paramètres.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.HasName("modify_blog")
            .Parameter(b => b.BlogId, "blog_id")
            .Parameter(b => b.Name, "blog_name")
            .Parameter(b => b.Url, "blog_url"))
        .Delete(d => d.HasName("delete_blog")
            .Parameter(b => b.BlogId, "blog_id"))
        .Insert(i => i.HasName("insert_blog")
            .Parameter(b => b.Name, "blog_name")
            .Parameter(b => b.Url, "blog_url")));

```

Vous pouvez également modifier le nom des colonnes du jeu de résultats qui contient des valeurs de la base de données générée.

```

modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Insert(i => i.Result(b => b.BlogId, "generated_blog_identity")));

```

```

CREATE PROCEDURE [dbo].[Blog_Insert]
    @Name nvarchar(max),
    @Url nvarchar(max)
AS
BEGIN
    INSERT INTO [dbo].[Blogs] ([Name], [Url])
    VALUES (@Name, @Url)

    SELECT SCOPE_IDENTITY() AS generated_blog_id
END

```

## Relations sans une clé étrangère dans la classe (Associations indépendantes)

Lorsqu'une propriété de clé étrangère est incluse dans la définition de classe, le paramètre correspondant peut être renommé dans la même façon que toute autre propriété. Lorsqu'il existe une relation sans une propriété de

clé étrangère dans la classe, le nom de paramètre par défaut est **<navigation\_property\_name>\_<primary\_key\_name>**.

Par exemple, les définitions de classe suivantes entraînerait un paramètre Blog\_BlogId qui est attendu dans les procédures stockées pour insérer et mettre à jour des publications.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}
```

### Substituer les valeurs par défaut

Vous pouvez modifier les paramètres pour les clés étrangères qui ne figurent pas dans la classe en fournissant le chemin d'accès à la propriété de clé primaire à la méthode de paramètre.

```
modelBuilder
    .Entity<Post>()
    .MapToStoredProcedures(s =>
        s.Insert(i => i.Parameter(p => p.Blog.BlogId, "blog_id"));
```

Si vous n'avez pas une propriété de navigation sur l'entité dépendante (ex.) aucune propriété Post.Blog) vous pouvez ensuite utiliser la méthode d'Association pour identifier l'autre extrémité de la relation, puis configurer les paramètres qui correspondent à chacune des propriétés de clé ne.

```
modelBuilder
    .Entity<Post>()
    .MapToStoredProcedures(s =>
        s.Insert(i => i.Navigation<Blog>(
            b => b.Posts,
            c => c.Parameter(b => b.BlogId, "blog_id")));
```

## Jetons d'accès concurrentiel

Mise à jour et supprimer les procédures serez peut-être amené à gérer avec l'accès concurrentiel :

- Si l'entité contient des jetons d'accès concurrentiel, la procédure stockée peut éventuellement avoir un paramètre de sortie qui retourne le nombre de lignes mises à jour/supprimées (lignes affectées). Un tel paramètre doit être configuré à l'aide de la méthode RowsAffectedParameter.  
Par défaut, EF utilise la valeur de retour à partir de la méthode ExecuteNonQuery pour déterminer combien de lignes ont été affectées. Spécification d'un paramètre de sortie de lignes affectées est utile si vous effectuez une logique dans votre procédure stockée qui entraînerait la valeur de retour de ExecuteNonQuery qui est incorrect (du point de vue d'Entity Framework) à la fin de l'exécution.
- Pour chaque d'accès concurrentiel, il le jeton sera un paramètre nommé **<property\_name>\_Original** (par exemple, Timestamp\_Original). Cela recevront la valeur d'origine de cette propriété : la valeur lorsqu'il est

interrogé à partir de la base de données.

- Les jetons d'accès concurrentiel qui sont calculées par la base de données – tels que les horodatages – a uniquement un paramètre de valeur d'origine.
- Propriétés non calculées qui sont définies en tant que jetons d'accès concurrentiel aura également un paramètre pour la nouvelle valeur dans la procédure de mise à jour. Cet exemple utilise les conventions d'affectation de noms déjà vu pour les nouvelles valeurs. Un exemple de ce jeton serait être à l'aide URL d'un Blog sous forme d'un jeton d'accès concurrentiel, la nouvelle valeur est requise, car cela peut être mis à jour vers une nouvelle valeur par votre code (contrairement à un jeton de l'horodatage qui est uniquement mis à jour par la base de données).

Il s'agit d'un exemple de classe et de mettre à jour de la procédure stockée avec un jeton d'accès concurrentiel d'horodatage.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    [Timestamp]
    public byte[] Timestamp { get; set; }
}
```

```
CREATE PROCEDURE [dbo].[Blog_Update]
    @BlogId int,
    @Name nvarchar(max),
    @Url nvarchar(max),
    @Timestamp_Original rowversion
AS
    UPDATE [dbo].[Blogs]
    SET [Name] = @Name, [Url] = @Url
    WHERE BlogId = @BlogId AND [Timestamp] = @Timestamp_Original
```

Voici un exemple de classe et de mettre à jour de la procédure stockée avec un jeton d'accès concurrentiel de non calculée.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    [ConcurrencyCheck]
    public string Url { get; set; }
}
```

```
CREATE PROCEDURE [dbo].[Blog_Update]
    @BlogId int,
    @Name nvarchar(max),
    @Url nvarchar(max),
    @Url_Original nvarchar(max),
AS
    UPDATE [dbo].[Blogs]
    SET [Name] = @Name, [Url] = @Url
    WHERE BlogId = @BlogId AND [Url] = @Url_Original
```

## Substituer les valeurs par défaut

Vous pouvez éventuellement introduire un paramètre des lignes affectées.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.RowsAffectedParameter("rows_affected")));

```

Pour les jetons d'accès concurrentiel calculée de la base de données – où seule la valeur d'origine est passée : vous pouvez simplement utiliser le mécanisme de changement de nom de paramètre standard pour renommer le paramètre pour la valeur d'origine.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s =>
        s.Update(u => u.Parameter(b => b.Timestamp, "blog_timestamp")));

```

Pour les jetons d'accès concurrentiel de non calculée – où est passée à la fois la valeur d'origine et nouvelle – vous pouvez utiliser une surcharge de paramètre qui vous permet de fournir un nom pour chaque paramètre.

```
modelBuilder
    .Entity<Blog>()
    .MapToStoredProcedures(s => s.Update(u => u.Parameter(b => b.Url, "blog_url", "blog_original_url")));

```

## Relations plusieurs-à-plusieurs

Nous allons utiliser les classes suivantes comme exemple dans cette section.

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public List<Tag> Tags { get; set; }
}

public class Tag
{
    public int TagId { get; set; }
    public string TagName { get; set; }

    public List<Post> Posts { get; set; }
}
```

Plusieurs à plusieurs relations peuvent être mappées à des procédures stockées avec la syntaxe suivante.

```
modelBuilder
    .Entity<Post>()
    .HasMany(p => p.Tags)
    .WithMany(t => t.Posts)
    .MapToStoredProcedures();
```

Si aucune autre configuration n'est pas fournie la forme de procédure stockée suivante est utilisée par défaut.

- Deux procédures nommés stockées **<type\_one><type\_two>\_Insérer** et **<type\_one><type\_two>\_Supprimer** (par exemple, PostTag\_Insert et PostTag\_Delete).
- Les paramètres seront les valeurs de clés pour chaque type. Le nom de chaque paramètre en cours **<type\_name>\_<property\_name>** (par exemple, Post\_PostId et Tag\_TagId).

Voici des exemples insérer et mettre à jour des procédures stockées.

```
CREATE PROCEDURE [dbo].[PostTag_Insert]
    @Post_PostId int,
    @Tag_TagId int
AS
    INSERT INTO [dbo].[Post_Tags] (Post_PostId, Tag_TagId)
    VALUES (@Post_PostId, @Tag_TagId)
CREATE PROCEDURE [dbo].[PostTag_Delete]
    @Post_PostId int,
    @Tag_TagId int
AS
    DELETE FROM [dbo].[Post_Tags]
    WHERE Post_PostId = @Post_PostId AND Tag_TagId = @Tag_TagId
```

### Substituer les valeurs par défaut

Les noms de procédure et le paramètre peuvent être configurés de manière similaire aux procédures de l'entité stockée.

```
modelBuilder
    .Entity<Post>()
    .HasMany(p => p.Tags)
    .WithMany(t => t.Posts)
    .MapToStoredProcedures(s =>
        s.Insert(i => i.HasName("add_post_tag")
            .LeftKeyParameter(p => p.PostId, "post_id")
            .RightKeyParameter(t => t.TagId, "tag_id"))
        s.Delete(d => d.HasName("remove_post_tag")
            .LeftKeyParameter(p => p.PostId, "post_id")
            .RightKeyParameter(t => t.TagId, "tag_id")));
    );
```

# Migrations Code First

04/02/2019 • 20 minutes to read

Migrations Code First est la méthode recommandée pour faire évoluer votre schéma de base de données d'application si vous utilisez le flux de travail Code First. Les migrations fournissent un ensemble d'outils pour les opérations suivantes :

1. Créer une base de données initiale qui fonctionne avec votre modèle EF
2. Générer des migrations pour suivre les changements que vous appliquez à votre modèle EF
3. Mettre à jour votre base de données avec ces changements

La procédure suivante fournit une vue d'ensemble de Migrations Code First dans Entity Framework. Vous pouvez suivre l'intégralité de la procédure pas à pas ou passer à la rubrique qui vous intéresse. Les rubriques suivantes sont couvertes :

## Génération d'un modèle et d'une base de données initiaux

Avant de commencer à utiliser les migrations, nous avons besoin d'un projet et d'un modèle Code First. Pour cette procédure pas à pas nous utilisons le modèle canonique **Blog** et **Post**.

- Créez une application console **MigrationsDemo**
- Ajoutez la dernière version du package NuGet **EntityFramework** au projet
  - Outils -> Gestionnaire de package de bibliothèque -> Console du Gestionnaire de package
  - Exécutez la commande **Install-Package EntityFramework**
- Ajoutez un fichier **Model.cs** avec le code indiqué ci-dessous. Ce code définit une seule classe **Blog** qui constitue notre modèle de domaine et une classe **BlogContext** qui est notre contexte EF Code First

```
using System.Data.Entity;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity.Infrastructure;

namespace MigrationsDemo
{
    public class BlogContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
    }
}
```

- Maintenant que nous avons un modèle, utilisons-le pour accéder aux données. Mettez à jour le fichier **Program.cs** avec le code indiqué ci-dessous.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

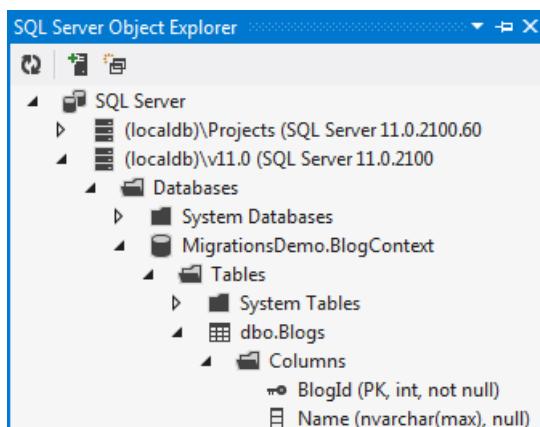
namespace MigrationsDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BlogContext())
            {
                db.Blogs.Add(new Blog { Name = "Another Blog" });
                db.SaveChanges();

                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(blog.Name);
                }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}

```

- Exécutez votre application : une base de données **MigrationsCodeDemo.BlogContext** est créée pour vous.



## Activation des migrations

Modifions un peu plus notre modèle.

- Nous introduisons une propriété Url à la classe Blog.

```
public string Url { get; set; }
```

Si vous réexécutez l'application, vous obtenez une exception `InvalidOperationException` indiquant *Le modèle permettant la sauvegarde du contexte 'BlogContext' a changé depuis la création de la base de données. Utilisez Migrations Code First pour mettre à jour la base de données* (<http://go.microsoft.com/fwlink/?LinkId=238269>).

Comme le suggère l'exception, utilisons donc Migrations Code First. La première étape est d'activer les migrations pour notre contexte.

- Exécutez la commande **Enable-Migrations** dans la Console du Gestionnaire de Package

Cette commande a ajouté un dossier **Migrations** à notre projet. Ce nouveau dossier contient deux fichiers :

- **La classe Configuration.** Cette classe vous permet de configurer le comportement des migrations pour votre contexte. Pour cette procédure pas à pas, nous utilisons simplement la configuration par défaut.  
*Comme il n'y a qu'un seul contexte Code First dans votre projet, Enable-Migrations a automatiquement renseigné le type de contexte auquel s'applique cette configuration.*
- **Une migration InitialCreate.** Cette migration a été générée, car Code First a déjà créé une base de données pour nous, avant l'activation des migrations. Le code dans cette migration générée automatiquement représente les objets qui ont déjà été créés dans la base de données. Dans notre cas, c'est la table **Blog** avec des colonnes **BlogId** et **Name**. Le nom de fichier contient un horodatage pour faciliter le classement. *Si la base de données n'a pas encore été créée, cette migration InitialCreate ne peut pas avoir été ajoutée au projet. À la place, la première fois que nous appelons Add-Migration, le code permettant de créer ces tables est généré automatiquement dans une nouvelle migration.*

### Plusieurs modèles ciblant la même base de données

Quand vous utilisez des versions antérieures à EF6, un seul modèle Code First peut servir à générer/gérer le schéma d'une base de données. Voici le résultat d'une seule table **\_\_MigrationsHistory** par base de données sans aucun moyen d'identifier les entrées qui appartiennent à tel ou tel modèle.

À partir d'EF6, la classe **Configuration** inclut une propriété **ContextKey**. Elle agit comme un identificateur unique pour chaque modèle Code First. Une colonne correspondante dans la table **\_\_MigrationsHistory** autorise des entrées provenant de plusieurs modèles pour partager la table. Par défaut, cette propriété est définie sur le nom complet de votre contexte.

## Génération et exécution des migrations

Migrations Code First a deux commandes principales que nous allons découvrir maintenant.

- **Add-Migration** génère automatiquement la prochaine migration en fonction des changements de votre modèle depuis la création de la dernière migration
- **Update-Database** applique toutes les migrations en attente à la base de données

Nous avons besoin de générer automatiquement une migration pour prendre en charge de la nouvelle propriété Url que nous avons ajoutée. La commande **Add-Migration** permet de nommer ces migrations. Appelons la notre **AddBlogUrl**.

- Exécutez la commande **Add-Migration AddBlogUrl** dans la Console du Gestionnaire de Package
- Dans le dossier **Migrations**, nous avons maintenant une nouvelle migration **AddBlogUrl**. Le nom de fichier de la migration est préfixé avec un horodatage pour faciliter le classement

```

namespace MigrationsDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddBlogUrl : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Blogs", "Url", c => c.String());
        }

        public override void Down()
        {
            DropColumn("dbo.Blogs", "Url");
        }
    }
}

```

Nous pouvons maintenant apporter des modifications ou effectuer des ajouts à cette migration, mais tout semble convenable. Utilisons **Update-Database** pour appliquer cette migration à la base de données.

- Exécutez la commande **Update-Database** dans la Console du Gestionnaire de Package
- Migrations Code First compare les migrations de notre dossier **Migrations** avec celles qui ont été appliquées à la base de données. Il voit que la migration **AddBlogUrl** doit être appliquée et l'exécute.

La base de données **MigrationsDemo.BlogContext** est désormais mise à jour pour inclure la colonne **Url** dans la table **Blogs**.

## Personnalisation des migrations

Jusqu'à présent, nous avons généré et exécuté une migration sans la modifier. Nous allons maintenant modifier le code généré par défaut.

- Modifions notre modèle en ajoutant une nouvelle propriété **Rating** à la classe **Blog**

```
public int Rating { get; set; }
```

- Ajoutons aussi une nouvelle classe **Post**

```

public class Post
{
    public int PostId { get; set; }
    [MaxLength(200)]
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}

```

- Nous allons aussi ajouter une collection **Posts** à la classe **Blog** pour former l'autre extrémité de la relation entre **Blog** et **Post**

```
public virtual List<Post> Posts { get; set; }
```

Nous utilisons la commande **Add-Migration** pour permettre à Migrations Code First de générer

automatiquement la migration qui convient le mieux selon lui. Appelons cette migration **AddPostClass**.

- Exécutez la commande **Add-Migration AddPostClass** dans la Console du Gestionnaire de Package.

Migrations Code First a obtenu de bons résultats en générant automatiquement ces changements, mais nous voulons modifier certaines choses :

1. Tout d'abord, nous ajoutons un index unique à la colonne **Posts.Title** (ajout aux lignes 22 et 29 dans le code ci-dessous).
2. Nous ajoutons aussi une colonne **Blogs.Rating** qui n'autorise pas les valeurs Null. Si des données existent déjà dans la table, elles reçoivent le type de données CLR par défaut pour la nouvelle colonne (Rating est un entier, donc la valeur est **0**). Toutefois, nous voulons spécifier une valeur par défaut égale à **3** pour que les lignes existantes de la table **Blogs** commencent avec un classement correct. (Vous pouvez voir la valeur par défaut spécifiée sur la ligne 24 du code ci-dessous)

```
namespace MigrationsDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddPostClass : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.Posts",
                c => new
                {
                    PostId = c.Int(nullable: false, identity: true),
                    Title = c.String(maxLength: 200),
                    Content = c.String(),
                    BlogId = c.Int(nullable: false),
                })
                .PrimaryKey(t => t.PostId)
                .ForeignKey("dbo.Blogs", t => t.BlogId, cascadeDelete: true)
                .Index(t => t.BlogId)
                .Index(p => p.Title, unique: true);

            AddColumn("dbo.Blogs", "Rating", c => c.Int(nullable: false, defaultValue: 3));
        }

        public override void Down()
        {
            DropIndex("dbo.Posts", new[] { "Title" });
            DropIndex("dbo.Posts", new[] { "BlogId" });
            DropForeignKey("dbo.Posts", "BlogId", "dbo.Blogs");
            DropColumn("dbo.Blogs", "Rating");
            DropTable("dbo.Posts");
        }
    }
}
```

Notre migration modifiée est prête, utilisons **Update-Database** pour mettre à jour la base de données. Cette fois, nous spécifions l'indicateur **-Verbose** pour pouvoir voir le code SQL que Migrations Code First exécute.

- Exécutez la commande **Update-Database -Verbose** dans la Console du Gestionnaire de Package.

## Déplacement de données / code SQL personnalisé

Jusqu'à présent, nous avons vu des opérations de migration qui ne changent pas ni ne déplacent des données. Voyons maintenant un cas où nous avons besoin de déplacer des données. Le déplacement de données n'est pas encore pris en charge de manière native, mais nous pouvons exécuter des commandes SQL arbitraires à tout

moment dans notre script.

- Ajoutons une propriété **Post.Abstract** à notre modèle. Plus tard, nous allons préremplir la colonne **Abstract** des publications existantes à l'aide de texte du début de la colonne **Content**.

```
public string Abstract { get; set; }
```

Nous utilisons la commande **Add-Migration** pour permettre à Migrations Code First de générer automatiquement la migration qui convient le mieux selon lui.

- Exécutez la commande **Add-Migration AddPostAbstract** dans la Console du Gestionnaire de Package.
- La migration générée prend en charge les changements de schéma, mais nous voulons aussi préremplir la colonne **Abstract** avec les 100 premiers caractères du contenu de chaque publication. Pour ce faire, nous repassons à SQL et exécutons une instruction **UPDATE** après l'ajout de la colonne. (Ajout à la ligne 12 dans le code ci-dessous)

```
namespace MigrationsDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddPostAbstract : DbMigration
    {
        public override void Up()
        {
            AddColumn("dbo.Posts", "Abstract", c => c.String());

            Sql("UPDATE dbo.Posts SET Abstract = LEFT(Content, 100) WHERE Abstract IS NULL");
        }

        public override void Down()
        {
            DropColumn("dbo.Posts", "Abstract");
        }
    }
}
```

Notre migration modifiée est prête, utilisons **Update-Database** pour mettre à jour la base de données. Spécifions l'indicateur **-Verbose** pour pouvoir voir le code SQL exécuté sur la base de données.

- Exécutez la commande **Update-Database -Verbose** dans la Console du Gestionnaire de Package.

## Migrer vers une version spécifique (y compris vers une version antérieure)

Jusqu'à présent, nous avons toujours effectué la mise à niveau vers la dernière migration, mais vous pouvez effectuer une mise à niveau vers une version spécifique, voire une version antérieure.

Supposons que nous voulons migrer notre base de données dans l'état où elle se trouvait après l'exécution de notre migration **AddBlogUrl**. Nous pouvons utiliser le commutateur **-TargetMigration** pour passer à cette migration antérieure.

- Exécutez la commande **Update-Database -TargetMigration: AddBlogUrl** dans la console du Gestionnaire de Package.

Cette commande exécute le script de passage à une version antérieure pour nos migrations **AddBlogAbstract** et **AddPostClass**.

Si vous voulez restaurer la base de données vide, vous pouvez utiliser la commande **Update-Database – TargetMigration: \$InitialDatabase**.

## Obtention d'un script SQL

Si un autre développeur veut ces changements sur son ordinateur, il peut effectuer la synchronisation dès que nous ajoutons nos changements dans le contrôle de code source. Une fois qu'il a récupéré nos nouvelles migrations, il n'a qu'à exécuter la commande Update-Database pour obtenir les changements appliqués localement. Toutefois, pour pousser ces changements sur un serveur de test et, finalement, sur un serveur de production, nous préférerons un script SQL que nous pouvons confier à notre administrateur de base de données.

- Exécutez la commande **Update-Database**, mais, cette fois, spécifiez l'indicateur **-Script** pour écrire les changements dans un script au lieu de les appliquer. Nous spécifions aussi une migration source et une migration cible pour générer le script. Nous voulons un script pour migrer une base de données vide (**\$InitialDatabase**) vers la dernière version (migration **AddPostAbstract**). *Si vous ne spécifiez pas de migration cible, Migrations utilise la dernière migration comme cible. Si vous ne spécifiez pas de migration source, Migrations utilise l'état actuel de la base de données.*
- Exécutez la commande **Update-Database –Script –SourceMigration: \$InitialDatabase – TargetMigration: AddPostAbstract** dans la console du Gestionnaire de Package.

Migrations Code First exécute le pipeline de migration, mais il écrit les changements dans un fichier .sql au lieu de les appliquer réellement. Une fois que le script est généré, il est ouvert dans Visual Studio pour que vous puissiez le consulter ou l'enregistrer.

### Génération de scripts idempotents

À partir d'EF6, si vous spécifiez **-SourceMigration \$InitialDatabase**, le script généré est « idempotent ». Les scripts idempotents peuvent mettre à niveau une base de données de n'importe quelle version vers la dernière version (ou la version spécifiée si vous utilisez **-TargetMigration**). Le script généré inclut une logique pour vérifier la table **\_\_MigrationsHistory** et applique uniquement les changements qui ne l'ont pas déjà été.

## Mise à niveau automatique au démarrage de l'application (initialiseur **MigrateDatabaseToLatestVersion**)

Si vous déployez votre application, vous pouvez lui permettre de mettre automatiquement à niveau la base de données (en appliquant les migrations en attente) quand elle démarre. Pour ce faire, inscrivez l'initialiseur de base de données **MigrateDatabaseToLatestVersion**. Un initialiseur de base de données contient simplement une logique qui garantit que la base de données est configurée correctement. Cette logique est exécutée la première fois que le contexte est utilisé dans le processus d'application (**AppDomain**).

Nous pouvons mettre à jour le fichier **Program.cs**, comme indiqué ci-dessous, afin de définir l'initialiseur **MigrateDatabaseToLatestVersion** pour BlogContext avant d'utiliser le contexte (ligne 14). Notez que vous devez également ajouter une instruction using pour l'espace de noms **System.Data.Entity** (ligne 5).

*Quand nous créons une instance de cet initialiseur, nous devons spécifier le type de contexte (**BlogContext**) et la configuration des migrations (**Configuration**) : la configuration des migrations est la classe qui a été ajoutée à notre dossier **Migrations** quand nous avons activé les migrations.*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Entity;
using MigrationsDemo.Migrations;

namespace MigrationsDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<BlogContext, Configuration>());

            using (var db = new BlogContext())
            {
                db.Blogs.Add(new Blog { Name = "Another Blog" });
                db.SaveChanges();

                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(blog.Name);
                }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

À partir de maintenant, chaque fois que notre application s'exécute, elle vérifie d'abord si la base de données ciblée est à jour, sinon, elle applique les migrations en attente.

# Automatique de Code First Migrations

27/09/2018 • 11 minutes to read

Les Migrations automatiques vous permet d'utiliser Code First Migrations sans qu'un fichier de code dans votre projet pour chaque modification apportée. Toutes les modifications peuvent être appliquées automatiquement, par exemple renomme de colonne requiert l'utilisation d'une migration de type de code.

## NOTE

Cet article suppose que vous savez comment utiliser Code First Migrations dans des scénarios de base. Si vous n'avez pas, vous devez lire [Migrations Code First](#) avant de continuer.

## Recommandation pour les environnements d'équipe

Vous pouvez intercaler migrations automatiques et de code, mais cela n'est pas recommandée dans les scénarios de développement d'équipe. Si vous faites partie d'une équipe de développeurs qui utilisent le contrôle de code source vous devez utiliser les migrations automatiques purement ou purement basée sur les migrations. Étant donné les limitations de migrations automatiques, nous recommandons à l'aide des migrations basées sur le code dans les environnements d'équipe.

## Génération d'un modèle et d'une base de données initiaux

Avant de commencer à utiliser les migrations, nous avons besoin d'un projet et d'un modèle Code First. Pour cette procédure pas à pas nous utilisons le modèle canonique **Blog** et **Post**.

- Créez un nouveau **MigrationsAutomaticDemo** application Console
- Ajoutez la dernière version du package NuGet **EntityFramework** au projet
  - **Outils -> Gestionnaire de package de bibliothèque -> Console du Gestionnaire de package**
  - Exécutez la commande **Install-Package EntityFramework**
- Ajoutez un fichier **Model.cs** avec le code indiqué ci-dessous. Ce code définit une seule classe **Blog** qui constitue notre modèle de domaine et une classe **BlogContext** qui est notre contexte EF Code First

```
using System.Data.Entity;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity.Infrastructure;

namespace MigrationsAutomaticDemo
{
    public class BlogContext : DbContext
    {
        public DbSet<Blog> Blogs { get; set; }
    }

    public class Blog
    {
        public int BlogId { get; set; }
        public string Name { get; set; }
    }
}
```

- Maintenant que nous avons un modèle, utilisons-le pour accéder aux données. Mettez à jour le fichier

**Program.cs** avec le code indiqué ci-dessous.

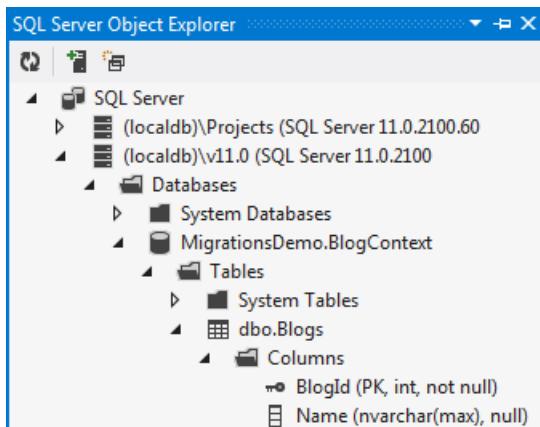
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MigrationsAutomaticDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var db = new BlogContext())
            {
                db.Blogs.Add(new Blog { Name = "Another Blog" });
                db.SaveChanges();

                foreach (var blog in db.Blogs)
                {
                    Console.WriteLine(blog.Name);
                }
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

- Exécutez votre application et vous verrez qu'un **MigrationsAutomaticCodeDemo.BlogContext** base de données est créée pour vous.



## Activation des migrations

Modifions un peu plus notre modèle.

- Nous introduisons une propriété Url à la classe Blog.

```
public string Url { get; set; }
```

Si vous réexécutez l'application, vous obtenez une exception InvalidOperationException indiquant *Le modèle permettant la sauvegarde du contexte 'BlogContext' a changé depuis la création de la base de données. Utilisez Migrations Code First pour mettre à jour la base de données* (<http://go.microsoft.com/fwlink/?LinkId=238269>).

Comme le suggère l'exception, utilisons donc Migrations Code First. Étant donné que nous souhaitons utiliser les migrations automatiques, nous allons spécifier le – **EnableAutomaticMigrations** basculer.

- Exécutez la **Enable-Migrations – EnableAutomaticMigrations** commande dans le Package Manager Console cette commande a ajouté un **Migrations** dossier à notre projet. Ce nouveau dossier contient un fichier :
  - **La classe Configuration.** Cette classe vous permet de configurer le comportement des migrations pour votre contexte. Pour cette procédure pas à pas, nous utilisons simplement la configuration par défaut. *Comme il n'y a qu'un seul contexte Code First dans votre projet, Enable-Migrations a automatiquement renseigné le type de contexte auquel s'applique cette configuration.*

## Votre première Migration automatique

Migrations Code First a deux commandes principales que nous allons découvrir maintenant.

- **Add-Migration** génère automatiquement la prochaine migration en fonction des changements de votre modèle depuis la création de la dernière migration
- **Update-Database** applique toutes les migrations en attente à la base de données

Nous allons afin d'éviter à l'aide de Add-Migration (sauf s'il faut) et de vous concentrer sur ce qui permet des Migrations Code First automatiquement calculer et appliquer les modifications. Nous allons utiliser **Update-Database** pour obtenir les Migrations Code First pour transmettre les modifications apportées à notre modèle (le nouveau **Blog.Url** propriété de l) à la base de données.

- Exécutez la **Update-Database** commande dans la Console du Gestionnaire de Package.

Le **MigrationsAutomaticDemo.BlogContext** base de données est désormais mis à jour pour inclure le **Url** colonne dans le **Blogs** table.

## Votre seconde Migration automatique

Nous allons effectuer une autre modifier et indiquer les Migrations Code First à transmettre automatiquement les modifications apportées à la base de données pour nous.

- Ajoutons aussi une nouvelle classe **Post**

```
public class Post
{
    public int PostId { get; set; }
    [MaxLength(200)]
    public string Title { get; set; }
    public string Content { get; set; }

    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

- Nous allons aussi ajouter une collection **Posts** à la classe **Blog** pour former l'autre extrémité de la relation entre **Blog** et **Post**

```
public virtual List<Post> Posts { get; set; }
```

À présent utiliser **Update-Database** pour mettre à jour de la base de données. Cette fois, nous spécifions l'indicateur **-Verbose** pour pouvoir voir le code SQL que Migrations Code First exécute.

- Exécutez la commande **Update-Database -Verbose** dans la Console du Gestionnaire de Package.

## Ajout d'un Code en fonction de Migration

Maintenant examinons quelque chose que nous pourrions utiliser une migration basée sur le code pour.

- Nous allons ajouter un **évaluation** propriété le **Blog** classe

```
public int Rating { get; set; }
```

Nous pourrions simplement exécuter **Update-Database** pour envoyer ces modifications à la base de données. Toutefois, nous ajoutons un non-nullable **Blogs.Rating** colonne, s'il existe toutes les données existantes dans la table de la valeur par défaut CLR du type de données pour la nouvelle colonne il est affectée (contrôle d'accès est entier, ce qui constitue **0**). Toutefois, nous voulons spécifier une valeur par défaut égale à **3** pour que les lignes existantes de la table **Blogs** commencent avec un classement correct. Nous allons utiliser la commande Add-Migration pour écrire cette modification out pour une migration basée sur le code afin que nous pouvons le modifier. Le **Add-Migration** commande permet de nommer ces migrations, appelons notre **AddBlogRating**.

- Exécutez le **Add-Migration AddBlogRating** commande dans la Console du Gestionnaire de Package.
- Dans le **Migrations** dossier nous disposons désormais d'un nouveau **AddBlogRating** migration. Le nom de fichier de migration est déjà résolu avec un horodatage pour faciliter le classement. Nous allons modifier le code généré pour spécifier une valeur par défaut de 3 pour Blog.Rating (ligne 10 dans le code ci-dessous)

*La migration a également un fichier code-behind qui capture des métadonnées. Ces métadonnées permettra de Migrations Code First répliquer les migrations automatiques, que nous avons effectué avant cette migration basée sur le code. Ceci est important si un autre développeur souhaite exécuter notre migrations ou lorsqu'il est temps de déployer notre application.*

```
namespace MigrationsAutomaticDemo.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddBlogRating : DbMigration
    {
        public override void Up()
        {
            AddColumn("Blogs", "Rating", c => c.Int(nullable: false, defaultValue: 3));
        }

        public override void Down()
        {
            DropColumn("Blogs", "Rating");
        }
    }
}
```

Notre migration modifiée est prête, utilisons **Update-Database** pour mettre à jour la base de données.

- Exécutez le **Update-Database** commande dans la Console du Gestionnaire de Package.

## Vers les Migrations automatiques

Nous sommes maintenant libres de revenir à des migrations automatiques pour nos modifications plus simples. Code First Migrations se chargera d'effectuer les migrations automatiques et basée sur le code dans l'ordre approprié en fonction des métadonnées stockées dans le fichier code-behind pour chaque migration basée sur le code.

- Nous allons ajouter une propriété Post.Abstract à notre modèle

```
public string Abstract { get; set; }
```

Maintenant, nous pouvons utiliser **Update-Database** pour obtenir les Migrations Code First pour envoyer cette modification à la base de données à l'aide d'une migration automatique.

- Exécutez le **Update-Database** commande dans la Console du Gestionnaire de Package.

## Récapitulatif

Dans cette procédure pas à pas, que vous avez vu comment utiliser les migrations automatiques pour transmettre le modèle change à la base de données. Vous avez également vu comment structurer et exécuter les migrations de base de code entre les migrations automatiques lorsque vous avez besoin de davantage de contrôle.

# Migrations Code First avec une base de données existante

23/11/2019 • 16 minutes to read

## NOTE

**EF 4.3 uniquement** : les fonctionnalités, les API, etc. présentées dans cette page ont été introduites dans Entity Framework 4.1. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Cet article traite de l'utilisation de Migrations Code First avec une base de données existante, qui n'a pas été créée par Entity Framework.

## NOTE

Cet article suppose que vous savez utiliser Migrations Code First dans les scénarios de base. Si ce n'est pas le cas, vous devez lire [migrations code First](#) avant de continuer.

## Captures

Si vous préférez regarder une capture vidéo plutôt que lire cet article, les deux vidéos suivantes couvrent le même contenu que cet article.

### Vidéo 1 : « migrations-en coulisses »

Cet enregistrement contient des informations sur le mode de suivi et d'utilisation des informations sur le modèle pour détecter les modifications du modèle.

### Vidéo 2 : « migrations-bases de données existantes »

En s'appuyant sur les concepts de la vidéo précédente, [cette capture](#) vidéo explique comment activer et utiliser les migrations avec une base de données existante.

## Étape 1 : créer un modèle

La première étape consiste à créer un modèle de Code First qui cible votre base de données existante. La rubrique [Code First à une base de données existante](#) fournit des conseils détaillés sur la façon de procéder.

## NOTE

Il est important de suivre les autres étapes de cette rubrique avant d'apporter des modifications à votre modèle qui nécessitent des modifications du schéma de la base de données. Les étapes suivantes requièrent que le modèle soit synchronisé avec le schéma de base de données.

## Étape 2 : activer les migrations

L'étape suivante consiste à activer les migrations. Pour ce faire, vous pouvez exécuter la commande **activer-migrations** dans la console du gestionnaire de package.

Cette commande crée un dossier dans votre solution, appelé migrations, et insère une classe unique à l'intérieur de celle-ci appelée configuration. La classe de configuration vous permet de configurer des migrations pour votre

application. Pour plus d'informations à ce sujet, consultez la rubrique [migrations code First](#).

## Étape 3 : ajouter une migration initiale

Une fois les migrations créées et appliquées à la base de données locale, vous pouvez également appliquer ces modifications à d'autres bases de données. Par exemple, votre base de données locale peut être une base de données de test et vous souhaiterez peut-être également appliquer les modifications à une base de données de production et/ou à d'autres développeurs pour tester les bases de données. Il existe deux options pour cette étape et celle que vous devez choisir dépend du fait que le schéma de toutes les autres bases de données est vide ou correspond actuellement au schéma de la base de données locale.

- **Option 1 : utiliser le schéma existant comme point de départ.** Vous devez utiliser cette approche lorsque d'autres bases de données auxquelles des migrations seront appliquées à l'avenir auront le même schéma que celui de votre base de données locale. Par exemple, vous pouvez l'utiliser si votre base de données de test locale correspond actuellement à V1 de votre base de données de production et que vous appliquerez ultérieurement ces migrations pour mettre à jour votre base de données de production vers v2.
- **Option 2 : utilisez une base de données vide comme point de départ.** Vous devez utiliser cette approche lorsque d'autres bases de données auxquelles des migrations seront appliquées à l'avenir sont vides (ou n'existent pas encore). Par exemple, vous pouvez l'utiliser si vous avez commencé à développer votre application à l'aide d'une base de données de test mais sans utiliser les migrations et que vous souhaitez ultérieurement créer une base de données de production à partir de zéro.

### Option 1 : utiliser le schéma existant comme point de départ

Migrations Code First utilise un instantané du modèle stocké dans la migration la plus récente pour détecter les modifications apportées au modèle (vous trouverez des informations détaillées à ce sujet dans [migrations code First dans les environnements d'équipe](#)). Étant donné que nous allons supposer que les bases de données ont déjà le schéma du modèle actuel, nous allons générer une migration vide (sans opération) dont le modèle actuel est un instantané.

1. Exécutez la commande **Add-migration InitialCreate – IgnoreChanges** dans la console du gestionnaire de package. Cela crée une migration vide avec le modèle actuel sous la forme d'un instantané.
2. Exécutez la commande **Update-Database** dans la console du gestionnaire de package. Cette opération applique la migration InitialCreate à la base de données. Étant donné que la migration réelle ne contient pas de modifications, il suffit d'ajouter une ligne au \_\_table MigrationsHistory indiquant que cette migration a déjà été appliquée.

### Option 2 : utiliser une base de données vide comme point de départ

Dans ce scénario, nous avons besoin de migrations pour pouvoir créer la totalité de la base de données à partir de zéro, y compris les tables déjà présentes dans notre base de données locale. Nous allons générer une migration InitialCreate qui comprend la logique permettant de créer le schéma existant. Nous allons ensuite faire en sorte que la base de données existante ressemble à cette migration.

1. Exécutez la commande **Add-migration InitialCreate** dans la console du gestionnaire de package. Cela crée une migration pour créer le schéma existant.
2. Commentez tout le code dans la méthode up de la migration nouvellement créée. Cela nous permettra d'appliquer la migration à la base de données locale sans essayer de recréer toutes les tables, etc., qui existent déjà.
3. Exécutez la commande **Update-Database** dans la console du gestionnaire de package. Cette opération applique la migration InitialCreate à la base de données. Dans la mesure où la migration réelle ne contient pas de modifications (car nous les commentons temporairement), elle ajoute simplement une ligne au \_\_table MigrationsHistory indiquant que cette migration a déjà été appliquée.
4. Annulez les marques de commentaire du code dans la méthode up. Cela signifie que lorsque cette migration est appliquée aux bases de données futures, le schéma qui existait déjà dans la base de données locale sera créé

par les migrations.

## Éléments à connaître

Vous devez connaître quelques éléments à prendre en compte lors de l'utilisation de migrations sur une base de données existante.

### **Les noms par défaut/calculés peuvent ne pas correspondre au schéma existant**

Les migrations spécifient explicitement les noms des colonnes et des tables lors de la génération de modèles automatique d'une migration. Toutefois, il existe d'autres objets de base de données dans lesquels les migrations calculent un nom par défaut lors de l'application des migrations. Cela comprend les index et les contraintes de clé étrangère. Lorsque vous ciblez un schéma existant, ces noms calculés peuvent ne pas correspondre à ce qui existe réellement dans votre base de données.

Voici quelques exemples de cas où vous devez être conscient de ce point :

#### **Si vous avez utilisé l'`option 1` : utiliser le schéma existant comme point de départ de l'étape 3 :**

- Si les modifications ultérieures de votre modèle requièrent la modification ou la suppression de l'un des objets de base de données dont le nom est différent, vous devrez modifier la migration par génération de modèles automatique pour spécifier le nom correct. Les API de migrations ont un paramètre de nom facultatif qui vous permet de le faire. Par exemple, votre schéma existant peut avoir une table de validation avec une colonne de clé étrangère `BlogId` avec un index nommé `IndexFk_BlogId`. Toutefois, par défaut, les migrations s'attendent à ce que l'index soit nommé `IX_BlogId`. Si vous apportez une modification à votre modèle qui entraîne la suppression de cet index, vous devez modifier l'appel `DropIndex` généré automatiquement pour spécifier le nom de `IndexFk_BlogId`.

#### **Si vous avez utilisé l'`option 2` : utiliser une base de données vide comme point de départ de l'étape 3 :**

- La tentative d'exécution de la méthode d'interruption de la migration initiale (c'est-à-dire la restauration d'une base de données vide) sur votre base de données locale peut échouer car les migrations tentent de supprimer les index et les contraintes de clé étrangère en utilisant des noms incorrects. Cela affecte uniquement votre base de données locale, car les autres bases de données sont créées à partir de zéro à l'aide de la méthode `up` de la migration initiale. Si vous souhaitez rétrograder votre base de données locale existante à un état vide, il est plus facile d'effectuer cette opération manuellement, soit en supprimant la base de données, soit en supprimant toutes les tables. Après cette rétrogradation initiale, tous les objets de base de données seront recréés avec les noms par défaut. ce problème ne se présentera donc pas de nouveau.
- Si les modifications ultérieures de votre modèle requièrent la modification ou la suppression de l'un des objets de base de données dont le nom est différent, cela ne fonctionnera pas sur votre base de données locale existante, puisque les noms ne correspondent pas aux valeurs par défaut. Toutefois, elle fonctionnera sur les bases de données qui ont été créées « à partir de zéro », car elles auront utilisé les noms par défaut choisis par les migrations. Vous pouvez apporter ces modifications manuellement sur votre base de données locale existante, ou envisager d'avoir des migrations qui recréent votre base de données à partir de zéro, comme c'est le cas sur les autres ordinateurs.
- Les bases de données créées à l'aide de la méthode `up` de votre migration initiale peuvent différer légèrement de la base de données locale, car les noms par défaut calculés pour les index et les contraintes de clé étrangère sont utilisés. Vous pouvez également vous retrouver avec des index supplémentaires, car les migrations créeront des index sur des colonnes de clés étrangères par défaut, ce qui n'a peut-être pas été le cas dans votre base de données locale d'origine.

### **Tous les objets de base de données ne sont pas représentés dans le modèle**

Les objets de base de données qui ne font pas partie de votre modèle ne sont pas gérés par les migrations. Cela peut inclure des vues, des procédures stockées, des autorisations, des tables qui ne font pas partie de votre modèle, des index supplémentaires, etc.

Voici quelques exemples de cas où vous devez être conscient de ce point :

- Quelle que soit l'option que vous avez choisie à l'étape 3, si des modifications ultérieures de votre modèle requièrent la modification ou la suppression de ces objets supplémentaires, les migrations d'objets supplémentaires ne sauront pas effectuer ces modifications. Par exemple, si vous supprimez une colonne contenant un index supplémentaire, les migrations ne sauront pas supprimer l'index. Vous devrez l'ajouter manuellement à la migration par génération de modèles automatique.
- Si vous avez utilisé l'option 2 : utiliser une base de données vide comme point de départ, ces objets supplémentaires ne seront pas créés par la méthode up de la migration initiale. Vous pouvez modifier les méthodes up et UpDown pour prendre en charge ces objets supplémentaires si vous le souhaitez. Pour les objets qui ne sont pas pris en charge en mode natif dans l'API migrations, tels que les vues, vous pouvez utiliser la méthode [SQL](#) pour exécuter du code SQL brut afin de les créer ou de les supprimer.

# Personnalisation de la table d'historique de migrations

07/01/2019 • 6 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement :** Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

## NOTE

Cet article suppose que vous savez comment utiliser Code First Migrations dans des scénarios de base. Si vous n'avez pas, vous devez lire [Migrations Code First](#) avant de continuer.

## Quelle est la Table d'historique de Migrations ?

Table d'historique de migrations est une table utilisée par les Migrations Code First pour stocker les détails sur les migrations appliquées à la base de données. Par défaut est le nom de la table dans la base de données `_MigrationHistory` et il est créé lors de l'application de la première migration à la base de données. Dans Entity Framework 5 cette table était une table système si l'application utilise la base de données Microsoft Sql Server. Cela a changé dans Entity Framework 6 toutefois et la table d'historique de migrations n'est plus marquée une table système.

## Pourquoi personnaliser les Migrations de la Table d'historique ?

Table d'historique de migrations est censé être utilisé uniquement par les Migrations Code First et changer manuellement peut interrompre les migrations. Cependant parfois la configuration par défaut ne convient pas, et la table doit être personnalisé, par exemple :

- Vous devez modifier les noms et/ou des facettes des colonnes pour activer un 3<sup>Bureau à distance</sup> fournisseur de Migrations de tiers
- Vous souhaitez modifier le nom de la table
- Vous devez utiliser un schéma par défaut pour le `_MigrationHistory` table
- Vous devez stocker des données supplémentaires pour une version donnée du contexte et par conséquent, vous devez ajouter une colonne supplémentaire à la table

## Mots de précaution

Modification de la table d'historique de migration est puissante, mais vous devez faire attention à ne pas abuser de cette possibilité. Runtime EF actuellement ne pas vérifie si la table d'historique des migrations personnalisé est compatible avec le runtime. Si elle n'est pas votre application peut rompre lors de l'exécution ou se comportent de manière imprévisible. Ceci est encore plus important si vous utilisez plusieurs contextes par base de données auquel cas plusieurs contextes peuvent utiliser la même table d'historique de migration pour stocker des informations sur les migrations.

## Comment personnaliser les Migrations de la Table d'historique ?

Avant de commencer, vous devez savoir que vous pouvez personnaliser la table d'historique de migrations uniquement avant d'appliquer la première migration. Maintenant, pour le code.

Tout d'abord, vous devez créer une classe dérivée de la classe `System.Data.Entity.Migrations.History.HistoryContext`. La classe `HistoryContext` est dérivée de la classe `DbContext` afin de la configuration de la table d'historique des migrations est très similaire à la configuration des modèles EF avec l'API fluent. Vous devez remplacer la méthode `OnModelCreating` et utiliser des API fluent pour configurer la table.

#### NOTE

En général, lorsque vous configurez des modèles EF ne pas devoir appeler la base. `OnModelCreating()` à partir de la méthode `OnModelCreating` substituée dans la mesure où le `DbContext.OnModelCreating()` a un corps vide. Cela n'est pas le cas lors de la configuration de la table d'historique de migrations. Dans ce cas, la première chose à faire dans votre remplacement `OnModelCreating()` est en fait appeler la base. `OnModelCreating()`. Cela configurera la table d'historique de migrations de la façon par défaut qui vous ajustez ensuite dans la méthode de substitution.

Supposons que vous souhaitez renommer la table d'historique de migrations et placez-le à un schéma personnalisé appelé « administrateur ». En outre votre DBA souhaitez que vous renommez la colonne `MigrationId` `Migration_ID`. Vous pourriez obtenir cela en créant la classe suivante dérivée `HistoryContext` :

```
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Migrations.History;

namespace CustomizableMigrationsHistoryTableSample
{
    public class MyHistoryContext : HistoryContext
    {
        public MyHistoryContext(DbConnection dbConnection, string defaultSchema)
            : base(dbConnection, defaultSchema)
        {
        }

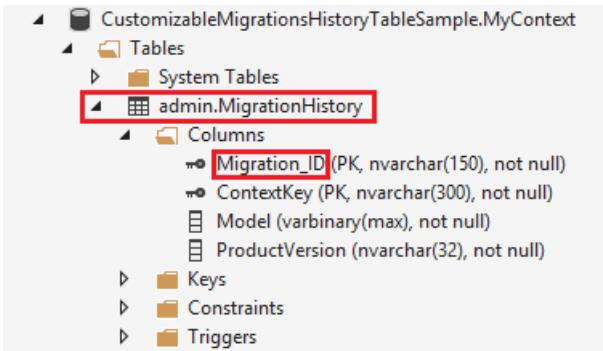
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<HistoryRow>().ToTable(tableName: "MigrationHistory", schemaName: "admin");
            modelBuilder.Entity<HistoryRow>().Property(p => p.MigrationId).HasColumnName("Migration_ID");
        }
    }
}
```

Une fois que votre `HistoryContext` personnalisée est prête vous devez faire EF prenant en charge de celui-ci en vous inscrivant via [configuration basée sur le code](#):

```
using System.Data.Entity;

namespace CustomizableMigrationsHistoryTableSample
{
    public class ModelConfiguration : DbConfiguration
    {
        public ModelConfiguration()
        {
            this.SetHistoryContext("System.Data.SqlClient",
                (connection, defaultSchema) => new MyHistoryContext(connection, defaultSchema));
        }
    }
}
```

C'est à peu près tout. Maintenant que vous pouvez accéder à la Console du Gestionnaire de Package, Enable-Migrations, Add-Migration et enfin de mise à jour la base de données. Cela doit entraîner l'ajout à la base de données une table d'historique de migrations configurée en fonction des détails que vous avez spécifié dans votre classe dérivée de HistoryContext.



# Utilisation de Migrate. exe

05/12/2019 • 9 minutes to read

Migrations Code First peut être utilisé pour mettre à jour une base de données à partir de Visual Studio, mais elle peut également être exécutée à l'aide de l'outil en ligne de commande Migrate. exe. Cette page propose une vue d'ensemble rapide de l'utilisation de Migrate. exe pour exécuter des migrations sur une base de données.

## NOTE

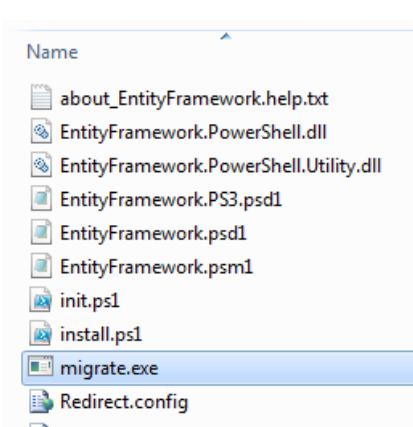
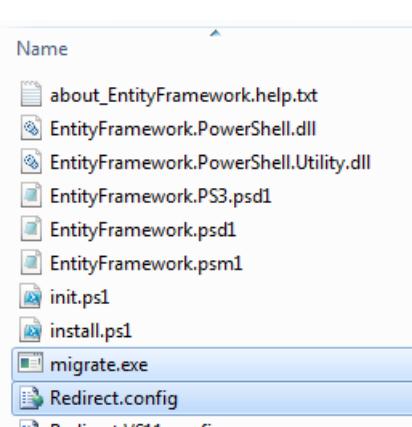
Cet article suppose que vous savez utiliser Migrations Code First dans les scénarios de base. Si ce n'est pas le cas, vous devez lire [migrations code First](#) avant de continuer.

## Copier Migrate. exe

Lorsque vous installez Entity Framework à l'aide de NuGet Migrate. exe se trouve dans le dossier Tools du package téléchargé. Dans <dossier du projet>\packages\EntityFramework.<version>\outils de \

Une fois que vous avez migré. exe, vous devez le copier à l'emplacement de l'assembly qui contient vos migrations.

Si votre application cible .NET 4 et non 4,5, vous devez copier le **fichier redirect. config** dans l'emplacement et le renommer **Migrate. exe. config**. Cela permet à Migrate. exe d'obtenir les redirections de liaison correctes pour pouvoir localiser l'assembly de Entity Framework.

| .NET 4,5  | .NET 4,0   |
|---|--|
|  |  |

## NOTE

Migrate. exe ne prend pas en charge les assemblies x64.

Une fois que vous avez déplacé MIGR. exe vers le dossier approprié, vous devez être en mesure de l'utiliser pour exécuter des migrations sur la base de données. Tout l'utilitaire est conçu pour exécuter des migrations. Il ne peut pas générer des migrations ou créer un script SQL.

## Voir les options

```
Migrate.exe /?
```

Le message ci-dessus affiche la page d'aide associée à cet utilitaire. Notez que vous devez disposer de l'EntityFramework.dll dans le même emplacement que vous exécutez Migrate.exe pour que cela fonctionne.

## Migrer vers la dernière migration

```
Migrate.exe MyMvcApplication.dll /startupConfigurationFile="..\web.config"
```

Lors de l'exécution de Migrate.exe, le seul paramètre obligatoire est l'assembly, qui est l'assembly qui contient les migrations que vous essayez d'exécuter, mais qui utilise tous les paramètres conventionnels si vous ne spécifiez pas le fichier de configuration.

## Migrer vers une migration spécifique

```
Migrate.exe MyApp.exe /startupConfigurationFile="MyApp.exe.config" /targetMigration="AddTitle"
```

Si vous souhaitez exécuter des migrations vers une migration spécifique, vous pouvez spécifier le nom de la migration. Cette opération exécute toutes les migrations précédentes si nécessaire, jusqu'à ce que vous obteniez la migration spécifiée.

## Spécifier le répertoire de travail

```
Migrate.exe MyApp.exe /startupConfigurationFile="MyApp.exe.config" /startupDirectory="c:\MyApp"
```

Si votre assembly a des dépendances ou lit des fichiers par rapport au répertoire de travail, vous devez définir startupDirectory.

## Spécifier la configuration de migration à utiliser

```
Migrate.exe MyAssembly CustomConfig /startupConfigurationFile="..\web.config"
```

Si vous disposez de plusieurs classes de configuration de migration, les classes héritent de DbMigrationConfiguration, vous devez spécifier lequel sera utilisé pour cette exécution. Cela est spécifié en fournissant le second paramètre facultatif sans commutateur comme indiqué ci-dessus.

## Fournir la chaîne de connexion

```
Migrate.exe BlogDemo.dll /connectionString="Data Source=localhost;Initial Catalog=BlogDemo;Integrated Security=SSPI" /connectionProviderName="System.Data.SqlClient"
```

Si vous souhaitez spécifier une chaîne de connexion au niveau de la ligne de commande, vous devez également fournir le nom du fournisseur. Si vous ne spécifiez pas le nom du fournisseur, une exception est levée.

## Problèmes courants

| MESSAGE D'ERREUR   | SOLUTION   |
|--|--|
| Exception non gérée : System. IO. FileLoadException : impossible de charger le fichier ou l'assembly'EntityFramework, version = 5.0.0.0, culture = neutral, PublicKeyToken = b77a5c561934e089 'ou l'une de ses dépendances. La définition du manifeste de l'assembly trouvé ne correspond pas à la référence de l'assembly. (Exception de HRESULT : 0x80131040)  | Cela signifie généralement que vous exécutez une application .NET 4 sans le fichier redirect. config. Vous devez copier le fichier redirect. config au même emplacement que Migrate. exe et le renommer en Migrate. exe. config.   |
| Exception non gérée : System. IO. FileLoadException : impossible de charger le fichier ou l'assembly'EntityFramework, version = 4.4.0.0, culture = neutral, PublicKeyToken = b77a5c561934e089 'ou l'une de ses dépendances. La définition du manifeste de l'assembly trouvé ne correspond pas à la référence de l'assembly. (Exception de HRESULT : 0x80131040)  | Cette exception signifie que vous exécutez une application .NET 4,5 avec le fichier redirect. config copié à l'emplacement Migrate. exe. Si votre application est .NET 4,5, vous n'avez pas besoin d'avoir le fichier de configuration avec les redirections à l'intérieur de. Supprimez le fichier Migrate. exe. config.  |
| ERREUR : impossible de mettre à jour la base de données pour qu'elle corresponde au modèle actuel, car des modifications sont en attente et la migration automatique est désactivée. Écrivez les modifications de modèle en attente dans une migration basée sur le code ou activez la migration automatique. Affectez la valeur true à DbMigrationsConfiguration. AutomaticMigrationsEnabled pour activer la migration automatique. | Cette erreur se produit en cas d'exécution de la migration lorsque vous n'avez pas créé de migration pour gérer les modifications apportées au modèle et que la base de données ne correspond pas au modèle. L'ajout d'une propriété à une classe de modèle, puis l'exécution de Migrate. exe sans créer de migration pour mettre à niveau la base de données en est un exemple. |
| ERREUR : le type n'est pas résolu pour le membre'System. Data. Entity. migrations. Design. ToolingFacade + UpdateRunner, EntityFramework, version = 5.0.0.0, culture = neutral, PublicKeyToken = b77a5c561934e089 '.   | Cette erreur peut être causée par la spécification d'un répertoire de démarrage incorrect. Il doit s'agir de l'emplacement de Migrate. exe.  |
| Exception non gérée : System. NullReferenceException : la référence d'objet n'est pas définie sur une instance d'un objet. dans System. Data. Entity. migrations. Console. Program. main (String [] args)  | Cela peut être dû à la non-spécification d'un paramètre obligatoire pour un scénario que vous utilisez. Par exemple, en spécifiant une chaîne de connexion sans spécifier le nom du fournisseur.   |
| ERREUR : plus d'un type de configuration de migrations a été trouvé dans l'assembly'ClassLibrary1 '. Spécifiez le nom de l'un à utiliser.  | Comme l'indique l'erreur, il y a plus d'une classe de configuration dans l'assembly donné. Vous devez utiliser le commutateur/configurationType pour spécifier l'utilisation de.   |
| ERREUR : impossible de charger le fichier ou l'assembly'<assemblyName>'ou l'une de ses dépendances. Le nom ou le code base de l'assembly donné n'est pas valide. (Exception de HRESULT : 0x80131047)   | Cela peut être dû à la spécification d'un nom d'assembly de manière incorrecte ou sans   |
| ERREUR : impossible de charger le fichier ou l'assembly'<assemblyName>'ou l'une de ses dépendances. Tentative de chargement d'un programme de format incorrect.  | Cela se produit si vous essayez d'exécuter Migrate. exe sur une application x64. EF 5,0 et versions antérieures ne fonctionnent que sur x86.   |

# Migrations Code First dans les environnements d'équipe

23/11/2019 • 30 minutes to read

## NOTE

Cet article suppose que vous savez utiliser Migrations Code First dans les scénarios de base. Si ce n'est pas le cas, vous devez lire [migrations code First](#) avant de continuer.

## Prenez un café, vous devez lire cet article entier

Les problèmes dans les environnements d'équipe concernent essentiellement la fusion des migrations lorsque deux développeurs ont généré des migrations dans leur base de code locale. Bien que les étapes à suivre pour les résoudre soient assez simples, elles vous obligent à comprendre le fonctionnement des migrations. N'hésitez pas à passer à la fin. Prenez le temps de lire tout l'article pour vous assurer que vous êtes bien en bonne voie.

## Quelques recommandations générales

Avant de nous plonger dans la manière de gérer la fusion des migrations générées par plusieurs développeurs, voici quelques recommandations générales qui vous aideront à vous préparer à la réussite.

### Chaque membre de l'équipe doit avoir une base de données de développement local

Migrations utilise la table **\_MigrationsHistory** pour stocker les migrations qui ont été appliquées à la base de données. Si vous avez plusieurs développeurs qui génèrent des migrations différentes en tentant de cibler la même base de données (et donc partager une **\_table MigrationsHistory**), les migrations seront très confuses.

Bien entendu, si vous avez des membres de l'équipe qui ne génèrent pas de migrations, il n'y a aucun problème de partage d'une base de données de développement central.

### Évitez les migrations automatiques

Le résultat est que les migrations automatiques semblent initialement bonnes dans les environnements d'équipe, mais en réalité, elles ne fonctionnent pas. Si vous souhaitez savoir pourquoi, continuez à lire, sinon, vous pouvez passer à la section suivante.

La migration automatique vous permet de mettre à jour votre schéma de base de données pour qu'il corresponde au modèle actuel sans avoir à générer des fichiers de code (migrations basées sur le code). Les migrations automatiques fonctionnent très bien dans un environnement d'équipe si vous ne les avez jamais utilisées et que vous n'avez jamais généré de migrations basées sur du code. Le problème est que les migrations automatiques sont limitées et ne gèrent pas un certain nombre d'opérations (renommages de propriétés/colonnes, déplacement de données vers une autre table, etc.). Pour gérer ces scénarios, vous devez générer des migrations basées sur du code (et modifier le code de génération de modèles automatique) qui sont mélangées entre les modifications générées par les migrations automatiques. Il est ainsi presque impossible de fusionner les modifications lorsque deux développeurs archivent des migrations.

## Captures

Si vous préférez regarder une capture vidéo plutôt que lire cet article, les deux vidéos suivantes couvrent le même contenu que cet article.

## Vidéo 1 : « migrations-en coulisses »

Cet enregistrement contient des informations sur le mode de suivi et d'utilisation des informations sur le modèle pour détecter les modifications du modèle.

## Vidéo 2 : « migrations-environnements d'équipe »

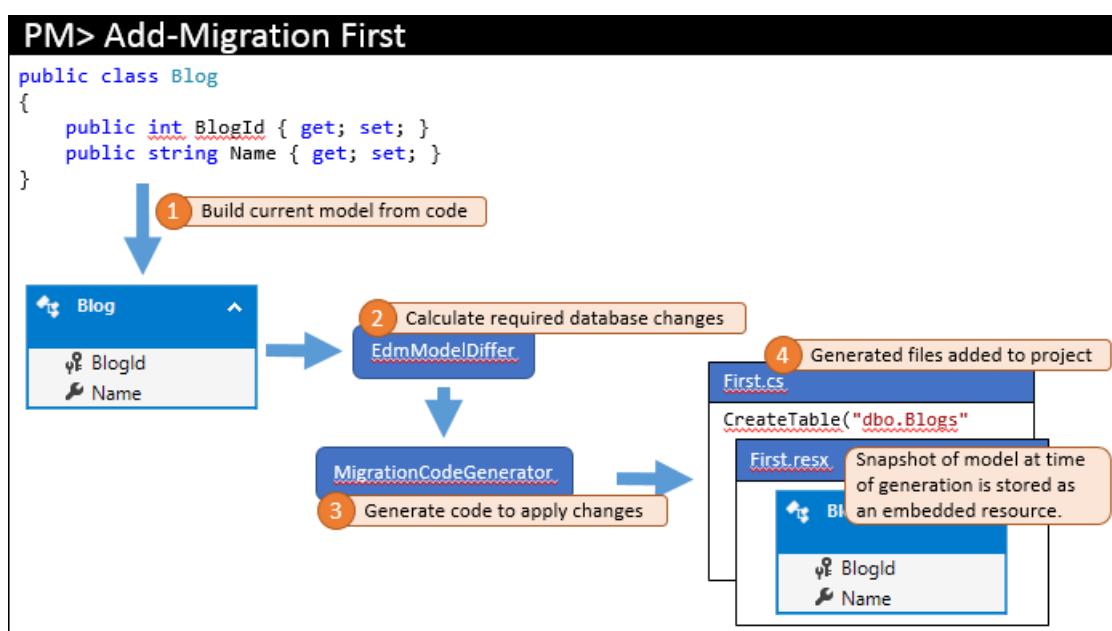
En s'appuyant sur les concepts de la vidéo précédente, [cette capture](#) vidéo couvre les problèmes qui surviennent dans un environnement d'équipe et comment les résoudre.

# Comprendre le fonctionnement des migrations

Pour utiliser avec succès des migrations dans un environnement d'équipe, il est essentiel de comprendre comment les migrations suivent et utilisent les informations relatives au modèle pour détecter les modifications du modèle.

### La première migration

Lorsque vous ajoutez la première migration à votre projet, vous exécutez un fichier comme **Add-migration en premier** dans la console du gestionnaire de package. Les étapes de haut niveau effectuées par cette commande sont illustrées ci-dessous.



Le modèle actuel est calculé à partir de votre code (1). Les objets de base de données requis sont ensuite calculés par le modèle différent (2). dans la mesure où il s'agit de la première migration, le modèle diffère simplement de l'utilisation d'un modèle vide pour la comparaison. Les modifications requises sont transmises au générateur de code pour générer le code de migration requis (3) qui est ensuite ajouté à votre solution Visual Studio (4).

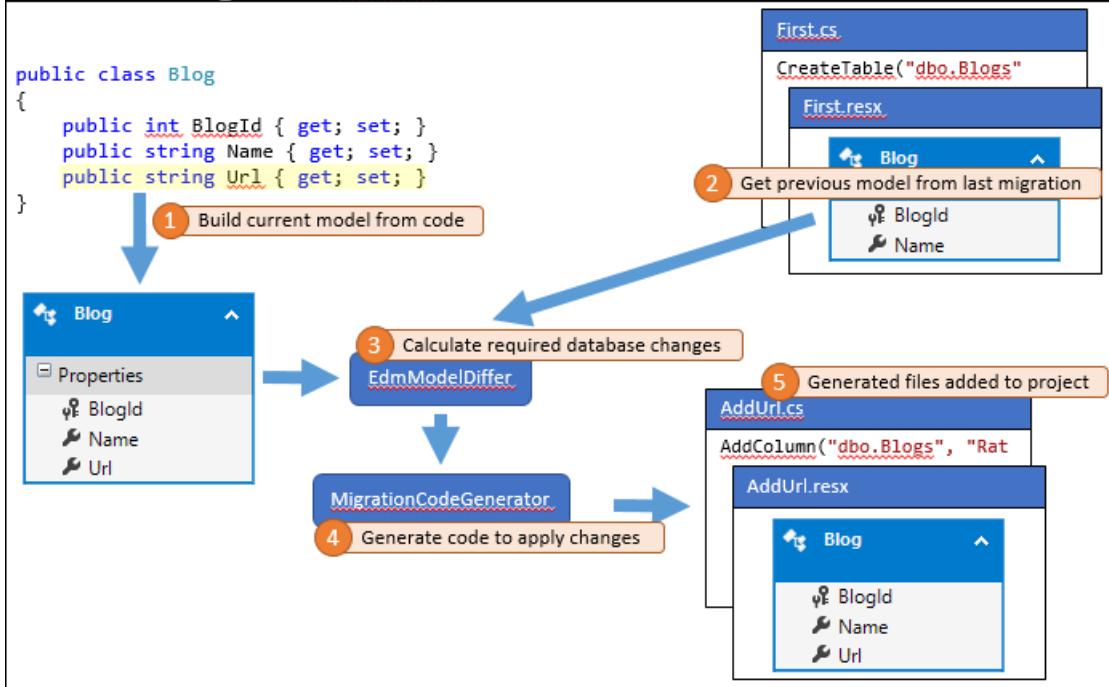
Outre le code de migration proprement dit stocké dans le fichier de code principal, les migrations génèrent également des fichiers code-behind supplémentaires. Ces fichiers sont des métadonnées qui sont utilisées par les migrations et ne sont pas des éléments que vous devez modifier. L'un de ces fichiers est un fichier de ressources (.resx) qui contient un instantané du modèle au moment de la création de la migration. Vous verrez comment cela est utilisé à l'étape suivante.

À ce stade, vous devriez probablement exécuter **Update-Database** pour appliquer vos modifications à la base de données, puis vous allez implémenter d'autres zones de votre application.

### Migrations suivantes

Plus tard, vous revenez et apportez des modifications à votre modèle. dans notre exemple, nous allons ajouter une propriété **URL** au **blog**. Vous devez ensuite émettre une commande telle que **Add-migration AddUrl** pour générer une migration en vue d'appliquer les modifications de base de données correspondantes. Les étapes de haut niveau effectuées par cette commande sont illustrées ci-dessous.

## PM > Add-Migration AddUrl



Tout comme la dernière fois, le modèle actuel est calculé à partir du code (1). Toutefois, cette fois, il existe des migrations afin que le modèle précédent soit récupéré de la dernière migration (2). Ces deux modèles sont comparés pour rechercher les modifications de base de données requises (3), puis le processus se termine comme avant.

Ce même processus est utilisé pour toutes les migrations supplémentaires que vous ajoutez au projet.

### Pourquoi ne pas vous soucier de l'instantané de modèle ?

Vous vous demandez peut-être pourquoi EF les deux avec l'instantané de modèle : pourquoi ne pas simplement examiner la base de données. Dans ce cas, lisez la suite. Si vous n'êtes pas intéressé, vous pouvez ignorer cette section.

EF conserve l'instantané de modèle pour plusieurs raisons :

- Cela permet à votre base de données de dériver du modèle EF. Ces modifications peuvent être apportées directement dans la base de données, ou vous pouvez modifier le code de génération de modèles automatique dans vos migrations pour effectuer les modifications. Voici quelques exemples de cette pratique :
  - Vous souhaitez ajouter une colonne insérée et mise à jour à une ou plusieurs de vos tables, mais vous ne souhaitez pas inclure ces colonnes dans le modèle EF. Si les migrations examinent la base de données, elle essaiera continuellement de supprimer ces colonnes chaque fois que vous générez une migration par génération de modèles automatique. À l'aide de l'instantané de modèle, EF détecte uniquement les modifications légitimes apportées au modèle.
  - Vous souhaitez modifier le corps d'une procédure stockée utilisée pour les mises à jour afin d'inclure une certaine journalisation. Si les migrations ont examiné cette procédure stockée à partir de la base de données, elle tentera continuellement de la réinitialiser en rétablissant la définition attendue par EF. À l'aide de l'instantané de modèle, EF n'effectue jamais de génération de code automatique pour modifier la procédure stockée lorsque vous modifiez la forme de la procédure dans le modèle EF.
  - Ces mêmes principes s'appliquent à l'ajout d'index supplémentaires, y compris des tables supplémentaires dans votre base de données, au mappage d'EF à une vue de base de données qui se trouve sur une table, etc.
- Le modèle EF contient plus que la seule forme de la base de données. Le fait de disposer de l'ensemble du modèle permet aux migrations d'examiner les informations sur les propriétés et les classes de votre modèle et la façon dont elles sont mappées aux colonnes et aux tables. Ces informations permettent aux migrations d'être plus intelligentes dans le code généré par le modèle. Par exemple, si vous modifiez le nom de la colonne

mappée aux migrations, vous pouvez détecter le changement de nom en vérifiant qu'il s'agit de la même propriété, ce qui ne peut pas être fait si vous n'avez que le schéma de base de données.

## Causes des problèmes dans les environnements d'équipe

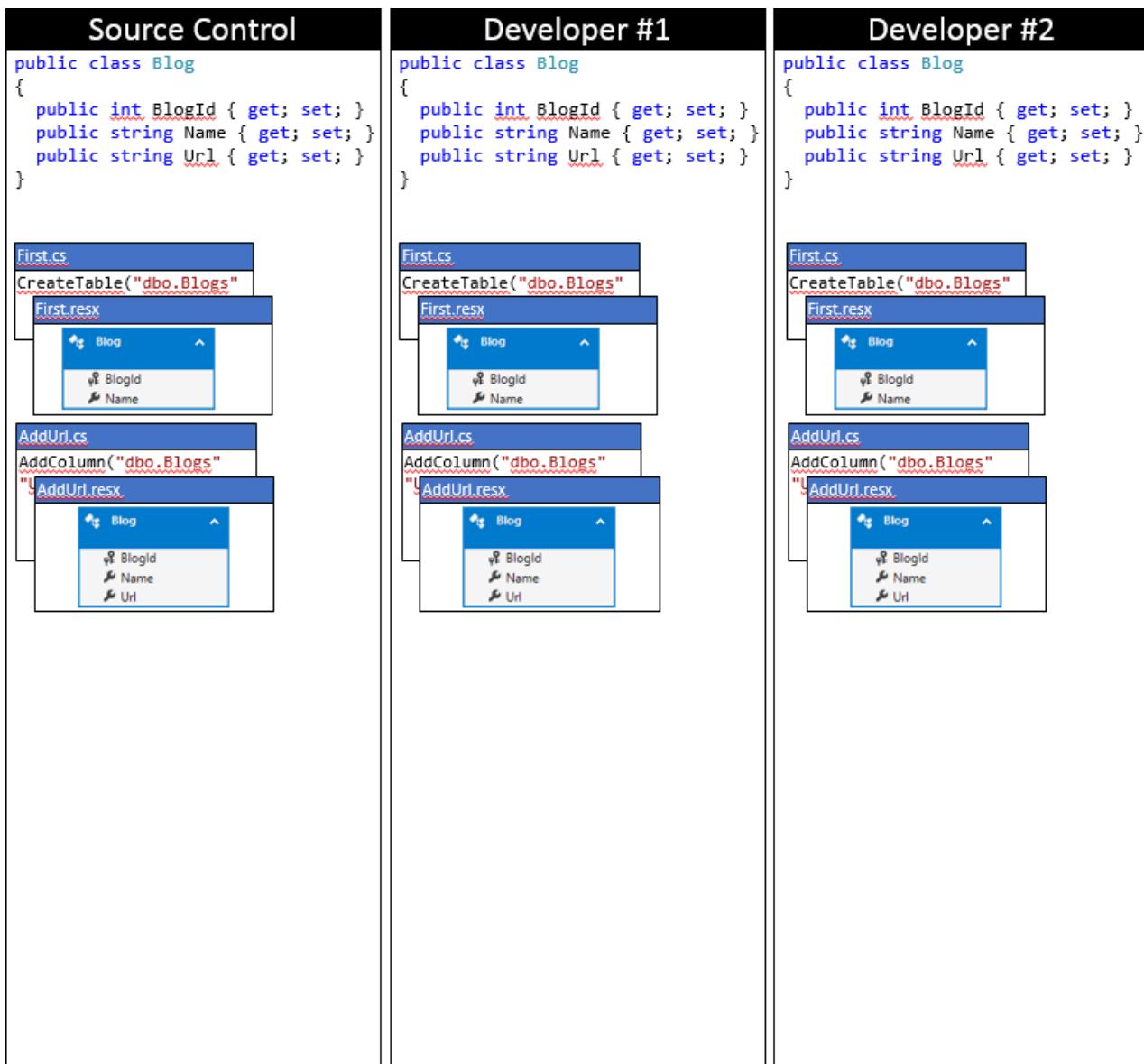
Le flux de travail abordé dans la section précédente fonctionne très bien lorsque vous êtes un développeur unique qui travaille sur une application. Il fonctionne également bien dans un environnement d'équipe si vous êtes la seule personne à apporter des modifications au modèle. Dans ce scénario, vous pouvez apporter des modifications au modèle, générer des migrations et les envoyer à votre contrôle de code source. D'autres développeurs peuvent synchroniser vos modifications et exécuter **Update-Database** pour appliquer les modifications de schéma.

Les problèmes commencent à se produire lorsque plusieurs développeurs modifient le modèle EF et envoient le contrôle de code source en même temps. Le manque d'EF est une méthode de premier ordre pour fusionner vos migrations locales avec les migrations qu'un autre développeur a soumises au contrôle de code source depuis la dernière synchronisation.

## Exemple de conflit de fusion

Tout d'abord, examinons un exemple concret de ce type de conflit de fusion. Nous poursuivrons avec l'exemple que nous avons vu précédemment. En guise de point de départ, supposons que les modifications de la section précédente ont été archivées par le développeur d'origine. Nous allons suivre deux développeurs lorsqu'ils apportent des modifications à la base de code.

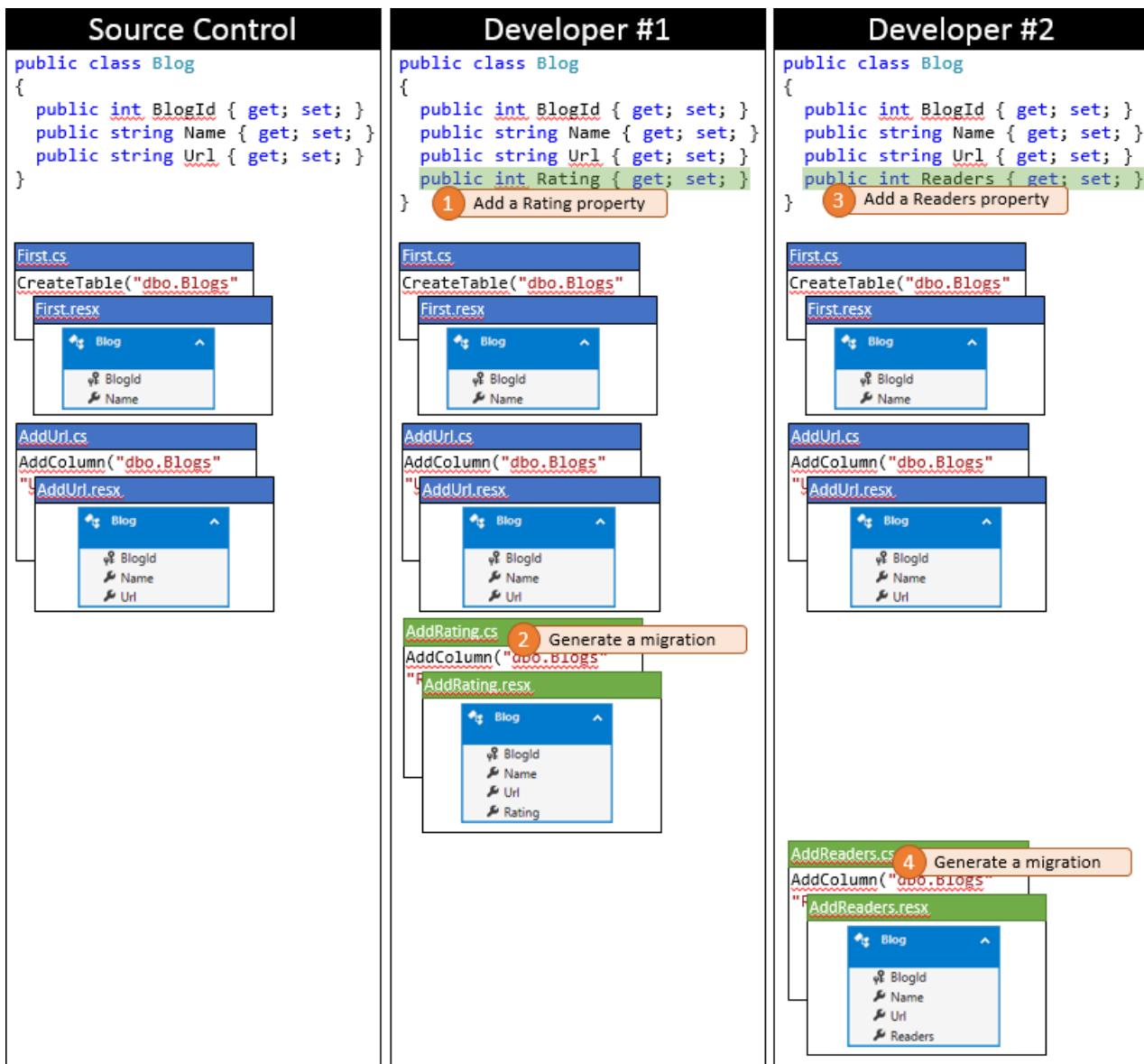
Nous allons suivre le modèle EF et les migrations à l'aide d'un certain nombre de modifications. Pour un point de départ, les deux développeurs sont synchronisés avec le référentiel de contrôle de code source, comme illustré dans le graphique suivant.



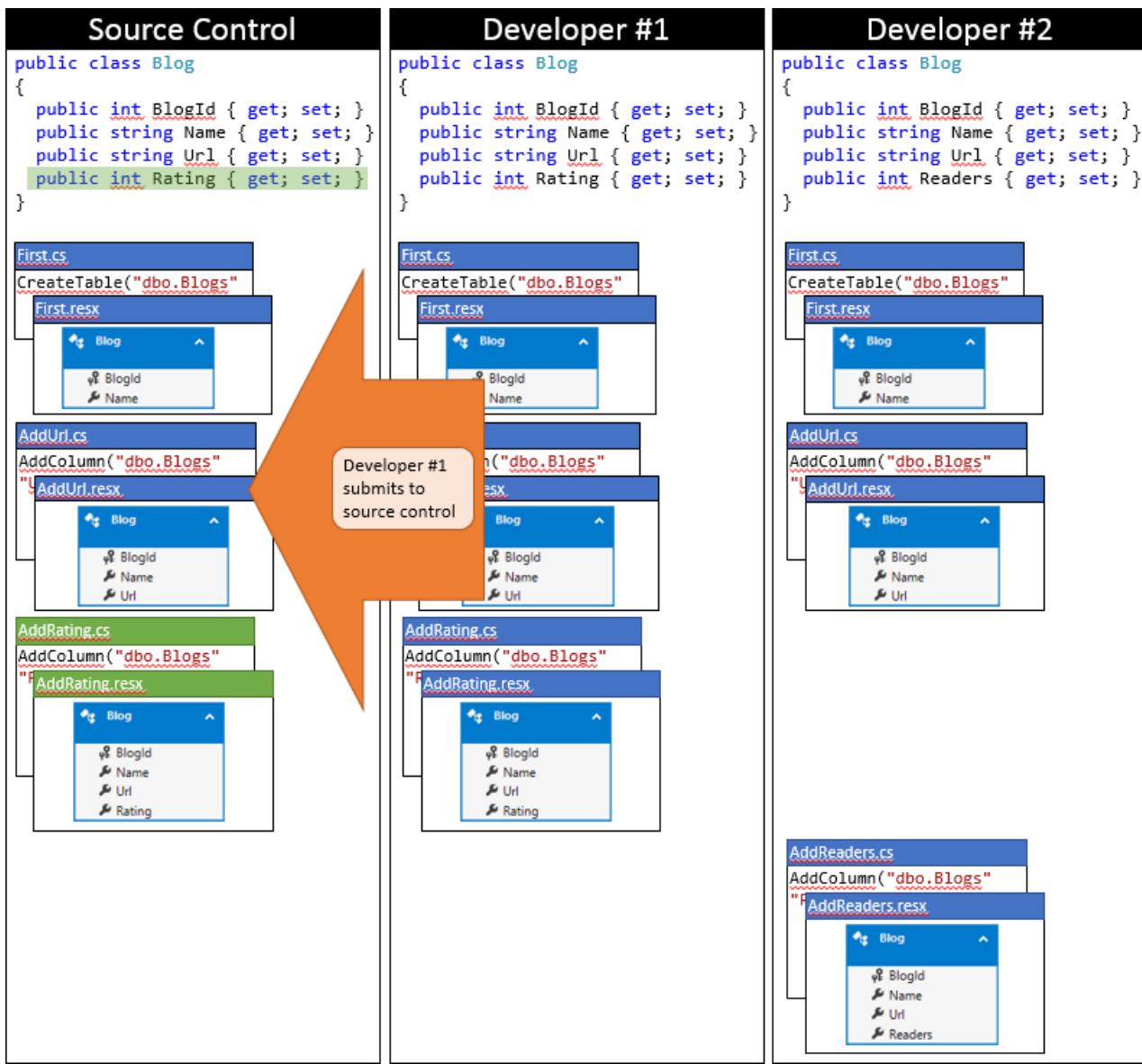
Developer #1 et Developer #2 modifient désormais le modèle EF dans leur base de code locale. Developer #1 ajoute une propriété **Rating** au **blog** , et génère une migration **AddRating** pour appliquer les modifications à la base de données. Developer #2 ajoute une propriété **Readers** au **blog** , et génère la migration **AddReaders** correspondante. Les deux développeurs exécutent **Update-Database**pour appliquer les modifications à leurs bases de données locales, puis poursuivre le développement de l'application.

#### NOTE

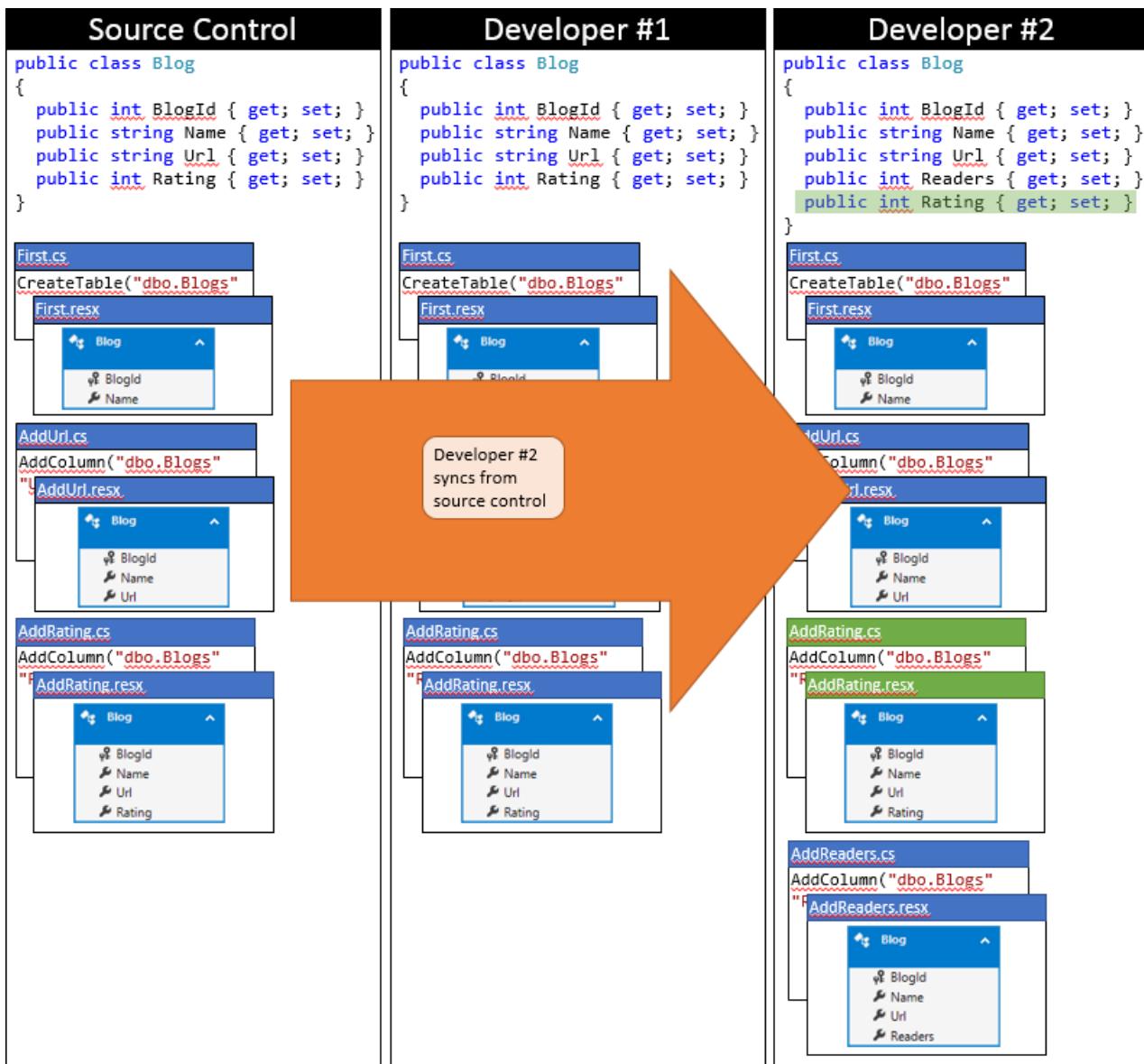
Les migrations sont précédées d'un horodateur. par conséquent, notre graphique représente que la migration AddReaders de Developer #2 vient après la migration AddRating de Developer #1. Que le développeur #1 ou le #2 a générée la migration, il ne fait aucune différence entre les problèmes de travail d'une équipe ou le processus de fusion que nous allons examiner dans la section suivante.



C'est un jour de chance pour les développeurs #1, car ils envoient d'abord leurs modifications. Étant donné que personne d'autre n'a archivé depuis qu'il a synchronisé son référentiel, il peut simplement envoyer les modifications sans effectuer de fusion.



Il est maintenant temps pour les développeurs #2 de les envoyer. Ils ne sont pas si heureux. Étant donné qu'un autre utilisateur a soumis des modifications depuis qu'il a été synchronisé, il devra extraire les modifications et les fusionner. Le système de contrôle de code source sera probablement en mesure de fusionner automatiquement les modifications au niveau du code, car elles sont très simples. L'état du référentiel local du développeur #2 après la synchronisation est illustré dans le graphique suivant.



À ce niveau, le développeur #2 peut exécuter **Update-Database** qui détectera la nouvelle migration **AddRating** (qui n'a pas été appliquée à la base de données du #2 du développeur) et l'appliquera. À présent, la colonne **Rating** est ajoutée à la table **blogs** et la base de données est synchronisée avec le modèle.

Toutefois, il existe quelques problèmes :

1. Bien que **Update-Database** applique la migration **AddRating**, il génère également un avertissement : *Impossible de mettre à jour la base de données pour qu'elle corresponde au modèle actuel, car des modifications sont en attente et la migration automatique est désactivée...* Le problème est que l'instantané de modèle stocké dans la dernière migration (**AddReader**) ne contient pas la propriété **Rating** sur le **blog** (puisque il ne fait pas partie du modèle lors de la création de la migration). Code First détecte que le modèle de la dernière migration ne correspond pas au modèle actuel et déclenche l'avertissement.
2. L'exécution de l'application génère une exception `InvalidOperationException` indiquant que «*le modèle sauvegardant le contexte «BloggingContext» a changé depuis la création de la base de données. Envisagez d'utiliser Migrations Code First pour mettre à jour la base de données...*» Là encore, le problème est que l'instantané de modèle stocké dans la dernière migration ne correspond pas au modèle actuel.
3. Enfin, l'exécution d'**Add-migration** génère désormais une migration vide (puisque il n'y a aucune modification à appliquer à la base de données). Toutefois, étant donné que migrations compare le modèle actuel à celui de la dernière migration (qui ne dispose pas de la propriété **Rating**), il génère en réalité un autre appel **AddColumn** à ajouter dans la colonne **Rating**. Bien entendu, cette migration échouerait pendant **Update-Database**, car la colonne d'**évaluation** existe déjà.

# Résolution du conflit de fusion

La bonne nouvelle, c'est qu'il n'est pas trop difficile de gérer la fusion manuellement, à condition que vous compreniez le fonctionnement des migrations. Par conséquent, si vous avez ignoré cette section... vous devez d'abord revenir en arrière et lire le reste de l'article.

Il existe deux options : la plus simple consiste à générer une migration vide qui a le modèle actuel correct en tant qu'instantané. La deuxième option consiste à mettre à jour l'instantané dans la dernière migration pour avoir l'instantané de modèle approprié. La deuxième option est un peu plus difficile et ne peut pas être utilisée dans tous les scénarios, mais elle est également plus propre, car elle n'implique pas l'ajout d'une migration supplémentaire.

## Option 1 : ajouter une migration « fusionner » vide

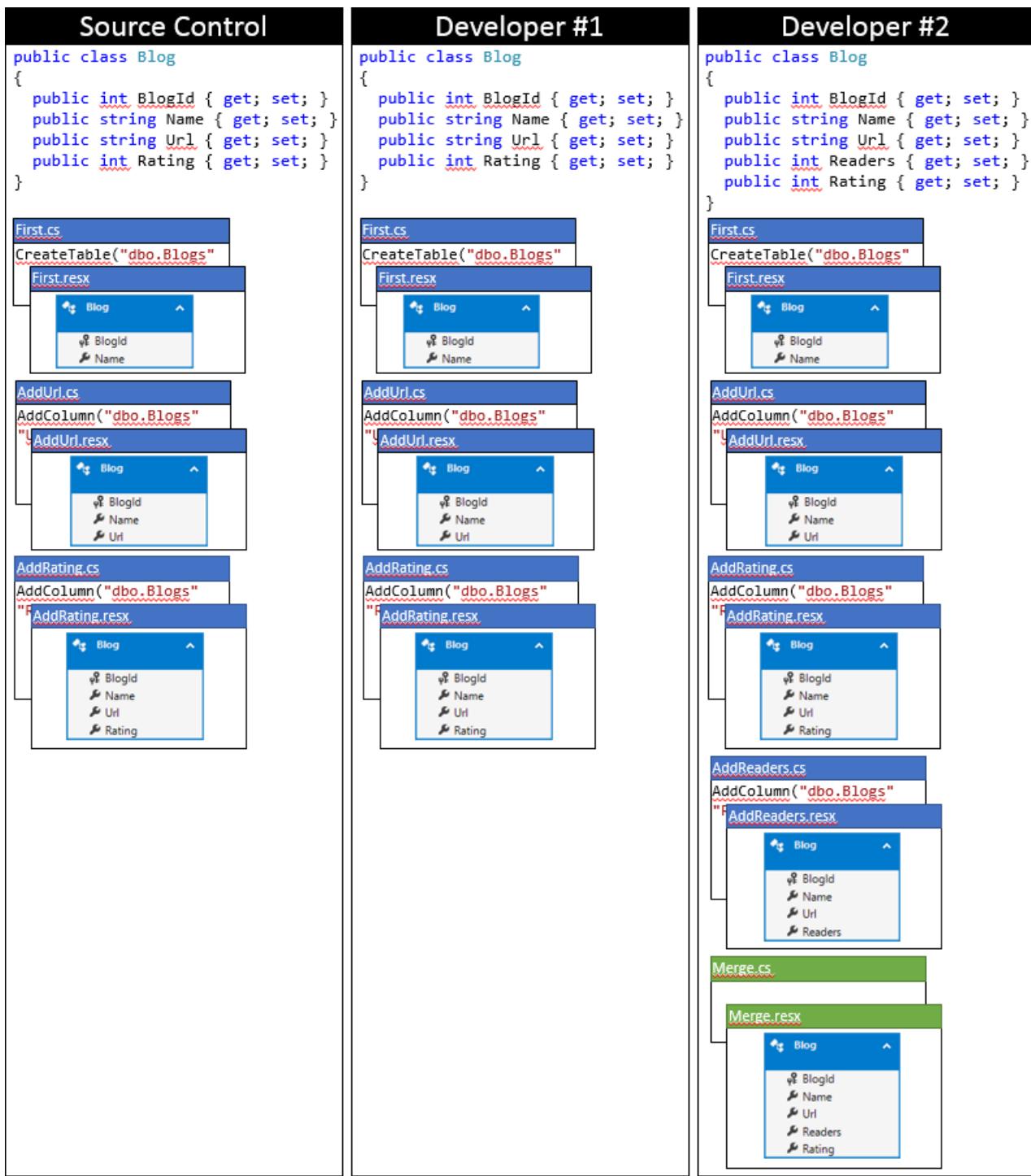
Dans cette option, nous générerons une migration vierge uniquement dans le but de vérifier que la dernière migration est stockée dans l'instantané de modèle approprié.

Cette option peut être utilisée, quelle que soit la personne qui a générée la dernière migration. Dans l'exemple, nous avons suivi le #de développement 2 pour s'occuper de la fusion et il s'est produit de générer la dernière migration. Toutefois, ces mêmes étapes peuvent être utilisées si le développeur #1 a généré la dernière migration. Les étapes s'appliquent également si plusieurs migrations sont impliquées : nous avons juste cherché deux pour la conserver simple.

Le processus suivant peut être utilisé pour cette approche, à partir du moment où vous réalisez que vous réalisez des modifications qui doivent être synchronisées à partir du contrôle de code source.

1. Vérifiez que toutes les modifications de modèle en attente dans votre base de code locale ont été écrites dans une migration. Cette étape garantit que vous ne manquez pas les modifications légitimes lorsqu'il est temps de générer la migration vide.
2. Synchroniser avec le contrôle de code source.
3. Exécutez **Update-Database** pour appliquer toutes les nouvelles migrations archivées par d'autres développeurs. **Remarque :** *si vous ne recevez aucun avertissement de la commande Update-Database, aucune nouvelle migration n'a été effectuée par les autres développeurs et il n'est pas nécessaire d'effectuer une fusion supplémentaire.*
4. Exécutez **Add-migration <pick\_un nom de\_> – IgnoreChanges** (par exemple, **Add-migration Merge – IgnoreChanges**). Cela génère une migration avec toutes les métadonnées (y compris un instantané du modèle actuel), mais ignore toutes les modifications détectées lors de la comparaison du modèle actuel à l'instantané dans les dernières migrations (ce qui signifie que vous recevez une méthode de **mise** en forme et de **réduction** de l'espace).
5. Poursuivez le développement ou envoyez-le au contrôle de code source (après avoir exécuté vos tests unitaires bien sûr).

Voici l'état de la base de code local du développeur #2 après l'utilisation de cette approche.



#### Option 2 : mettre à jour l'instantané de modèle dans la dernière migration

Cette option est très similaire à l'option 1, mais elle supprime la migration vide supplémentaire, car nous allons l'affronter, qui souhaite des fichiers de code supplémentaires dans leur solution.

**Cette approche est uniquement possible si la dernière migration existe uniquement dans votre base de code locale et n'a pas encore été soumise au contrôle de code source (par exemple, si la dernière migration a été générée par l'utilisateur qui exécute la fusion)**. La modification des métadonnées des migrations que d'autres développeurs ont déjà appliquées à leur base de données de développement, ou même pire encore, peut entraîner des effets secondaires inattendus. Pendant le processus, nous allons restaurer la dernière migration dans notre base de données locale et la réappliquer avec les métadonnées mises à jour.

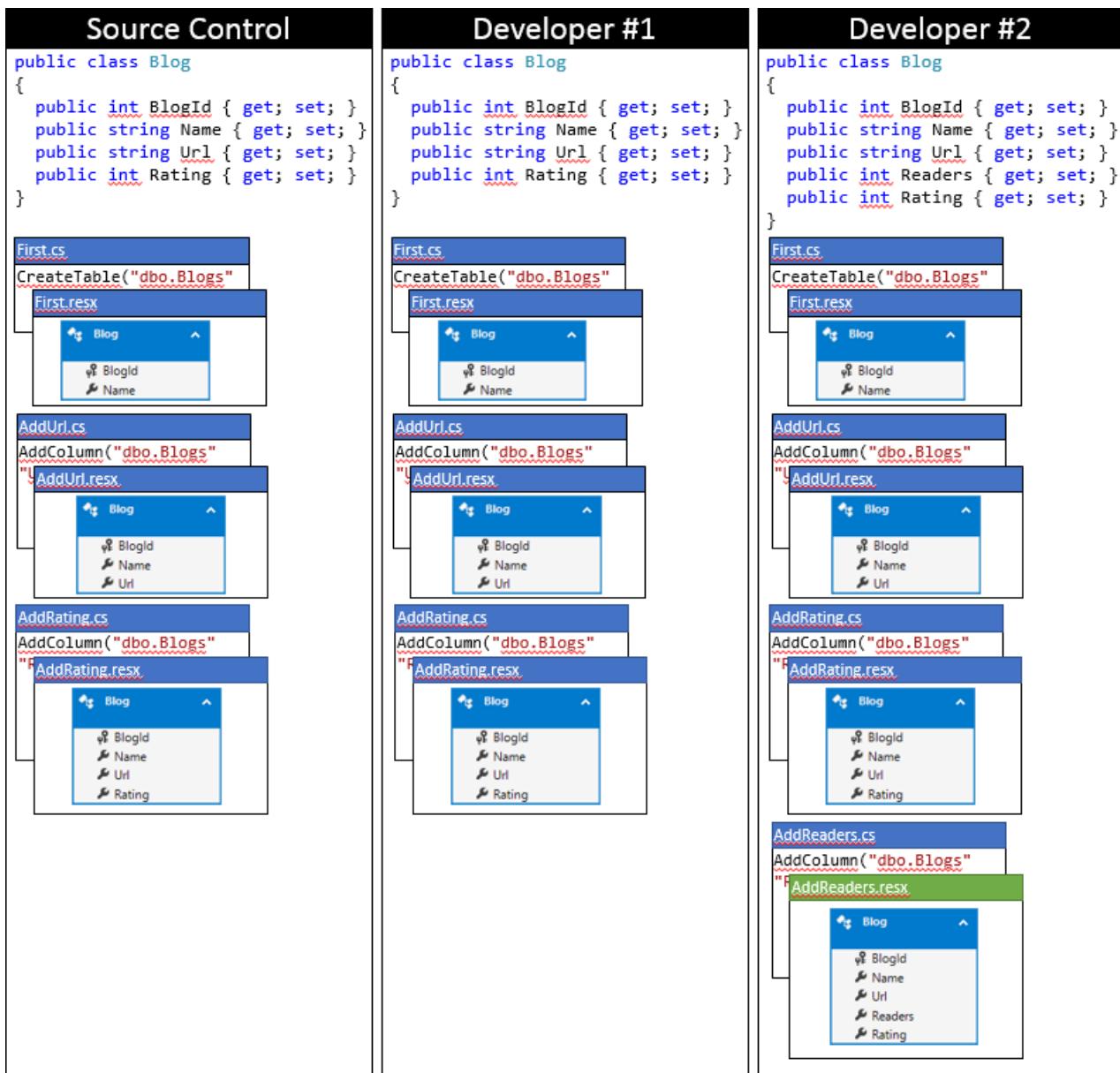
Alors que la dernière migration doit se trouver dans la base de code locale, il n'existe aucune restriction quant au nombre ou à l'ordre des migrations qui ont lieu. Il peut y avoir plusieurs migrations à partir de plusieurs développeurs différents et les mêmes étapes s'appliquent. nous avons juste étudié deux pour rester simple.

Le processus suivant peut être utilisé pour cette approche, à partir du moment où vous réalisez que vous réalisez

des modifications qui doivent être synchronisées à partir du contrôle de code source.

1. Vérifiez que toutes les modifications de modèle en attente dans votre base de code locale ont été écrites dans une migration. Cette étape garantit que vous ne manquez pas les modifications légitimes lorsqu'il est temps de générer la migration vide.
2. Synchronisez avec le contrôle de code source.
3. Exécutez **Update-Database** pour appliquer toutes les nouvelles migrations archivées par d'autres développeurs. **Remarque :** si vous ne recevez aucun avertissement de la commande `Update-Database`, aucune nouvelle migration n'a été effectuée par les autres développeurs et il n'est pas nécessaire d'effectuer une fusion supplémentaire.
4. Exécutez **Update-Database – TargetMigration <seconde\_dernière>de migration de\_** (dans l'exemple, nous avons effectué la commande **Update-Database – TargetMigration AddRating**). Cela fait passer la base de données à l'état de la deuxième dernière migration, en « désappliquant » la dernière migration à partir de la base de données. **Remarque :** cette étape est requise pour permettre la modification des métadonnées de la migration, car les métadonnées sont également stockées dans le `__MigrationsHistoryTable` de la base de données. C'est pourquoi vous ne devez utiliser cette option que si la dernière migration est uniquement dans votre base de code locale. Si la dernière migration a été appliquée à d'autres bases de données, vous devez également les restaurer et réappliquer la dernière migration pour mettre à jour les métadonnées.
5. Exécutez **Add-Migration <nom\_complet\_y\_compris\_horodatage\_de\_dernière\_de\_migration>** (dans notre exemple, nous avons effectué une opération similaire à **add-migration 201311062215252\_AddReaders**). **Remarque :** vous devez inclure l'horodateur afin que les migrations sachent que vous souhaitez modifier la migration existante plutôt que d'en créer une nouvelle. Cela permet de mettre à jour les métadonnées de la dernière migration pour correspondre au modèle actuel. Une fois la commande terminée, vous obtenez l'avertissement suivant, mais c'est exactement ce que vous voulez. « Seul le code du concepteur pour la migration «`201311062215252_AddReaders` » a été recréé. Pour regénérer la structure de l'ensemble de la migration, utilisez le paramètre `force`.
6. Exécutez **Update-Database** pour réappliquer la dernière migration avec les métadonnées mises à jour.
7. Poursuivez le développement ou envoyez-le au contrôle de code source (après avoir exécuté vos tests unitaires bien sûr).

Voici l'état de la base de code local du développeur #2 après l'utilisation de cette approche.



## Résumé

L'utilisation de Migrations Code First dans un environnement d'équipe présente des difficultés. Toutefois, une compréhension de base du fonctionnement des migrations et de quelques approches simples pour résoudre les conflits de fusion facilite la résolution de ces problèmes.

Le problème fondamental est lié à des métadonnées incorrectes stockées dans la dernière migration. De ce fait, Code First détecte de manière incorrecte que le schéma actuel et le schéma de base de données ne correspondent pas et pour générer un code incorrect dans la prochaine migration. Cette situation peut être résolue en générant une migration vierge avec le bon modèle ou en mettant à jour les métadonnées dans la dernière migration.

# Model First

23/11/2019 • 16 minutes to read

Cette vidéo et la procédure pas à pas fournissent une introduction au développement Model First à l'aide de Entity Framework. Model First vous permet de créer un nouveau modèle à l'aide du Entity Framework Designer puis de générer un schéma de base de données à partir du modèle. Le modèle est stocké dans un fichier EDMX (extension. edmx) et peut être affiché et modifié dans le Entity Framework Designer. Les classes avec lesquelles vous interagissez dans votre application sont générées automatiquement à partir du fichier EDMX.

## Regarder la vidéo

Cette vidéo et la procédure pas à pas fournissent une introduction au développement Model First à l'aide de Entity Framework. Model First vous permet de créer un nouveau modèle à l'aide du Entity Framework Designer puis de générer un schéma de base de données à partir du modèle. Le modèle est stocké dans un fichier EDMX (extension. edmx) et peut être affiché et modifié dans le Entity Framework Designer. Les classes avec lesquelles vous interagissez dans votre application sont générées automatiquement à partir du fichier EDMX.

**Présentée par :** Rowan Miller

**Vidéo:** [wmv](#) | [MP4](#) | [WMV \(zip\)](#)

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous devez avoir installé Visual Studio 2010 ou Visual Studio 2012.

Si vous utilisez Visual Studio 2010, [NuGet](#) doit également être installé.

## 1. créer l'application

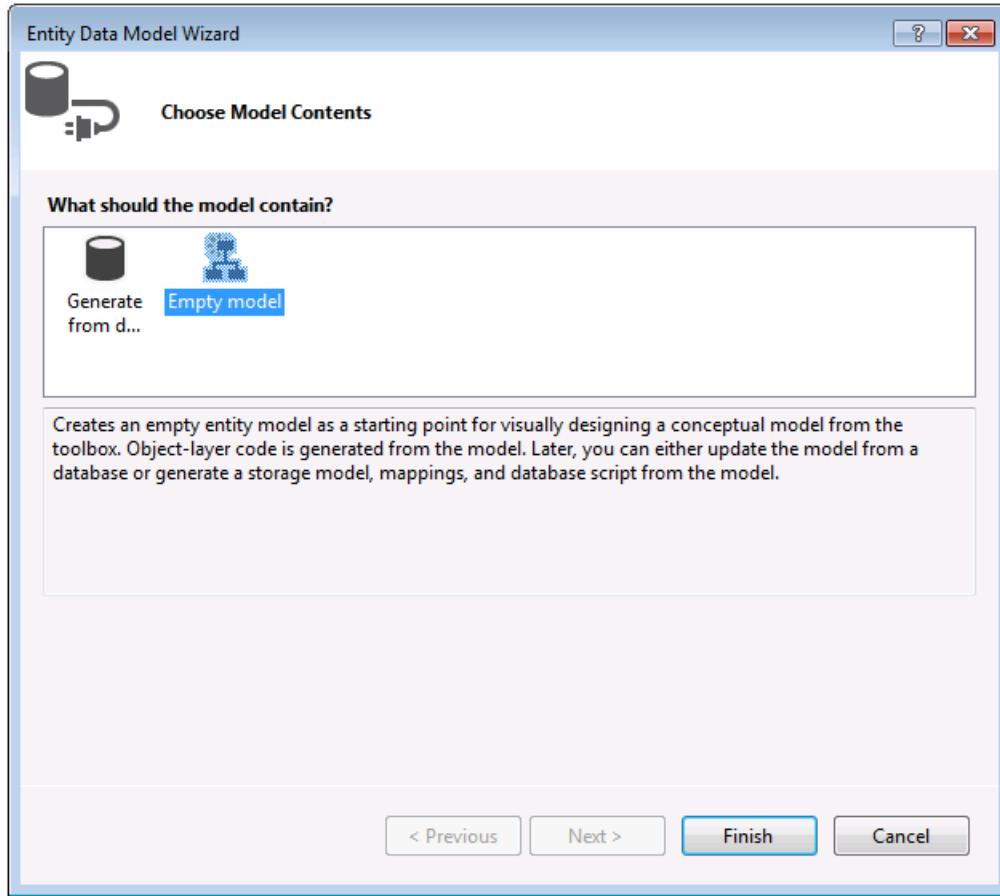
Pour simplifier les choses, nous allons créer une application console de base qui utilise le Model First pour effectuer l'accès aux données :

- Ouvrez Visual Studio
- **Fichier> nouveau>...**
- Sélectionnez **Windows** dans le menu de gauche et dans l'**application console** .
- Entrez **ModelFirstSample** comme nom
- Sélectionnez **OK**.

## 2. créer un modèle

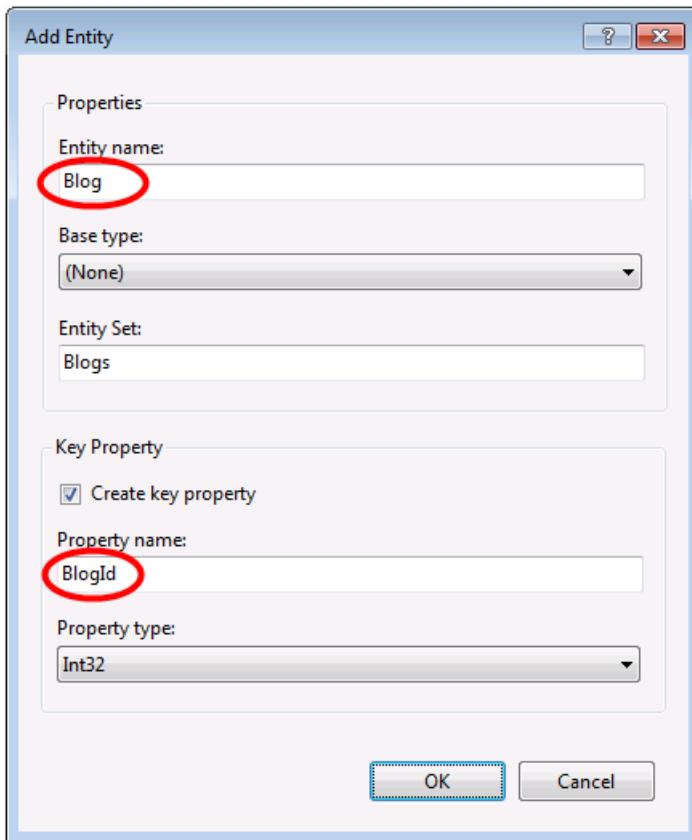
Nous allons utiliser Entity Framework Designer, inclus dans le cadre de Visual Studio, pour créer notre modèle.

- **Projet-> ajouter un nouvel élément...**
- Sélectionnez **données** dans le menu de gauche, puis **ADO.NET Entity Data Model**
- Entrez **BloggingModel** comme nom et cliquez sur **OK** pour lancer l'Assistant Entity Data Model
- Sélectionnez **modèle vide** , puis cliquez sur **Terminer**



Le Entity Framework Designer est ouvert avec un modèle vide. Nous pouvons maintenant commencer à ajouter des entités, des propriétés et des associations au modèle.

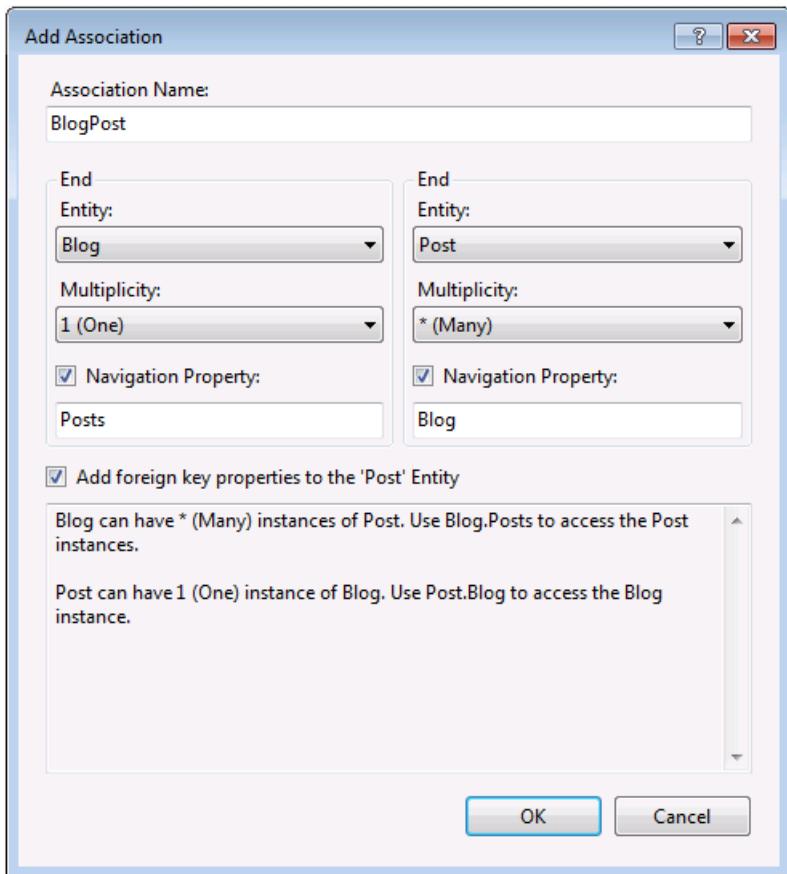
- Cliquez avec le bouton droit sur l'aire de conception et sélectionnez **Propriétés**.
- Dans la Fenêtre Propriétés modifiez le **nom du conteneur d'entités** en **BloggingContext** *il s'agit du nom du contexte dérivé qui sera généré pour vous, le contexte représente une session avec la base de données, ce qui nous permet d'interroger et d'enregistrer les données .*
- Cliquez avec le bouton droit sur l'aire de conception, puis sélectionnez **Ajouter un nouveau-> entité...**
- Entrez **blog** comme nom de l'entité et **BlogId** comme nom de clé, puis cliquez sur **OK**.



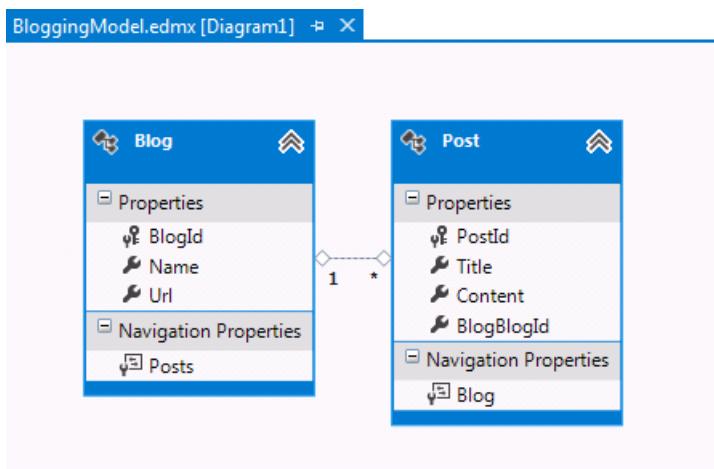
- Cliquez avec le bouton droit sur la nouvelle entité sur l'aire de conception et sélectionnez **Ajouter une nouvelle-> propriété scalaire**, entrez **nom** comme nom de la propriété.
- Répétez cette procédure pour ajouter une propriété **URL**.
- Cliquez avec le bouton droit sur la propriété **URL** dans l'aire de conception, puis sélectionnez **Propriétés**. dans la fenêtre Propriétés modifiez le paramètre **Nullable** sur **true** cela nous permet d'enregistrer un blog dans la base de données sans l'affecter à une URL
- À l'aide des techniques que vous venez d'apprendre, ajoutez une entité de **publication** avec une propriété de clé **PostId**
- Ajouter des propriétés scalaires de **titre** et de **contenu** à l'entité de **publication**

Maintenant que nous avons deux entités, il est temps d'ajouter une association (ou une relation) entre elles.

- Cliquez avec le bouton droit sur l'aire de conception, puis sélectionnez **Ajouter un nouveau-> Association...**
- Créez une terminaison de la relation pointant vers le **blog** avec une multiplicité d'**un** et l'autre point de terminaison pour **publier** avec une multiplicité de **plusieurs** cela signifie qu'un blog contient de nombreuses publications et qu'un billet appartient à un blog
- Vérifiez que la case **Ajouter les propriétés de clé étrangère à l'entité « poster »** est cochée, puis cliquez sur **OK**.



Nous disposons à présent d'un modèle simple qui nous permet de générer une base de données et de l'utiliser pour lire et écrire des données.



### Étapes supplémentaires dans Visual Studio 2010

Si vous travaillez dans Visual Studio 2010, vous devez suivre certaines étapes supplémentaires pour effectuer une mise à niveau vers la dernière version de Entity Framework. La mise à niveau est importante, car elle vous permet d'accéder à une surface d'API améliorée, qui est beaucoup plus facile à utiliser, ainsi que les derniers correctifs de bogues.

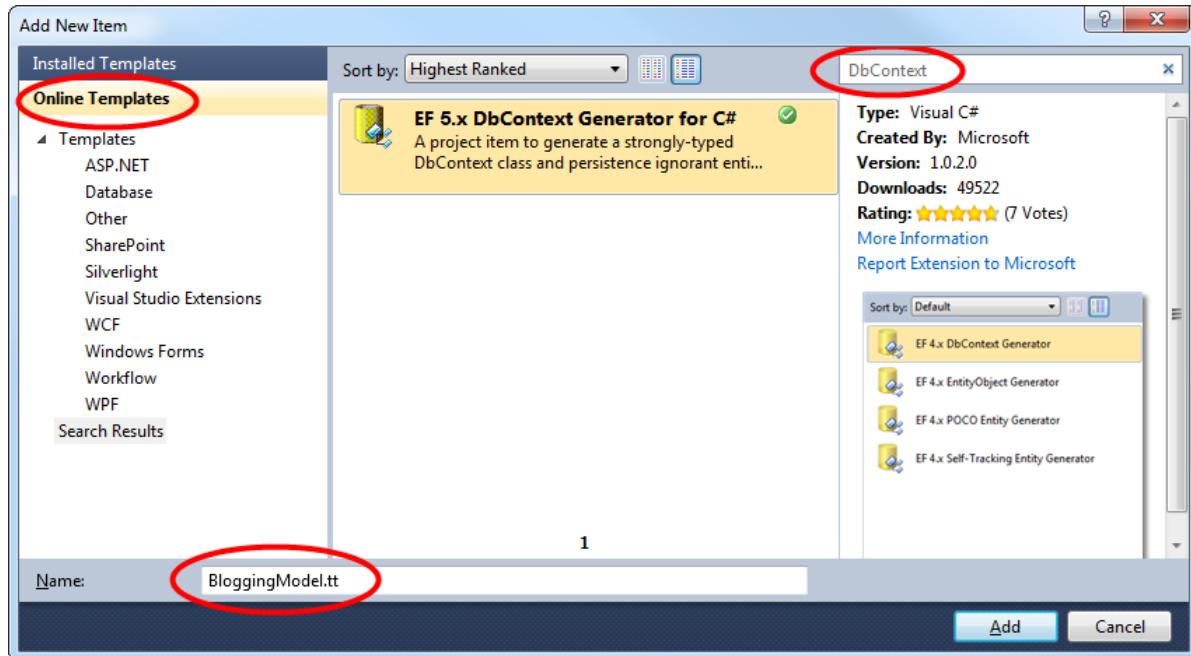
Tout d'abord, nous devons récupérer la dernière version de Entity Framework à partir de NuGet.

- **Projet-> gérer les packages NuGet...** \*si vous n'avez pas l'option **gérer les packages NuGet...** vous devez installer la [dernière version de NuGet](#) \*
- Sélectionner l'onglet **en ligne**
- Sélectionner le package **EntityFramework**
- Cliquez sur **installer**

Ensuite, nous devons permuter notre modèle pour générer le code qui utilise l'API DbContext, qui a été introduite

dans les versions ultérieures de Entity Framework.

- Cliquez avec le bouton droit sur une zone vide de votre modèle dans le concepteur EF, puis sélectionnez **Ajouter un élément de génération de code...**
- Sélectionnez **modèles en ligne** dans le menu de gauche et recherchez **DbContext**
- Sélectionnez le **Générateur de DBCONTEXT EF 5. x pour C#**, entrez **BloggingModel** comme nom et cliquez sur **Ajouter**.



### 3. génération de la base de données

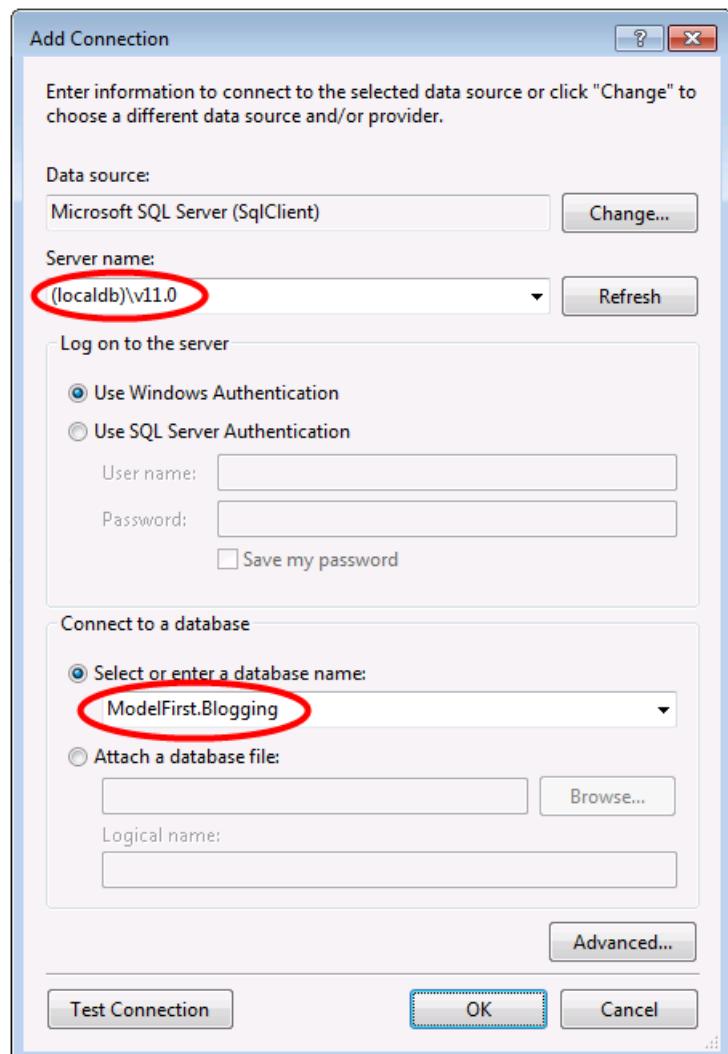
Étant donné notre modèle, Entity Framework pouvez calculer un schéma de base de données qui nous permettra de stocker et de récupérer des données à l'aide du modèle.

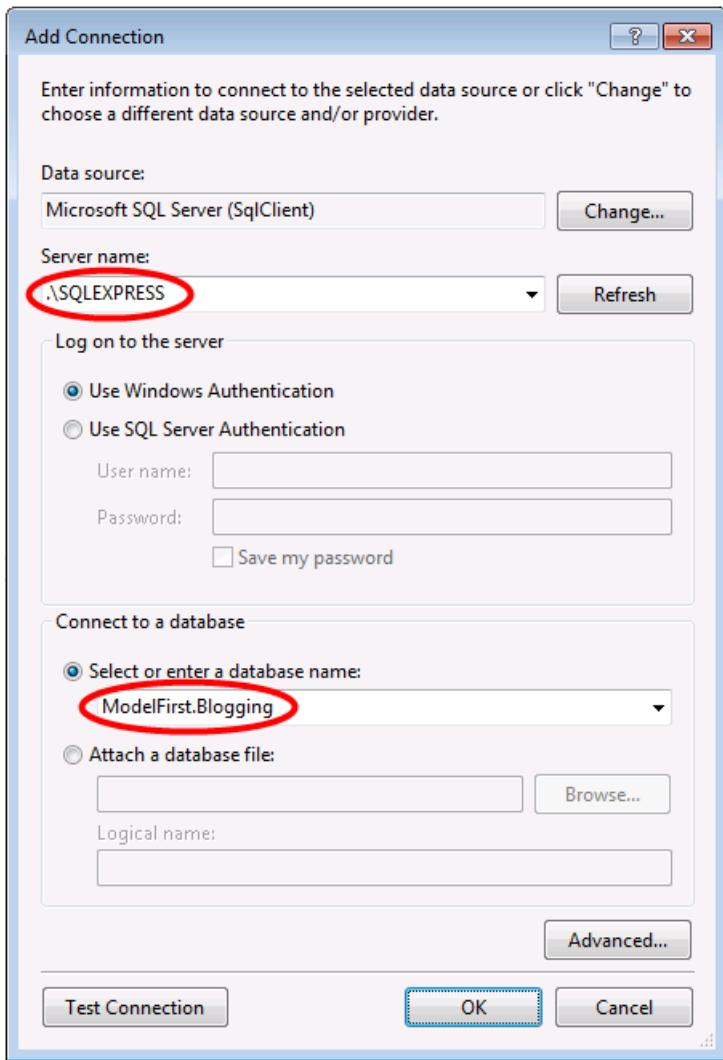
Le serveur de base de données installé avec Visual Studio diffère selon la version de Visual Studio que vous avez installée :

- Si vous utilisez Visual Studio 2010, vous allez créer une base de données SQL Express.
- Si vous utilisez Visual Studio 2012, vous allez créer une base de données de [base de données locale](#).

Commençons par générer la base de données.

- Cliquez avec le bouton droit sur l'aire de conception et sélectionnez **générer la base de données à partir du modèle...**
- Cliquez sur **nouvelle connexion...** et spécifiez la base de données locale ou SQL Express, en fonction de la version de Visual Studio que vous utilisez, entrez **ModelFirst. blog** comme nom de la base de données.



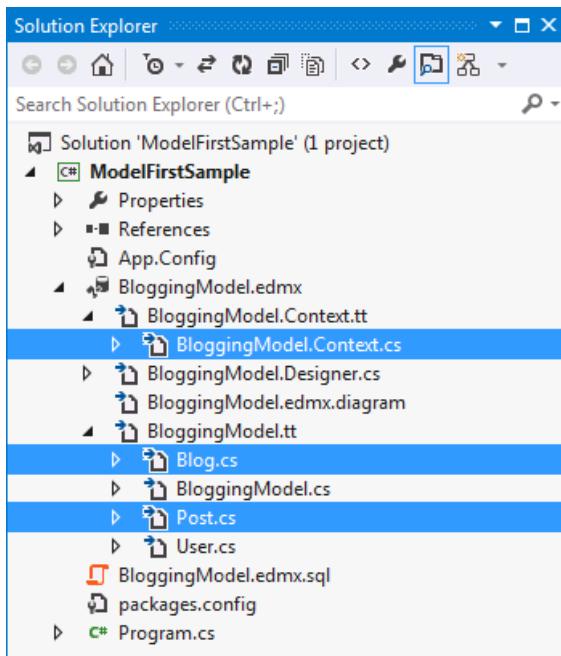


- Sélectionnez **OK**. vous serez invité à créer une nouvelle base de données, sélectionnez **Oui** .
- Sélectionnez **suivant** pour que le Entity Framework designer calcule un script pour créer le schéma de base de données.
- Une fois le script affiché, cliquez sur **Terminer** pour ajouter le script à votre projet et l'ouvrir.
- Cliquez avec le bouton droit sur le script et sélectionnez **exécuter**. vous serez invité à spécifier la base de données à laquelle vous connecter, spécifiez la base de données locale ou la SQL Server Express, selon la version de Visual Studio que vous utilisez.

## 4. lecture & écriture de données

Maintenant que nous avons un modèle, il est temps de l'utiliser pour accéder à certaines données. Les classes que nous allons utiliser pour accéder aux données sont générées automatiquement pour vous en fonction du fichier EDMX.

*Cette capture d'écran provient de Visual Studio 2012, si vous utilisez Visual Studio 2010, les fichiers BloggingModel.tt et BloggingModel.Context.tt sont directement sous votre projet plutôt que imbriqués sous le fichier EDMX.*



Implémentez la méthode main dans Program.cs comme indiqué ci-dessous. Ce code crée une nouvelle instance de notre contexte, puis l'utilise pour insérer un nouveau blog. Il utilise ensuite une requête LINQ pour récupérer tous les blogs de la base de données classée par ordre alphabétique par titre.

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Vous pouvez maintenant exécuter l'application et la tester.

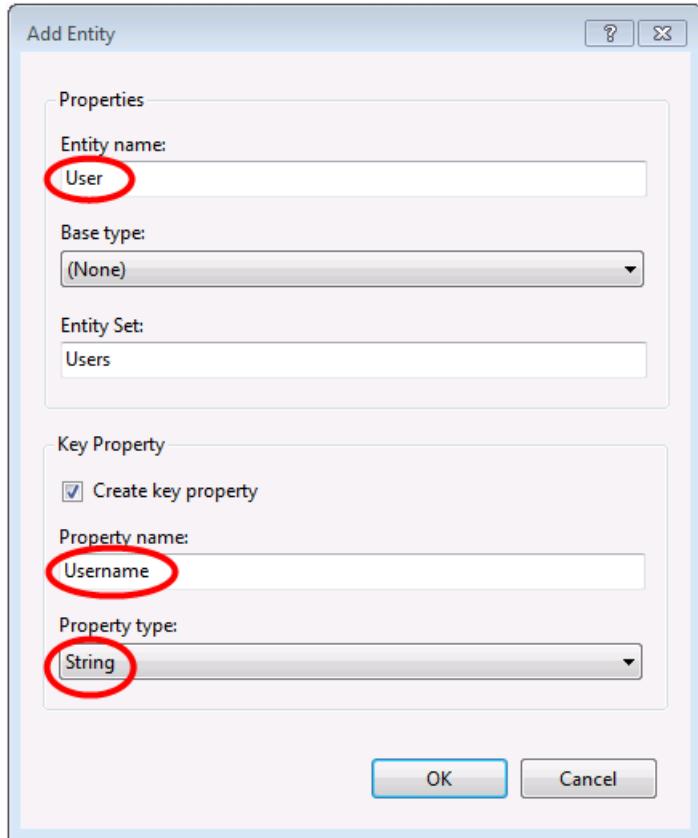
```
Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
ADO.NET Blog
Press any key to exit...
```

## 5. traitement des modifications de modèle

À présent, il est temps d'apporter des modifications à notre modèle, lorsque nous effectuons ces modifications, nous devons également mettre à jour le schéma de base de données.

Nous allons commencer par ajouter une nouvelle entité utilisateur à notre modèle.

- Ajoutez un nouveau nom d'entité **utilisateur** avec nom **d'** utilisateur comme nom de clé et **chaîne** comme type de propriété pour la clé.



- Cliquez avec le bouton droit sur la propriété **username** sur l'aire de conception, puis sélectionnez **Propriétés**. dans la fenêtre Propriétés modifiez le paramètre **MaxLength** sur **50** cela limite les données qui peuvent être stockées dans le nom d'utilisateur à 50 caractères .
- Ajouter une propriété scalaire **DisplayName** à l'entité **User**

Nous disposons désormais d'un modèle mis à jour et nous sommes prêts à mettre à jour la base de données pour qu'elle s'adapte à notre nouveau type d'entité utilisateur.

- Cliquez avec le bouton droit sur l'aire de conception, puis sélectionnez **générer la base de données à partir du modèle...**, Entity Framework calcule un script pour recréer un schéma basé sur le modèle mis à jour.
- Cliquez sur **Terminer**
- Vous pouvez recevoir des avertissements concernant le remplacement du script DDL existant et des parties de mappage et de stockage du modèle. cliquez sur **Oui** pour ces deux avertissements.
- Le script SQL mis à jour pour créer la base de données s'ouvre pour vous  
*Le script généré supprimera toutes les tables existantes, puis recréera le schéma à partir de zéro. Cela peut fonctionner pour le développement local, mais n'est pas une solution viable pour envoyer des modifications à une base de données qui a déjà été déployée. Si vous avez besoin de publier des modifications dans une base de données qui a déjà été déployée, vous devez modifier le script ou utiliser un outil de comparaison de schémas pour calculer un script de migration.*
- Cliquez avec le bouton droit sur le script et sélectionnez **exécuter**. vous serez invité à spécifier la base de données à laquelle vous connecter, spécifiez la base de données locale ou la SQL Server Express, selon la

version de Visual Studio que vous utilisez.

## Résumé

Dans cette procédure pas à pas, nous avons examiné Model First développement, ce qui nous a permis de créer un modèle dans le concepteur EF, puis de générer une base de données à partir de ce modèle. Nous avons ensuite utilisé le modèle pour lire et écrire des données de la base de données. Enfin, nous avons mis à jour le modèle, puis recréé le schéma de base de données pour qu'il corresponde au modèle.

# Database First

23/11/2019 • 13 minutes to read

Cette vidéo et la procédure pas à pas fournissent une introduction au développement Database First à l'aide de Entity Framework. Database First vous permet de rétroconcevoir un modèle à partir d'une base de données existante. Le modèle est stocké dans un fichier EDMX (extension. edmx) et peut être affiché et modifié dans le Entity Framework Designer. Les classes avec lesquelles vous interagissez dans votre application sont générées automatiquement à partir du fichier EDMX.

## Regarder la vidéo

Cette vidéo fournit une introduction au développement Database First à l'aide de Entity Framework. Database First vous permet de rétroconcevoir un modèle à partir d'une base de données existante. Le modèle est stocké dans un fichier EDMX (extension. edmx) et peut être affiché et modifié dans le Entity Framework Designer. Les classes avec lesquelles vous interagissez dans votre application sont générées automatiquement à partir du fichier EDMX.

**Présentée par :** Rowan Miller

**Vidéo:** [wmv](#) | [MP4](#) | [WMV \(zip\)](#)

## Conditions préalables

Vous devez avoir au moins Visual Studio 2010 ou Visual Studio 2012 installé pour effectuer cette procédure pas à pas.

Si vous utilisez Visual Studio 2010, [NuGet](#) doit également être installé.

## 1. créer une base de données existante

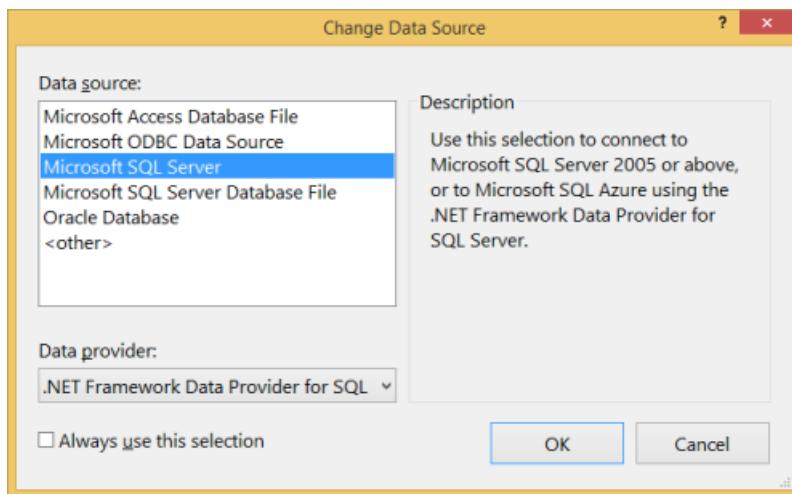
En général, lorsque vous ciblez une base de données existante, elle est déjà créée, mais pour cette procédure pas à pas, nous devons créer une base de données à laquelle accéder.

Le serveur de base de données installé avec Visual Studio diffère selon la version de Visual Studio que vous avez installée :

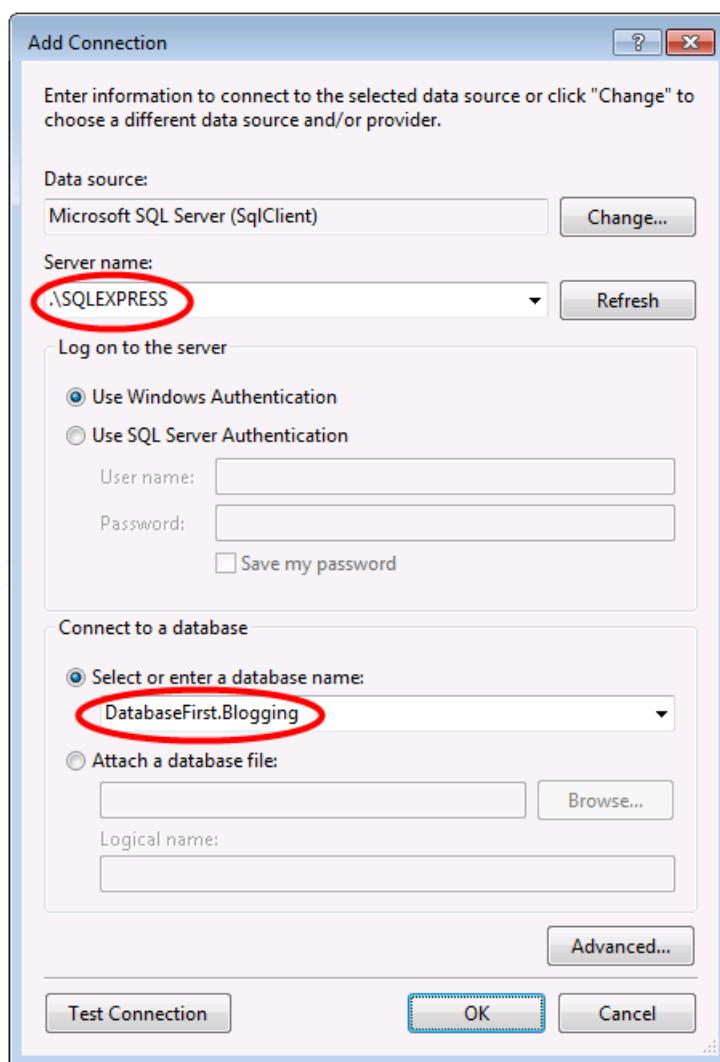
- Si vous utilisez Visual Studio 2010, vous allez créer une base de données SQL Express.
- Si vous utilisez Visual Studio 2012, vous allez créer une base de données de [base de données locale](#).

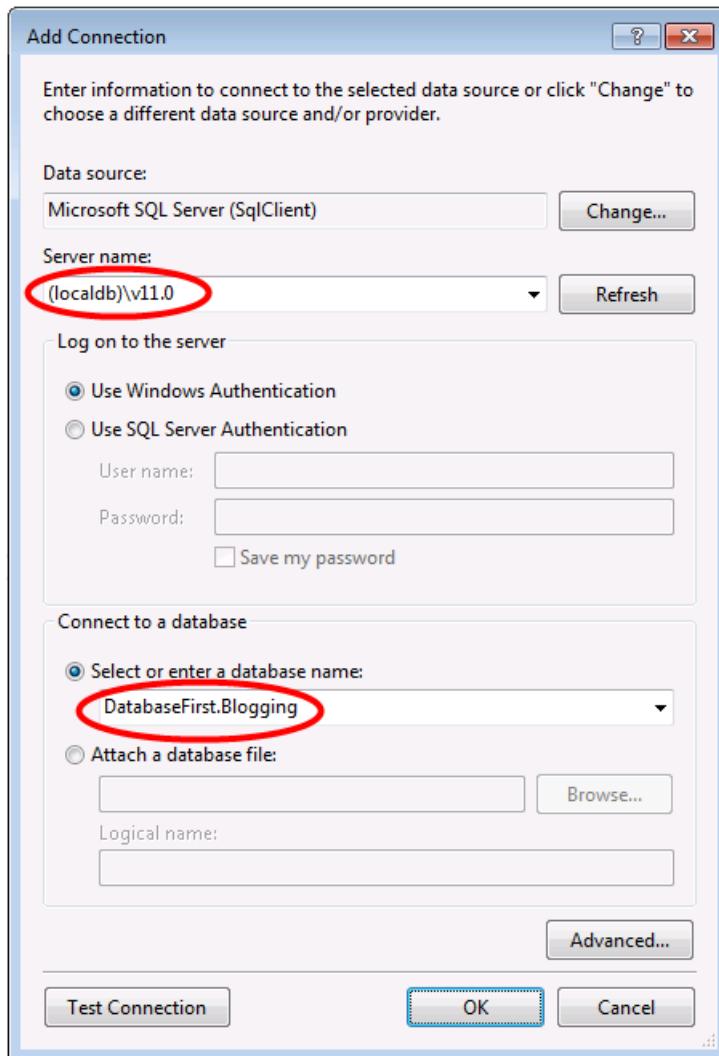
Commençons par générer la base de données.

- Ouvrez Visual Studio
- **Vue-> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **connexions de données-> ajouter une connexion...**
- Si vous n'êtes pas connecté à une base de données à partir de Explorateur de serveurs avant de devoir sélectionner Microsoft SQL Server comme source de données



- Connectez-vous à la base de données locale ou SQL Express, en fonction de celle que vous avez installée, puis entrez **DatabaseFirst.blog** comme nom de la base de données.





- Sélectionnez **OK**. vous serez invité à créer une nouvelle base de données, sélectionnez **Oui** .



- La nouvelle base de données s'affiche alors dans Explorateur de serveurs, cliquez dessus avec le bouton droit et sélectionnez **nouvelle requête** .
- Copiez le code SQL suivant dans la nouvelle requête, cliquez avec le bouton droit sur la requête et sélectionnez **exécuter** .

```

CREATE TABLE [dbo].[Blogs] (
    [BlogId] INT IDENTITY (1, 1) NOT NULL,
    [Name] NVARCHAR (200) NULL,
    [Url] NVARCHAR (200) NULL,
    CONSTRAINT [PK_dbo.Blogs] PRIMARY KEY CLUSTERED ([BlogId] ASC)
);

CREATE TABLE [dbo].[Posts] (
    [PostId] INT IDENTITY (1, 1) NOT NULL,
    [Title] NVARCHAR (200) NULL,
    [Content] NTEXT NULL,
    [BlogId] INT NOT NULL,
    CONSTRAINT [PK_dbo.Posts] PRIMARY KEY CLUSTERED ([PostId] ASC),
    CONSTRAINT [FK_dbo.Posts_dbo.Blogs_BlogId] FOREIGN KEY ([BlogId]) REFERENCES [dbo].[Blogs] ([BlogId]) ON
DELETE CASCADE
);

```

## 2. créer l'application

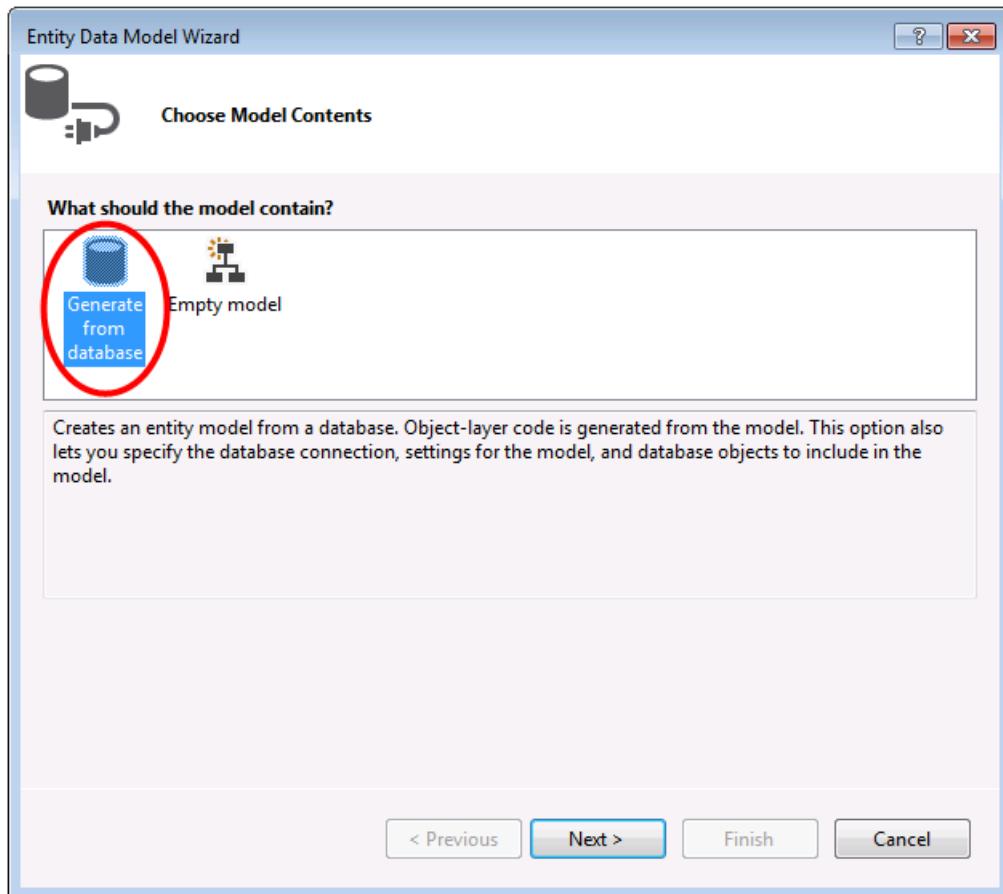
Pour simplifier les choses, nous allons créer une application console de base qui utilise le Database First pour effectuer l'accès aux données :

- Ouvrez Visual Studio
- **Fichier> nouveau>...**
- Sélectionnez **Windows** dans le menu de gauche et dans l'**application console** .
- Entrez **DatabaseFirstSample** comme nom
- Sélectionnez **OK**.

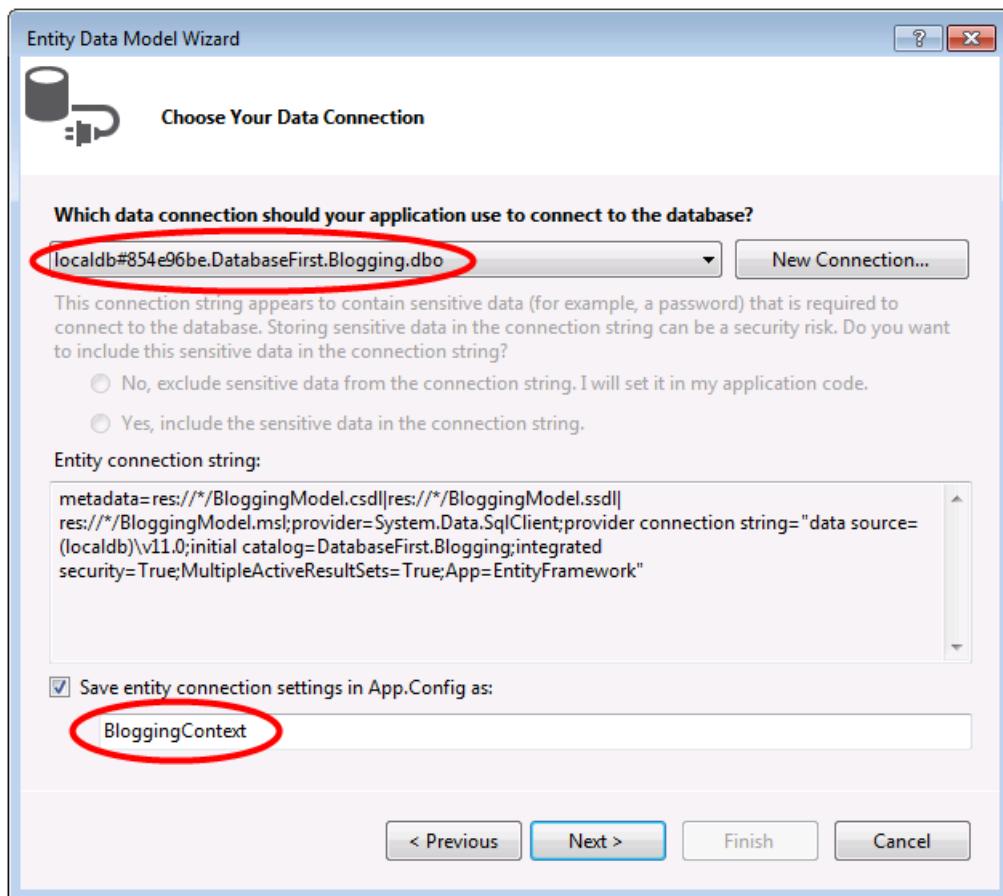
## 3. modèle d'ingénierie à rebours

Nous allons utiliser Entity Framework Designer, inclus dans le cadre de Visual Studio, pour créer notre modèle.

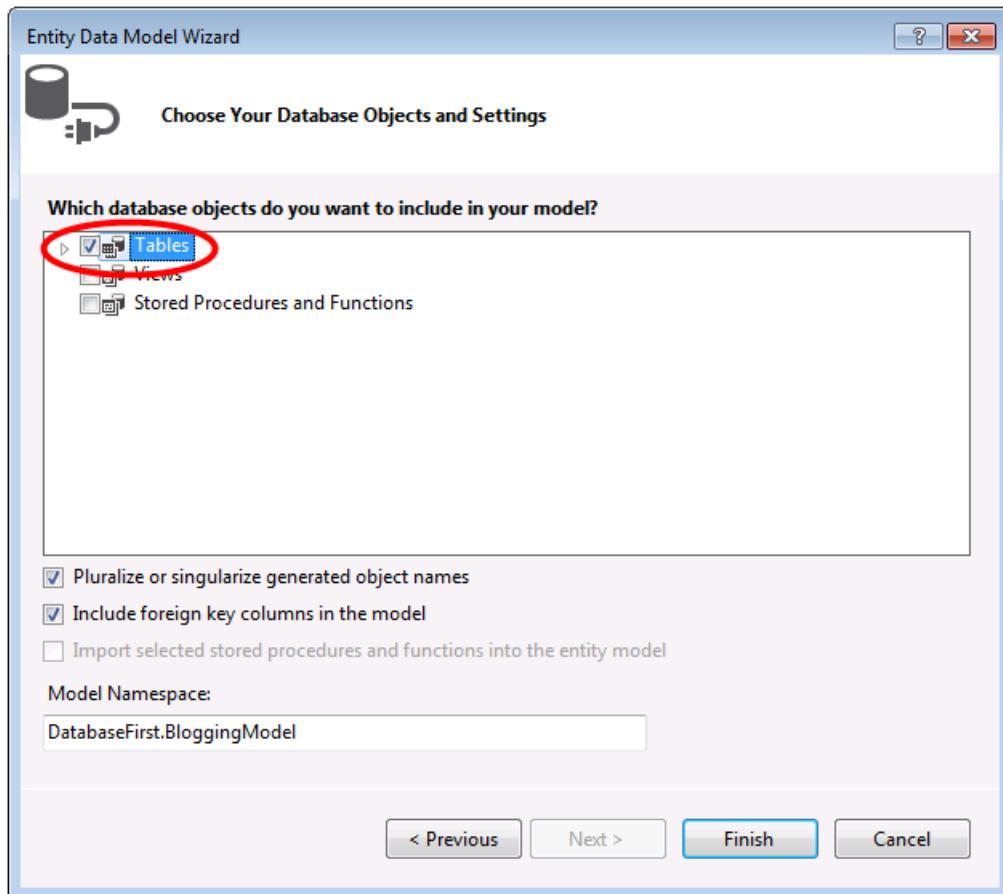
- **Projet-> ajouter un nouvel élément...**
- Sélectionnez **données** dans le menu de gauche, puis **ADO.NET Entity Data Model**
- Entrez **BloggingModel** comme nom et cliquez sur **OK** .
- Cela lance l'**assistant Entity Data Model**
- Sélectionnez **générer à partir de la base de données** , puis cliquez sur **suivant** .



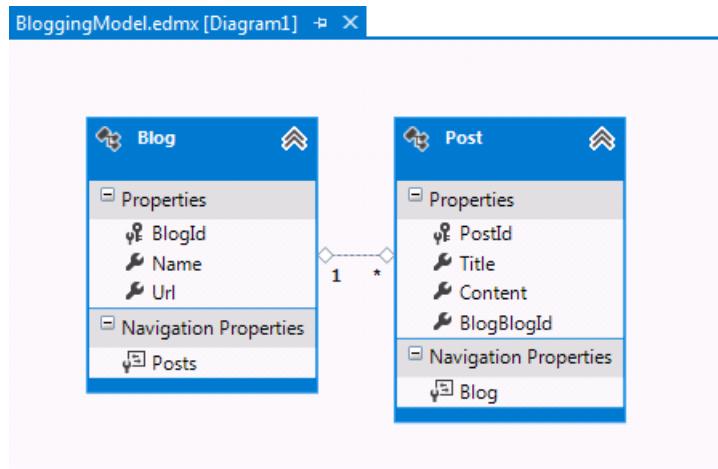
- Sélectionnez la connexion à la base de données que vous avez créée dans la première section, entrez **BloggingContext** comme nom de la chaîne de connexion, puis cliquez sur **suivant**.



- Cochez la case en regard de « tables » pour importer toutes les tables, puis cliquez sur « Terminer ».



Une fois le processus d'ingénierie à rebours terminé, le nouveau modèle est ajouté à votre projet et vous est ouvert pour que vous l'affichez dans le Entity Framework Designer. Un fichier app.config a également été ajouté à votre projet avec les détails de connexion de la base de données.



### Étapes supplémentaires dans Visual Studio 2010

Si vous travaillez dans Visual Studio 2010, vous devez suivre certaines étapes supplémentaires pour effectuer une mise à niveau vers la dernière version de Entity Framework. La mise à niveau est importante, car elle vous permet d'accéder à une surface d'API améliorée, qui est beaucoup plus facile à utiliser, ainsi que les derniers correctifs de bogues.

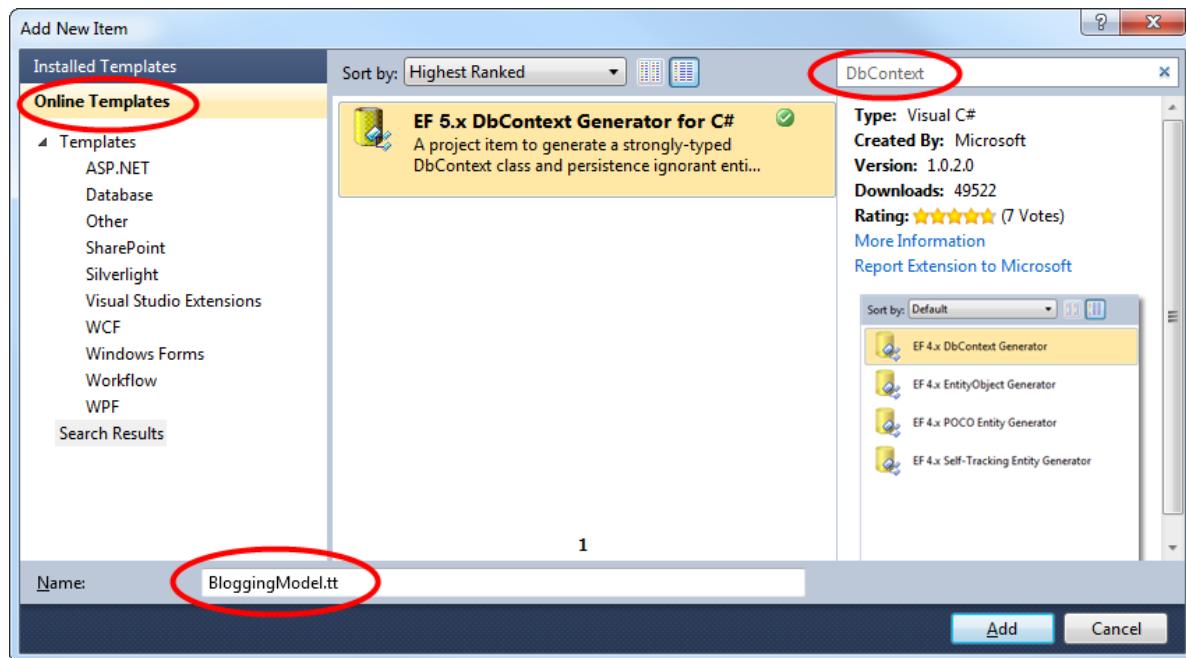
Tout d'abord, nous devons récupérer la dernière version de Entity Framework à partir de NuGet.

- **Projet-> gérer les packages NuGet...** \*si vous n'avez pas l'option **gérer les packages NuGet...** vous devez installer la [dernière version de NuGet](#) \*
- Sélectionner l'onglet **en ligne**
- Sélectionner le package **EntityFramework**

- Cliquez sur **installer**

Ensuite, nous devons permutez notre modèle pour générer le code qui utilise l'API DbContext, qui a été introduite dans les versions ultérieures de Entity Framework.

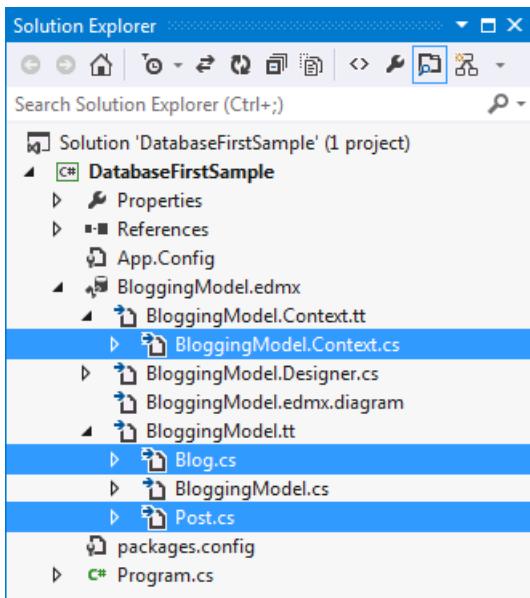
- Cliquez avec le bouton droit sur une zone vide de votre modèle dans le concepteur EF, puis sélectionnez **Ajouter un élément de génération de code...**
- Sélectionnez **modèles en ligne** dans le menu de gauche et recherchez **DbContext**
- Sélectionnez le **Générateur de DBCONTEXT EF 5. x pour C#**, entrez **BloggingModel** comme nom et cliquez sur **Ajouter**.



## 4. lecture & écriture de données

Maintenant que nous avons un modèle, il est temps de l'utiliser pour accéder à certaines données. Les classes que nous allons utiliser pour accéder aux données sont générées automatiquement pour vous en fonction du fichier EDMX.

*Cette capture d'écran provient de Visual Studio 2012, si vous utilisez Visual Studio 2010, les fichiers BloggingModel.tt et BloggingModel.Context.tt sont directement sous votre projet plutôt que imbriqués sous le fichier EDMX.*



Implémentez la méthode main dans Program.cs comme indiqué ci-dessous. Ce code crée une nouvelle instance de notre contexte, puis l'utilise pour insérer un nouveau blog. Il utilise ensuite une requête LINQ pour récupérer tous les blogs de la base de données classée par ordre alphabétique par titre.

```
class Program
{
    static void Main(string[] args)
    {
        using (var db = new BloggingContext())
        {
            // Create and save a new Blog
            Console.Write("Enter a name for a new Blog: ");
            var name = Console.ReadLine();

            var blog = new Blog { Name = name };
            db.Blogs.Add(blog);
            db.SaveChanges();

            // Display all Blogs from the database
            var query = from b in db.Blogs
                        orderby b.Name
                        select b;

            Console.WriteLine("All blogs in the database:");
            foreach (var item in query)
            {
                Console.WriteLine(item.Name);
            }

            Console.WriteLine("Press any key to exit...");
            Console.ReadKey();
        }
    }
}
```

Vous pouvez maintenant exécuter l'application et la tester.

```
Enter a name for a new Blog: ADO.NET Blog
All blogs in the database:
ADO.NET Blog
Press any key to exit...
```

## 5. traitement des modifications de la base de données

À présent, il est temps d'apporter des modifications à notre schéma de base de données, lorsque nous effectuons ces modifications, nous devons également mettre à jour notre modèle pour refléter ces modifications.

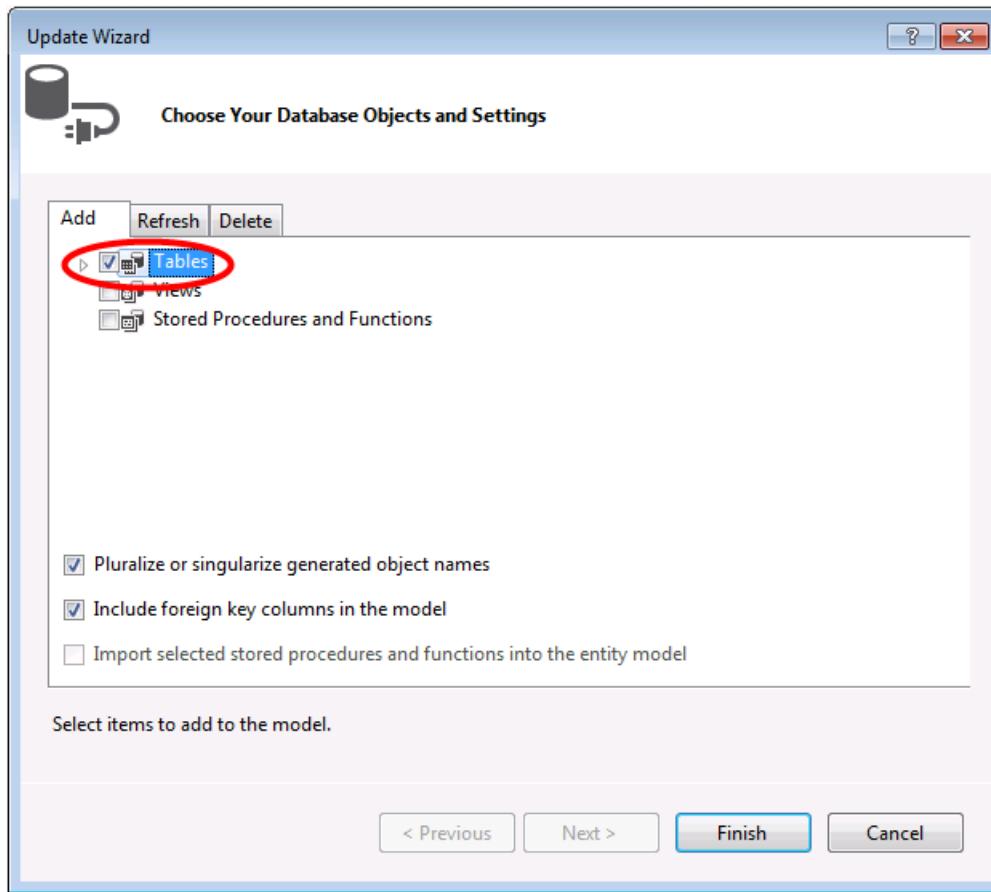
La première étape consiste à apporter des modifications au schéma de la base de données. Nous allons ajouter une table users au schéma.

- Cliquez avec le bouton droit sur la base de données **DatabaseFirst. blogs** dans Explorateur de serveurs puis sélectionnez **nouvelle requête**.
- Copiez le code SQL suivant dans la nouvelle requête, cliquez avec le bouton droit sur la requête et sélectionnez **exécuter**.

```
CREATE TABLE [dbo].[Users]
(
    [Username] NVARCHAR(50) NOT NULL PRIMARY KEY,
    [DisplayName] NVARCHAR(MAX) NULL
)
```

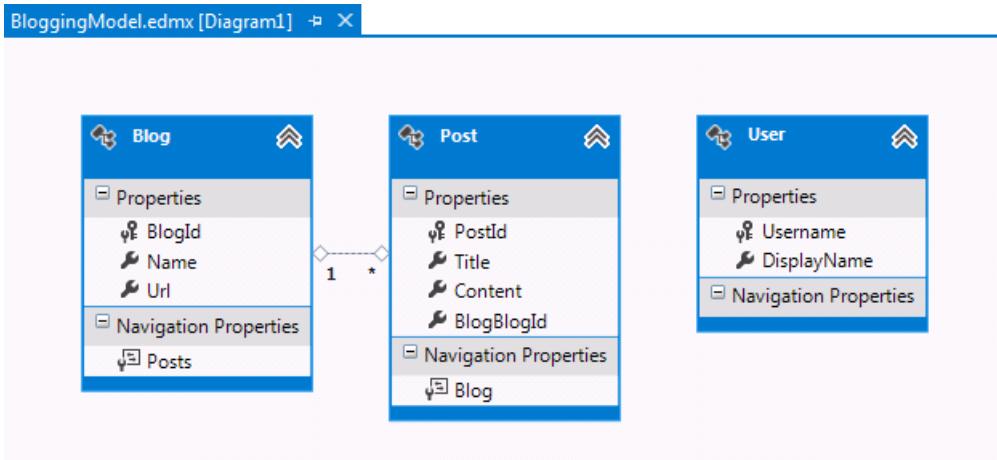
Maintenant que le schéma est mis à jour, il est temps de mettre à jour le modèle avec ces modifications.

- Cliquez avec le bouton droit sur une zone vide de votre modèle dans le concepteur EF et sélectionnez mettre à jour le modèle à partir de la base de données. cette opération lance l'Assistant Mise à jour.
- Dans l'onglet ajouter de l'Assistant Mise à jour, activez la case à cocher en regard de tables. cela indique que nous souhaitons ajouter de nouvelles tables à partir du schéma. *L'onglet actualiser affiche toutes les tables existantes dans le modèle dont les modifications sont recherchées pendant la mise à jour. Les onglets supprimer affichent toutes les tables qui ont été supprimées du schéma et sont également supprimées du modèle dans le cadre de la mise à jour. Les informations de ces deux onglets sont automatiquement détectées et sont fournies à titre d'information uniquement, vous ne pouvez pas modifier les paramètres.*



- Cliquez sur terminer dans l'Assistant Mise à jour

Le modèle est maintenant mis à jour pour inclure une nouvelle entité utilisateur qui est mappée à la table users que nous avons ajoutée à la base de données.



## Résumé

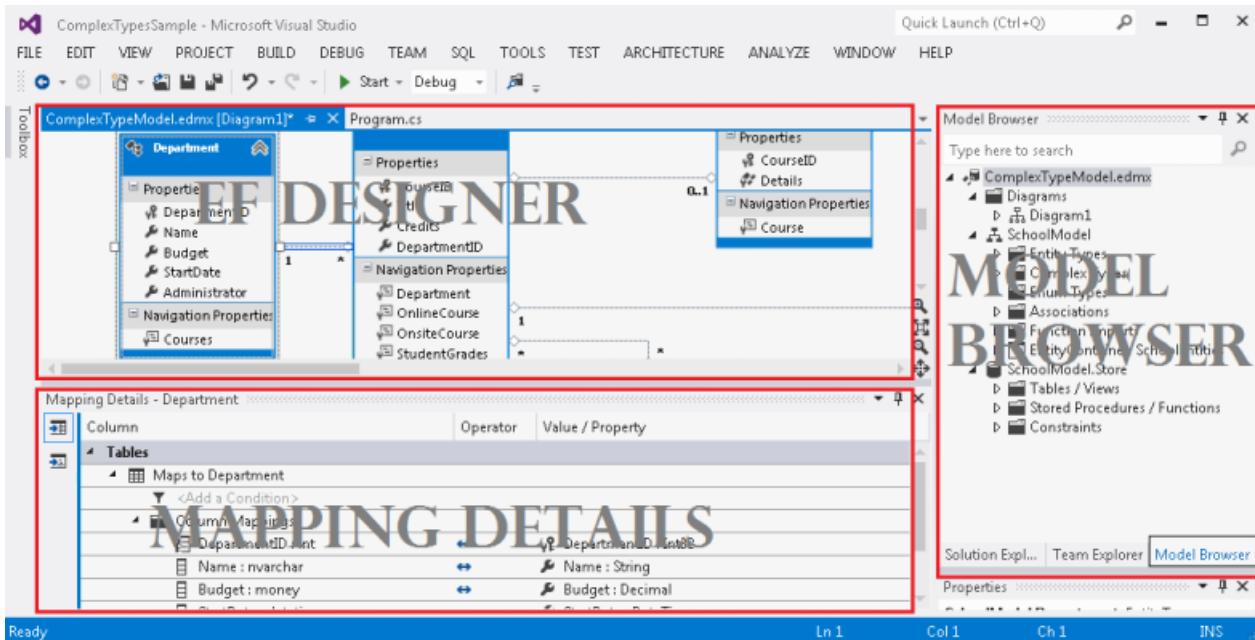
Dans cette procédure pas à pas, nous avons examiné Database First développement, ce qui nous a permis de créer un modèle dans le concepteur EF basé sur une base de données existante. Nous avons ensuite utilisé ce modèle pour lire et écrire des données de la base de données. Enfin, nous avons mis à jour le modèle pour refléter les modifications que nous avons apportées au schéma de base de données.

# Types complexes - Entity Framework Designer

13/09/2018 • 13 minutes to read

Cette rubrique montre comment mapper des types complexes avec Entity Framework Designer (Concepteur d'EF) et comment interroger des entités qui contiennent des propriétés de type complexe.

L'illustration suivante montre les principales fenêtres qui sont utilisées lorsque vous travaillez avec le Concepteur EF.



## NOTE

Lorsque vous générez le modèle conceptuel, les avertissements sur les entités non mappées et les associations peuvent apparaître dans la liste d'erreurs. Vous pouvez ignorer ces avertissements, car une fois que vous choisissez de générer la base de données à partir du modèle, les erreurs disparaîtront.

## Qu'est un Type complexe

Les types complexes sont les propriétés non scalaires des types d'entités qui permettent d'organiser les propriétés scalaires au sein des entités. À l'instar des entités, les types complexes regroupent des propriétés scalaires ou d'autres propriétés de type complexe.

Lorsque vous travaillez avec des objets qui représentent des types complexes, tenez compte des éléments suivants :

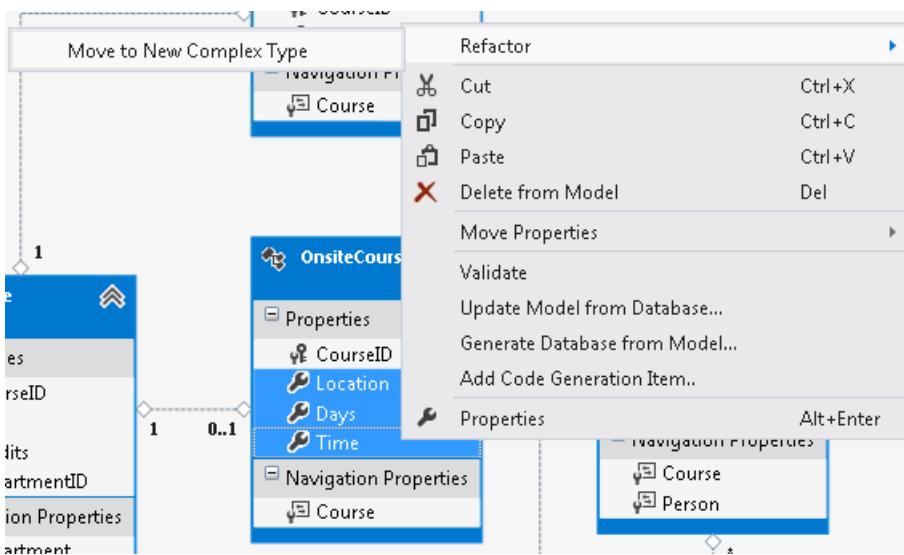
- Les types complexes n'ont pas de clés et par conséquent ne peut pas exister indépendamment. Les types complexes peuvent uniquement exister en tant que propriétés de types d'entités ou d'autres types complexes.
- Les types complexes ne peuvent pas participer aux associations et ne peut pas contenir les propriétés de navigation.
- Propriétés de type complexe ne peut pas être **null**. Un **InvalidOperationException** se produit lorsque **DbContext.SaveChanges** est appelée et un objet complexe null est rencontré. Les propriétés scalaires des objets complexes peuvent être **null**.
- Les types complexes ne peuvent pas hériter d'autres types complexes.

- Vous devez définir le type complexe en tant qu'un **classe**.
- Entity Framework détecte les modifications apportées aux membres sur un objet de type complexe lorsque **DbContext.DetectChanges** est appelée. Entity Framework appelle **DetectChanges** automatiquement lorsque les membres suivants sont appelées : **DbSet.Find**, **DbSet.Local**, **DbSet.Remove**, **DbSet.Add**, **DbSet.Attach**, **DbContext.SaveChanges**, **DbContext.GetValidationErrors**, **DbContext.Entry**, **DbContext.ChangeTracker**.Entries.

## Refactoriser des propriétés d'une entité en nouveau Type complexe

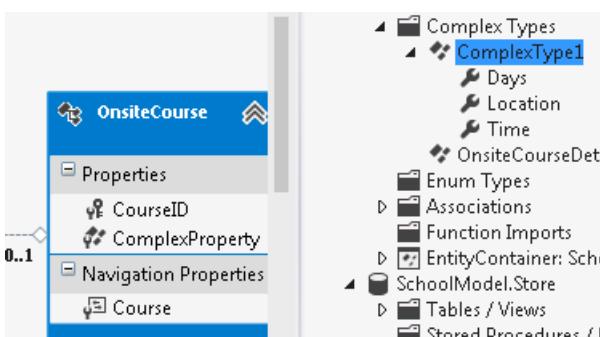
Si vous disposez déjà d'une entité dans votre modèle conceptuel, que vous souhaitez peut-être refactoriser une partie des propriétés dans une propriété de type complexe.

Sur l'aire du concepteur, sélectionnez une ou plusieurs propriétés (à l'exclusion des propriétés de navigation) d'une entité, puis avec le bouton droit et sélectionnez **refactoriser -> déplacer vers le nouveau Type complexe**.



Un nouveau type complexe avec les propriétés sélectionnées est ajouté à la **Explorateur de modèles**. Un nom par défaut est affecté au type complexe.

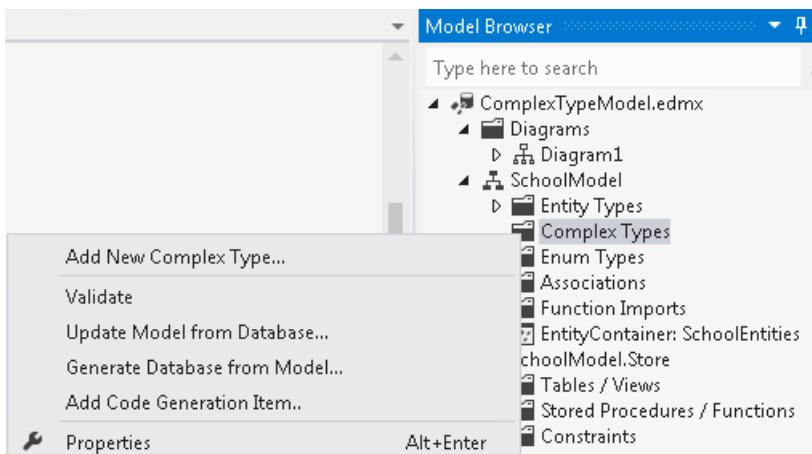
Une propriété complexe du type récemment créé remplace les propriétés sélectionnées. Tous les mappages de propriété sont conservés.



## Créer un nouveau Type complexe

Vous pouvez également créer un nouveau type complexe qui ne contient pas de propriétés d'une entité existante.

Avec le bouton droit le **des Types complexes** dossier dans l'Explorateur de modèles, pointez sur **Type complexe AddNew...**. Vous pouvez également sélectionner le **des Types complexes** dossier et appuyez sur la **insérer** clé de votre clavier.



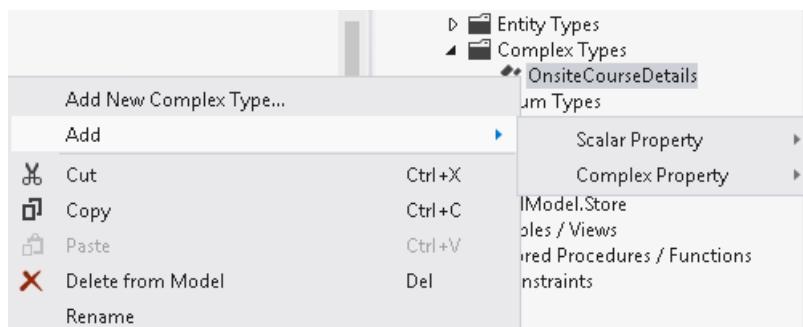
Un nouveau type complexe est ajouté au dossier avec un nom par défaut. Vous pouvez maintenant ajouter des propriétés pour le type.

## Ajouter des propriétés à un Type complexe

Les propriétés d'un type complexe peuvent être des types scalaires ou des types complexes existants. Toutefois, les propriétés de type complexe ne peuvent pas avoir de références circulaires. Par exemple, un type complexe **OnsiteCourseDetails** ne peut pas avoir une propriété de type complexe **OnsiteCourseDetails**.

Vous pouvez ajouter une propriété à un type complexe en appliquant l'une des méthodes répertoriées ci-dessous.

- Avec le bouton droit à un type complexe dans l'Explorateur de modèles, pointez sur **ajouter**, puis pointez sur **propriété scalaire** ou **propriété complexe**, puis sélectionnez le type de propriété souhaitée. Ou bien, vous pouvez sélectionner un type complexe et appuyez sur la **insérer** clé de votre clavier.



Une nouvelle propriété est ajoutée au type complexe avec un nom par défaut.

- OR :
- Avec le bouton droit d'une propriété d'entité sur le **Entity Framework Designer** aire de conception et sélectionnez **copie**, puis cliquez sur le type complexe dans le **Explorateur de modèles** et sélectionnez **Coller**.

## Renommer un Type complexe

Lorsque vous renommez un type complexe, toutes les références au type sont mises à jour dans tout le projet.

- Double-cliquez lentement sur un type complexe dans le **Explorateur de modèles**. Le nom est sélectionné et passe en mode édition.
- OR :
- Avec le bouton droit à un type complexe dans le **Explorateur de modèles** et sélectionnez **renommer**.
- OR :

- Sélectionnez un type complexe dans l'Explorateur de modèles, puis appuyez sur la touche F2.
- OR :
- Avec le bouton droit à un type complexe dans le **Explorateur de modèles** et sélectionnez **propriétés**. Modifier le nom dans la **propriétés** fenêtre.

## Ajouter un Type complexe existant à une entité et ses propriétés de la carte aux colonnes de Table

1. Cliquez sur une entité, pointez sur **Ajouter nouveau**, puis sélectionnez **propriété complexe**. Une propriété de type complexe avec un nom par défaut est ajoutée à l'entité. Un type par défaut (sélectionné dans les types complexes existants) est affecté à la propriété.
2. Affectez le type désiré à la propriété dans le **propriétés** fenêtre. Après avoir ajouté une propriété de type complexe à une entité, vous devez mapper ses propriétés aux colonnes de la table.
3. Avec le bouton droit sur l'aire de conception ou dans un type d'entité le**Explorateur de modèles** et sélectionnez **mappages de Table**. Les mappages de table s'affichent dans le **détails de Mapping** fenêtre.
4. Développez le **est mappé à <nom de la Table>** nœud. Un **mappages de colonnes** nœud s'affiche.
5. Développez le **mappages de colonnes** nœud. Une liste de toutes les colonnes de la table s'affiche. Les propriétés par défaut (le cas échéant) à laquelle mappent les colonnes sont répertoriés sous la **valeur/propriété** titre.
6. Sélectionnez la colonne que vous souhaitez mapper et avec le bouton droit puis correspondant **valeur/propriété** champ. Une liste déroulante de toutes les propriétés scalaires s'affiche.
7. Sélectionnez la propriété appropriée.

| Column               | Operator          | Value / Property          |
|----------------------|-------------------|---------------------------|
| <b>Tables</b>        |                   |                           |
| Maps to OnsiteCourse | <Add a Condition> |                           |
| Column Mappings      |                   |                           |
| CourseID : int       | ↔                 | CourseID : Int32          |
| Location : nvarchar  | ↔                 | Details.Location : String |
| Days : nvarchar      | ↔                 | Details.Days : String     |
| Time : smalldatetime | ↔                 | Details.Time : DateTime   |

8. Répétez les étapes 6 et 7 pour chaque colonne de la table.

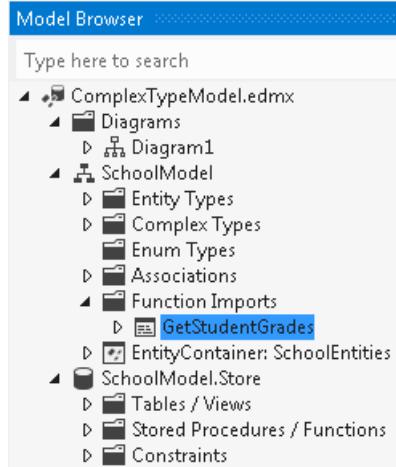
### NOTE

Pour supprimer un mappage de colonnes, sélectionnez la colonne que vous souhaitez mapper, puis cliquez sur le **valeur/propriété** champ. Ensuite, sélectionnez **supprimer** dans la liste déroulante.

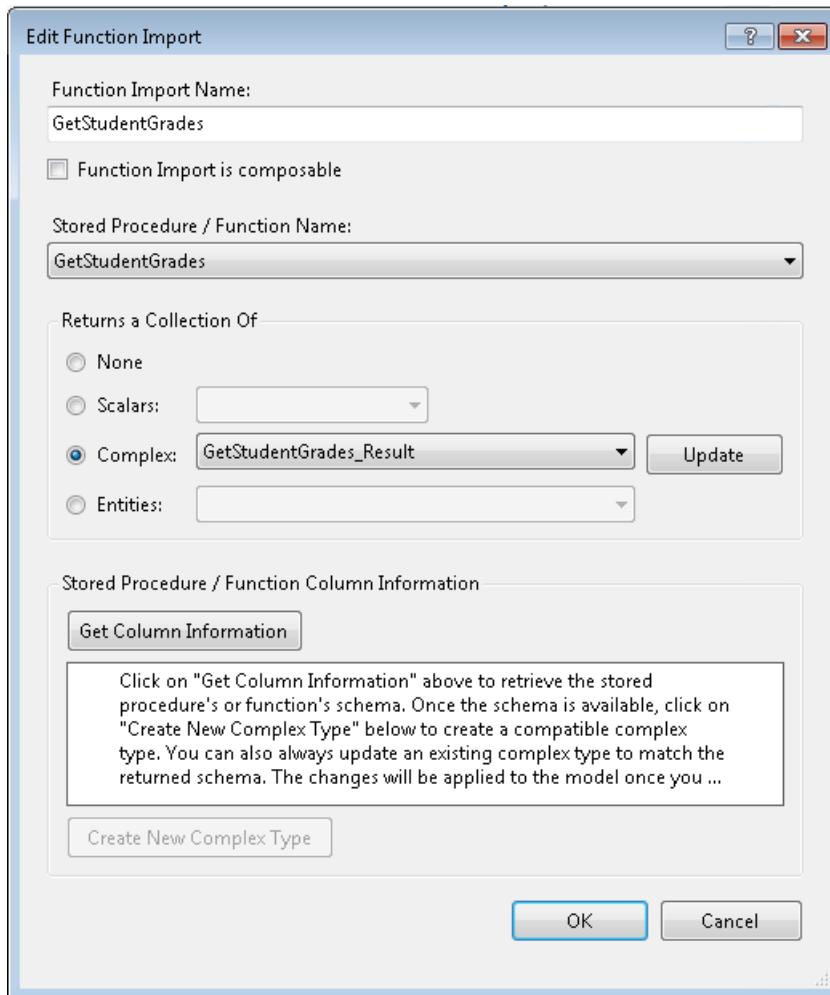
## Mapper une importation de fonction à un Type complexe

Les importations de fonction sont basées sur les procédures stockées. Pour mapper une importation de fonction à un type complexe, les colonnes renvoyées par la procédure stockée correspondante doivent correspondre aux propriétés du type complexe en nombre et doivent avoir des types de stockage qui sont compatibles avec les types de propriété.

- Double-cliquez sur une fonction importée que vous souhaitez mapper à un type complexe.



- Définissez les paramètres de la nouvelle importation de fonction comme suit :
  - Spécifiez la procédure stockée pour laquelle vous créez une importation de fonction dans le **nom de la procédure stockée** champ. Ce champ est une liste déroulante qui affiche toutes les procédures stockées dans le modèle de stockage.
  - Spécifiez le nom de l'importation de fonction dans le **nom du Function Import** champ.
  - Sélectionnez **complexes** en tant que la valeur de retour tapez, puis spécifiez le type de retour complex spécifique en choisissant le type approprié dans la liste déroulante.



- Cliquez sur **OK**. L'entrée function import est créée dans le modèle conceptuel.

#### Personnaliser le mappage pour l'importation de fonction de colonnes

- Cliquez sur l'importation de fonction dans l'Explorateur de modèles et sélectionnez **mappage d'importation de fonction**. La **détails de Mapping** fenêtre apparaît et affiche le mappage par défaut pour l'importation de fonction. Les flèches indiquent les mappages entre valeurs de colonne et valeurs de propriété. Par défaut, les noms de colonne sont supposés être les mêmes que ceux de la propriété du type complexe. Les noms de colonne par défaut s'affichent en gris.
- Si nécessaire, changez les noms des colonnes pour qu'ils correspondent aux noms des colonnes rentrés par la procédure stockée correspondant à l'importation de fonction.

## Supprimer un Type complexe

Lorsque vous supprimez un type complexe, le type est supprimé du modèle conceptuel, et les mappages pour toutes les instances du type sont supprimés. Toutefois, les références au type ne sont pas mises à jour. Par exemple, si une entité a une propriété de type complexe de type ComplexType1 et que ComplexType1 est supprimé dans le **Explorateur de modèles**, la propriété d'entité correspondante n'est pas mis à jour. Le modèle ne sera pas validé, car elle contient une entité qui fait référence à un type complexe supprimé. Vous pouvez mettre à jour ou supprimer des références à des types complexes supprimés à l'aide du Concepteur d'entités.

- Cliquez sur un type complexe dans l'Explorateur de modèles et sélectionnez **supprimer**.
- OR :
- Sélectionnez un type complexe dans l'Explorateur de modèles, puis appuyez sur la touche Suppr de votre clavier.

## Requête pour les entités qui contient les propriétés de Type complexe

Le code suivant montre comment exécuter une requête qui retourne une collection d'entités des objets de type qui contiennent une propriété de type complexe.

```
using (SchoolEntities context = new SchoolEntities())
{
    var courses =
        from c in context.OnsiteCourses
        order by c.Details.Time
        select c;

    foreach (var c in courses)
    {
        Console.WriteLine("Time: " + c.Details.Time);
        Console.WriteLine("Days: " + c.Details.Days);
        Console.WriteLine("Location: " + c.Details.Location);
    }
}
```

# Prise en charge d'enum-concepteur EF

23/11/2019 • 11 minutes to read

## NOTE

**EF5 uniquement** : les fonctionnalités, les API, etc. présentées dans cette page ont été introduites dans Entity Framework 5. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Cette vidéo et la procédure pas à pas montrent comment utiliser les types ENUM avec l'Entity Framework Designer. Il montre également comment utiliser des enums dans une requête LINQ.

Cette procédure pas à pas utilise Model First pour créer une base de données, mais le concepteur EF peut également être utilisé avec le flux de travail [Database First](#) pour mapper à une base de données existante.

La prise en charge des énumérations a été introduite dans Entity Framework 5. Pour utiliser les nouvelles fonctionnalités telles que les énumérations, les types de données spatiales et les fonctions table, vous devez cibler .NET Framework 4,5. Visual Studio 2012 cible .NET 4,5 par défaut.

Dans Entity Framework, une énumération peut avoir les types sous-jacents suivants : **Byte**, **Int16**, **Int32**, **Int64** ou **SByte**.

## Regarder la vidéo

Cette vidéo montre comment utiliser les types ENUM avec l'Entity Framework Designer. Il montre également comment utiliser des enums dans une requête LINQ.

**Présenté par:** Julia Kornich

**Vidéo:** [wmv](#) | [MP4](#) | [WMV \(zip\)](#)

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous devez avoir installé Visual Studio 2012, Ultimate, Premium, Professional ou Web Express Edition.

## Configurer le projet

1. Ouvrir Visual Studio 2012
2. Dans le menu **fichier**, pointez sur **nouveau**, puis cliquez sur **projet** .
3. Dans le volet gauche, cliquez sur **Visual C#** , puis sélectionnez le modèle **console** .
4. Entrez **EnumEFDesigner** comme nom du projet, puis cliquez sur **OK** .

## Créer un nouveau modèle à l'aide du concepteur EF

1. Cliquez avec le bouton droit sur le nom du projet dans Explorateur de solutions, pointez sur **Ajouter**, puis cliquez sur **nouvel élément** .
2. Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet modèles.
3. Entrez **EnumTestModel.edmx** comme nom de fichier, puis cliquez sur **Ajouter** .
4. Dans la page Entity Data Model de l'Assistant, sélectionnez **modèle vide** dans la boîte de dialogue choisir le

contenu du modèle.

## 5. Cliquez sur **Terminer**

Le Entity Designer, qui fournit une aire de conception pour la modification de votre modèle, est affiché.

L'assistant exécute les actions suivantes :

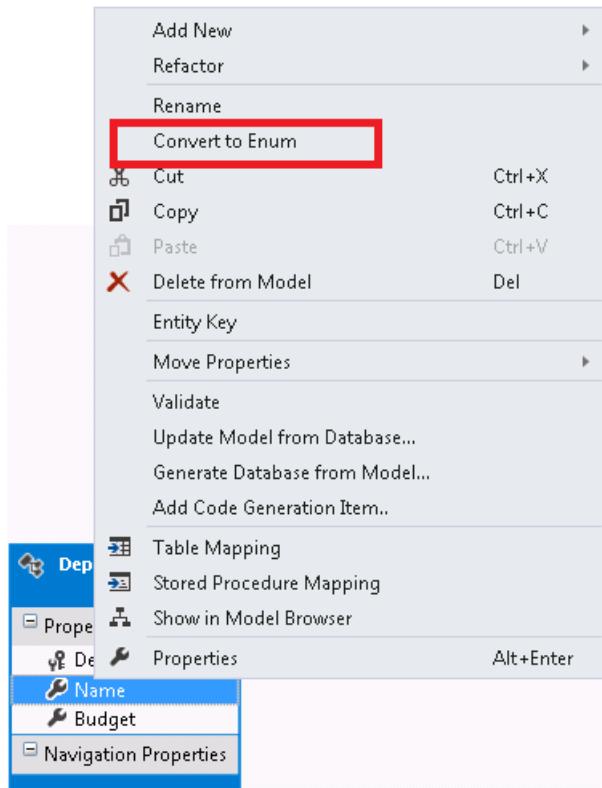
- Génère le fichier EnumTestModel.edmx qui définit le modèle conceptuel, le modèle de stockage et le mappage entre eux. Définit la propriété de traitement des artefacts de métadonnées du fichier.edmx à incorporer dans l'assembly de sortie afin que les fichiers de métadonnées générés soient incorporés dans l'assembly.
- Ajoute une référence aux assemblies suivants : EntityFramework, System. ComponentModel. DataAnnotations et System. Data. Entity.
- Crée des fichiers EnumTestModel.tt et EnumTestModel.Context.tt et les ajoute sous le fichier. edmx. Ces fichiers de modèle T4 génèrent le code qui définit le type dérivé DbContext et les types POCO qui mappent aux entités dans le modèle. edmx.

## Ajouter un nouveau type d'entité

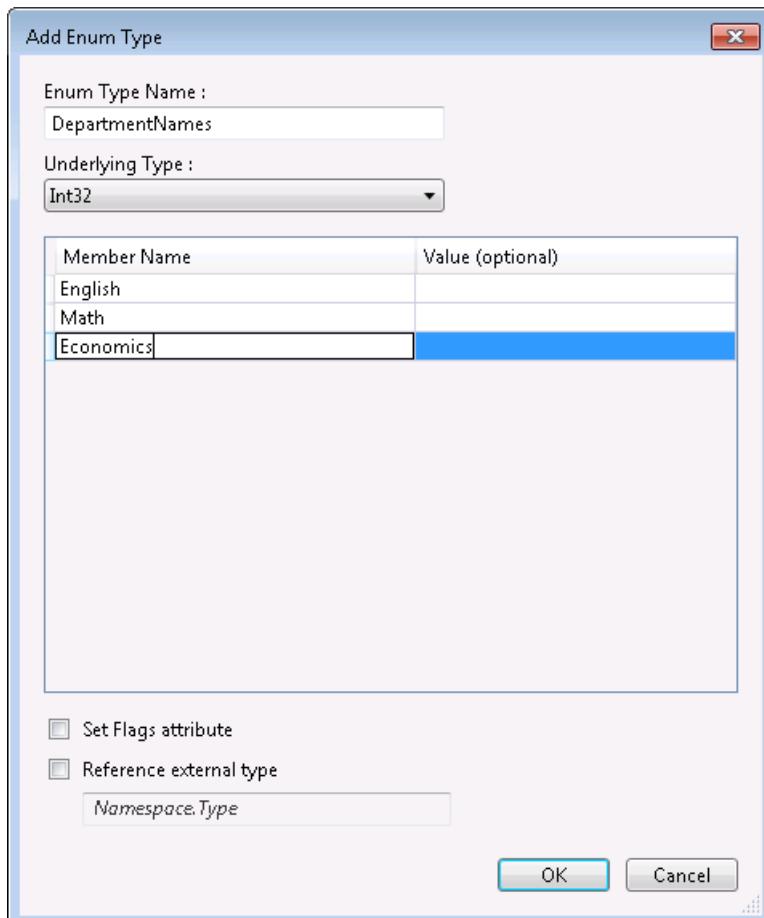
1. Cliquez avec le bouton droit sur une zone vide de l'aire de conception, sélectionnez **Ajouter-> entité**, la boîte de dialogue nouvelle entité s'affiche.
2. Spécifiez **Department** comme nom de type et spécifiez **DepartmentID** pour le nom de la propriété de clé, laissez le type **Int32**
3. Cliquez sur **OK**.
4. Cliquez avec le bouton droit sur l'entité, puis sélectionnez **Ajouter une nouvelle> propriété scalaire**
5. Renommez la nouvelle propriété en **Name**
6. Remplacez le type de la nouvelle propriété par **Int32** (par défaut, la nouvelle propriété est de type chaîne) pour modifier le type, ouvrez le fenêtre Propriétés et remplacez la propriété type par **Int32** .
7. Ajoutez une autre propriété scalaire et renommez-la **budget**, modifiez le type en **Decimal**

## Ajouter un type enum

1. Dans la Entity Framework Designer, cliquez avec le bouton droit sur la propriété Name, sélectionnez **convertir en enum** .



2. Dans la boîte de dialogue **Ajouter un enum**, tapez **DepartmentNames** pour le nom du type d'énumération, remplacez le type sous-jacent par **Int32**, puis ajoutez les membres suivants au type : anglais, mathématiques et économie



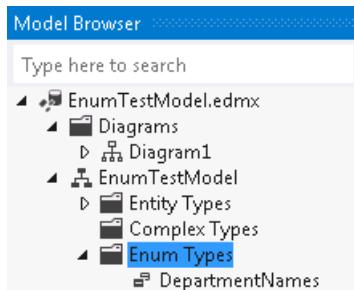
3. Appuyez sur **OK**
4. Enregistrer le modèle et générer le projet

#### NOTE

Lorsque vous générez, les avertissements relatifs aux entités et associations non mappés peuvent apparaître dans la Liste d'erreurs. Vous pouvez ignorer ces avertissements, car une fois que vous avez choisi de générer la base de données à partir du modèle, les erreurs disparaissent.

Si vous examinez le Fenêtre Propriétés, vous remarquerez que le type de la propriété Name a été remplacé par **DepartmentNames** et que le type enum récemment ajouté a été ajouté à la liste des types.

Si vous basculez vers la fenêtre Explorateur de modèles, vous verrez que le type a été également ajouté au nœud types d'énumération.



#### NOTE

Vous pouvez également ajouter de nouveaux types d'énumération à partir de cette fenêtre en cliquant sur le bouton droit de la souris et en sélectionnant **Ajouter un type d'énumération**. Une fois le type créé, il apparaît dans la liste des types et vous pouvez l'associer à une propriété.

## Générer la base de données à partir du modèle

Nous pouvons maintenant générer une base de données basée sur le modèle.

1. Cliquez avec le bouton droit sur un espace vide sur l'aire de Entity Designer et sélectionnez **générer la base de données à partir du modèle**.
2. La boîte de dialogue choisir votre connexion de données de l'Assistant Création d'une base de données s'affiche. cliquez sur le bouton **nouvelle connexion**, spécifiez (base de données locale )\mssqllocaldb pour le nom du serveur et **EnumTest** pour la base de données, puis cliquez sur **OK**.
3. Une boîte de dialogue vous demandant si vous souhaitez créer une nouvelle base de données s'affiche, cliquez sur **Oui**.
4. Cliquez sur **suivant** pour que l'Assistant Création d'une base de données génère le langage de définition de données (DDL) pour la création d'une base de données. la DDL générée est affichée dans la boîte de dialogue Résumé et paramètres. Notez que le DDL ne contient pas de définition pour une table qui correspond au type d'énumération.
5. Cliquez sur **Terminer** pour ne pas exécuter le script DDL.
6. L'Assistant Création d'une base de données effectue les opérations suivantes : ouvre **EnumTest. edmx. SQL** dans l'éditeur T-SQL génère les sections de schéma de stockage et de mappage du fichier edmx ajoute les informations de chaîne de connexion au fichier app. config.
7. Cliquez sur le bouton droit de la souris dans l'éditeur T-SQL et sélectionnez **exécuter** la boîte de dialogue se connecter au serveur, entrez les informations de connexion de l'étape 2, puis cliquez sur **se connecter**.
8. Pour afficher le schéma généré, cliquez avec le bouton droit sur le nom de la base de données dans Explorateur d'objets SQL Server puis sélectionnez **Actualiser**.

## Conserver et récupérer des données

Ouvrez le fichier Program.cs dans lequel la méthode main est définie. Ajoutez le code suivant à la fonction main. Le code ajoute un nouvel objet Department au contexte. Il enregistre ensuite les données. Le code exécute également une requête LINQ qui retourne un service dont le nom est DepartmentNames. English.

```
using (var context = new EnumTestModelContainer())
{
    context.Departments.Add(new Department{ Name = DepartmentNames.English });

    context.SaveChanges();

    var department = (from d in context.Departments
                      where d.Name == DepartmentNames.English
                      select d).FirstOrDefault();

    Console.WriteLine(
        "DepartmentID: {0} and Name: {1}",
        department.DepartmentID,
        department.Name);
}
```

Compilez et exécutez l'application. Le programme génère la sortie suivante :

```
DepartmentID: 1 Name: English
```

Pour afficher les données de la base de données, cliquez avec le bouton droit sur le nom de la base de données dans Explorateur d'objets SQL Server, puis sélectionnez **Actualiser**. Ensuite, cliquez sur le bouton droit de la souris sur la table et sélectionnez **afficher les données**.

## Résumé

Dans cette procédure pas à pas, nous avons vu comment mapper des types énumération à l'aide de la Entity Framework Designer et comment utiliser des enums dans le code.

# Concepteur spatial-EF

23/11/2019 • 10 minutes to read

## NOTE

**EF5 uniquement** : les fonctionnalités, les API, etc. présentées dans cette page ont été introduites dans Entity Framework 5. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

La vidéo et la procédure pas à pas montrent comment mapper des types spatiaux avec l'Entity Framework Designer. Il montre également comment utiliser une requête LINQ pour rechercher une distance entre deux emplacements.

Cette procédure pas à pas utilise Model First pour créer une base de données, mais le concepteur EF peut également être utilisé avec le flux de travail [Database First](#) pour mapper à une base de données existante.

La prise en charge des types spatiaux a été introduite dans Entity Framework 5. Notez que pour utiliser les nouvelles fonctionnalités telles que le type spatial, les énumérations et les fonctions table, vous devez cibler .NET Framework 4,5. Visual Studio 2012 cible .NET 4,5 par défaut.

Pour utiliser des types de données spatiales, vous devez également utiliser un fournisseur de Entity Framework qui a une prise en charge spatiale. Pour plus d'informations, consultez [prise en charge des fournisseurs pour les types spatiaux](#).

Il existe deux types de données spatiales principales : Geography et Geometry. Le type de données geography stocke des données ellipsoïdal (par exemple, des coordonnées de latitude et de longitude GPS). Le type de données geometry représente le système de coordonnées euclidienne (Flat).

## Regarder la vidéo

Cette vidéo montre comment mapper des types spatiaux avec l'Entity Framework Designer. Il montre également comment utiliser une requête LINQ pour rechercher une distance entre deux emplacements.

**Présenté par:** Julia Kornich

**Vidéo:** [wmv](#) | [MP4](#) | [WMV \(zip\)](#)

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous devez avoir installé Visual Studio 2012, Ultimate, Premium, Professional ou Web Express Edition.

## Configurer le projet

1. Ouvrir Visual Studio 2012
2. Dans le menu **fichier**, pointez sur **nouveau**, puis cliquez sur **projet**.
3. Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le modèle **console**.
4. Entrez **SpatialEFDesigner** comme nom du projet, puis cliquez sur **OK**.

## Créer un nouveau modèle à l'aide du concepteur EF

1. Cliquez avec le bouton droit sur le nom du projet dans Explorateur de solutions, pointez sur **Ajouter**, puis

- cliquez sur **nouvel élément**.
2. Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet modèles.
  3. Entrez **UniversityModel.edmx** comme nom de fichier, puis cliquez sur **Ajouter**.
  4. Dans la page Entity Data Model de l'Assistant, sélectionnez **modèle vide** dans la boîte de dialogue choisir le contenu du modèle.
  5. Cliquez sur **Terminer**

Le Entity Designer, qui fournit une aire de conception pour la modification de votre modèle, est affiché.

L'assistant exécute les actions suivantes :

- Génère le fichier EnumTestModel.edmx qui définit le modèle conceptuel, le modèle de stockage et le mappage entre eux. Définit la propriété de traitement des artefacts de métadonnées du fichier. edmx à incorporer dans l'assembly de sortie afin que les fichiers de métadonnées générés soient incorporés dans l'assembly.
- Ajoute une référence aux assemblies suivants : EntityFramework, System. ComponentModel. DataAnnotations et System. Data. Entity.
- Crée des fichiers UniversityModel.tt et UniversityModel.Context.tt et les ajoute sous le fichier. edmx. Ces fichiers de modèle T4 génèrent le code qui définit le type dérivé DbContext et les types POCO qui mappent aux entités dans le modèle. edmx

## Ajouter un nouveau type d'entité

1. Cliquez avec le bouton droit sur une zone vide de l'aire de conception, sélectionnez **Ajouter-> entité**, la boîte de dialogue nouvelle entité s'affiche.
2. Spécifiez **University** comme nom de type et spécifiez **UniversityID** pour le nom de la propriété de clé, laissez le type **Int32**
3. Cliquez sur **OK**.
4. Cliquez avec le bouton droit sur l'entité, puis sélectionnez **Ajouter une nouvelle> propriété scalaire**
5. Renommez la nouvelle propriété en **Name**
6. Ajoutez une autre propriété scalaire et renommez-la **emplacement** ouvrir le fenêtre Propriétés et remplacez le type de la nouvelle propriété par **Geography**
7. Enregistrer le modèle et générer le projet

### NOTE

Lorsque vous générez, les avertissements relatifs aux entités et associations non mappés peuvent apparaître dans la Liste d'erreurs. Vous pouvez ignorer ces avertissements, car une fois que vous avez choisi de générer la base de données à partir du modèle, les erreurs disparaissent.

## Générer la base de données à partir du modèle

Nous pouvons maintenant générer une base de données basée sur le modèle.

1. Cliquez avec le bouton droit sur un espace vide sur l'aire de Entity Designer et sélectionnez **générer la base de données à partir du modèle**.
2. La boîte de dialogue choisir votre connexion de données de l'Assistant Création d'une base de données s'affiche. cliquez sur le bouton **nouvelle connexion**, spécifiez (base de données locale )\mssqllocaldb pour le nom du serveur et l' **Université** pour la base de données, puis cliquez sur **OK**.
3. Une boîte de dialogue vous demandant si vous souhaitez créer une nouvelle base de données s'affiche, cliquez sur **Oui**.
4. Cliquez sur **suivant** pour que l'Assistant Création d'une base de données génère le langage de définition de

données (DDL) pour la création d'une base de données. la DDL générée est affichée dans la boîte de dialogue Résumé et paramètres. Notez que le DDL ne contient pas de définition pour une table qui correspond au type d'énumération.

5. Cliquez sur **Terminer** pour ne pas exécuter le script DDL.
6. L'Assistant Création d'une base de données effectue les opérations suivantes : ouvre **UniversityModel.edmx**. **SQL** dans l'éditeur T-SQL génère les sections de schéma de stockage et de mappage du fichier edmx ajoute les informations de chaîne de connexion au fichier app.config.
7. Cliquez sur le bouton droit de la souris dans l'éditeur T-SQL et sélectionnez **exécuter** la boîte de dialogue se connecter au serveur, entrez les informations de connexion de l'étape 2, puis cliquez sur **se connecter**.
8. Pour afficher le schéma généré, cliquez avec le bouton droit sur le nom de la base de données dans Explorateur d'objets SQL Server puis sélectionnez **Actualiser**.

## Conserver et récupérer des données

Ouvrez le fichier Program.cs dans lequel la méthode main est définie. Ajoutez le code suivant à la fonction main.

Le code ajoute deux nouveaux objets University au contexte. Les propriétés spatiales sont initialisées à l'aide de la méthode DbGeography. FromText. Le point géographique représenté en tant que WellKnownText est passé à la méthode. Le code enregistre ensuite les données. Ensuite, la requête LINQ qui retourne un objet University dans lequel son emplacement est le plus proche de l'emplacement spécifié est construite et exécutée.

```
using (var context = new UniversityModelContainer())
{
    context.Universities.Add(new University()
    {
        Name = "Graphic Design Institute",
        Location = DbGeography.FromText("POINT(-122.336106 47.605049)"),
    });

    context.Universities.Add(new University()
    {
        Name = "School of Fine Art",
        Location = DbGeography.FromText("POINT(-122.335197 47.646711)"),
    });

    context.SaveChanges();

    var myLocation = DbGeography.FromText("POINT(-122.296623 47.640405)");

    var university = (from u in context.Universities
                      orderby u.Location.Distance(myLocation)
                      select u).FirstOrDefault();

    Console.WriteLine(
        "The closest University to you is: {0}.",
        university.Name);
}
```

Compilez et exécutez l'application. Le programme génère la sortie suivante :

```
The closest University to you is: School of Fine Art.
```

Pour afficher les données de la base de données, cliquez avec le bouton droit sur le nom de la base de données dans Explorateur d'objets SQL Server, puis sélectionnez **Actualiser**. Ensuite, cliquez sur le bouton droit de la souris sur la table et sélectionnez **afficher les données**.

## Résumé

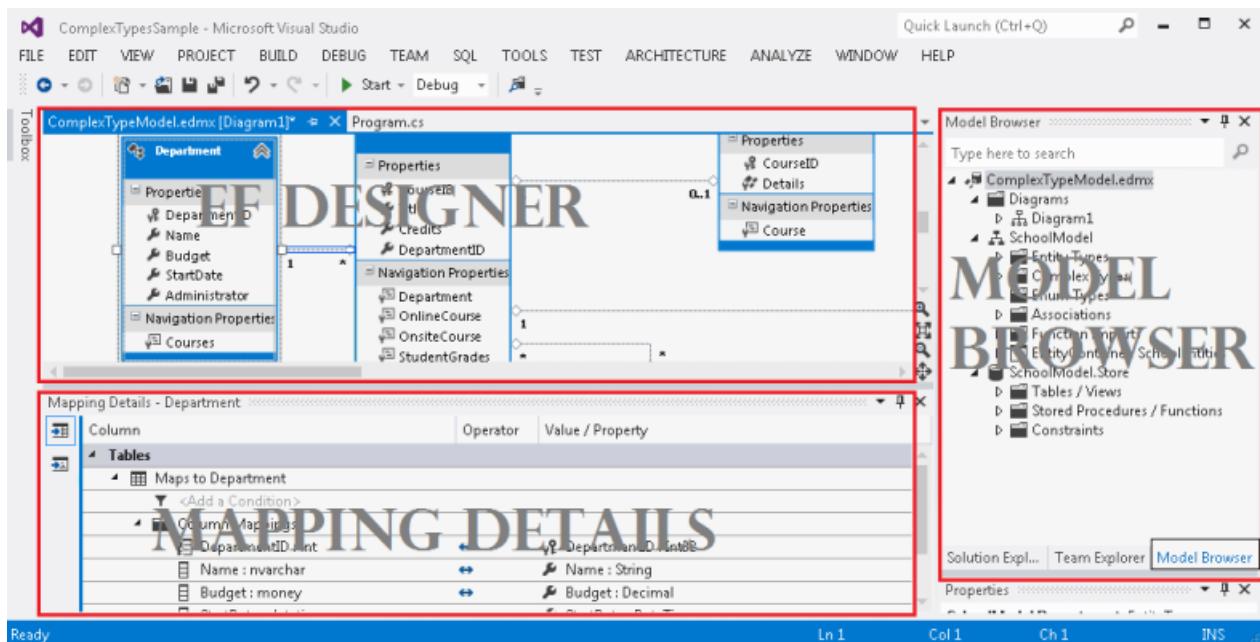
Dans cette procédure pas à pas, nous avons vu comment mapper des types spatiaux à l'aide de la Entity Framework Designer et comment utiliser des types spatiaux dans le code.

# Fractionnement d'entité Concepteur

13/09/2018 • 8 minutes to read

Cette procédure pas à pas montre comment mapper un type d'entité à deux tables en modifiant un modèle avec Entity Framework Designer (Concepteur d'EF). Vous pouvez mapper une entité à plusieurs tables quand les tables partagent une clé commune. Les concepts qui s'appliquent au mappage d'un type d'entité à deux tables sont facilement étendus au mappage d'un type d'entité à plusieurs tables.

L'illustration suivante montre les principales fenêtres qui sont utilisées lorsque vous travaillez avec le Concepteur EF.



## Prérequis

Visual Studio 2012 ou Visual Studio 2010, Premium, Professionnel, Édition Ultimate ou Web Express.

## Créer la base de données

Le serveur de base de données qui est installé avec Visual Studio est différent selon la version de Visual Studio que vous avez installé :

- Si vous utilisez Visual Studio 2012 puis vous allez créer une base de données de base de données locale.
- Si vous utilisez Visual Studio 2010, vous créerez une base de données SQL Express.

Tout d'abord, nous allons créer une base de données avec deux tables que nous allons à combiner en une seule entité.

- Ouvrir Visual Studio
- **Vue -> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **des connexions de données -> ajouter une connexion...**
- Si vous n'avez pas connecté à une base de données à partir de l'Explorateur de serveurs avant que vous devrez sélectionner **Microsoft SQL Server** comme source de données
- Se connecter à la base de données locale ou de SQL Express, en fonction de celles que vous avez installé
- Entrez **EntitySplitting** en tant que le nom de la base de données

- Sélectionnez **OK** et vous demandera si vous souhaitez créer une base de données, sélectionnez **Oui**
- La nouvelle base de données s'affiche désormais dans l'Explorateur de serveurs
- Si vous utilisez Visual Studio 2012
  - Avec le bouton droit sur la base de données dans l'Explorateur de serveurs, puis sélectionnez **nouvelle requête**
  - Copiez le code SQL suivant dans la nouvelle requête, puis avec le bouton droit sur la requête, puis sélectionnez **Execute**
- Si vous utilisez Visual Studio 2010
  - Sélectionnez **données -> Transact SQL éditeur -> nouvelle connexion à la requête...**
  - Entrez **.\\SQLEXPRESS** en tant que nom du serveur et cliquez sur **OK**
  - Sélectionnez le **EntitySplitting** de base de données dans la liste déroulante vers le bas en haut de l'éditeur de requête
  - Copiez le code SQL suivant dans la nouvelle requête, puis avec le bouton droit sur la requête, puis sélectionnez **exécuter SQL**

```

CREATE TABLE [dbo].[Person] (
[PersonId] INT IDENTITY (1, 1) NOT NULL,
[FirstName] NVARCHAR (200) NULL,
[LastName] NVARCHAR (200) NULL,
CONSTRAINT [PK_Person] PRIMARY KEY CLUSTERED ([PersonId] ASC)
);

CREATE TABLE [dbo].[PersonInfo] (
[PersonId] INT NOT NULL,
[Email] NVARCHAR (200) NULL,
[Phone] NVARCHAR (50) NULL,
CONSTRAINT [PK_PersonInfo] PRIMARY KEY CLUSTERED ([PersonId] ASC),
CONSTRAINT [FK_Person_PersonInfo] FOREIGN KEY ([PersonId]) REFERENCES [dbo].[Person] ([PersonId]) ON DELETE CASCADE
);

```

## Créer le projet

- Dans le menu **Fichier**, pointez sur **Nouveau**, puis cliquez sur **Projet**.
- Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le **Application Console** modèle.
- Entrez **MapEntityToTablesSample** en tant que le nom du projet et cliquez sur **OK**.
- Cliquez sur **non** si vous êtes invité à enregistrer la requête SQL créée dans la première section.

## Créer un modèle basé sur la base de données

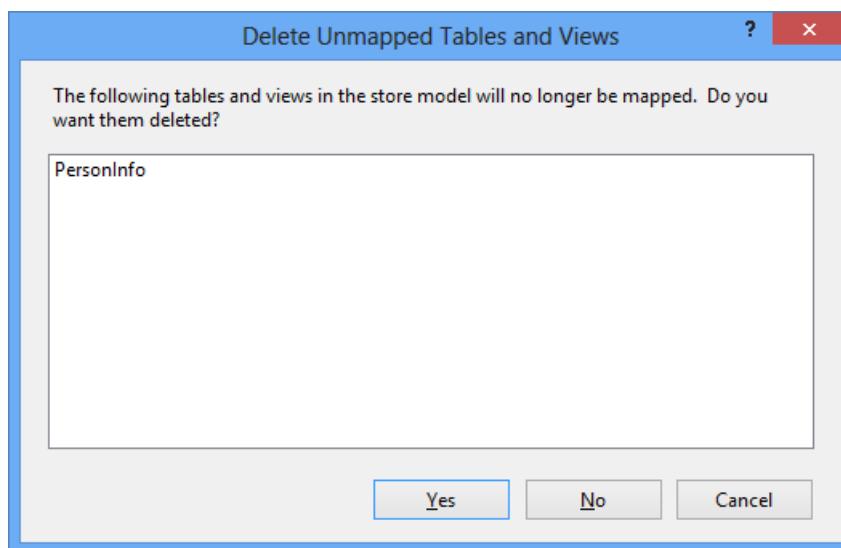
- Cliquez sur le nom de projet dans l'Explorateur de solutions, pointez sur **ajouter**, puis cliquez sur **un nouvel élément**.
- Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet Modèles.
- Entrez **MapEntityToTablesModel.edmx** pour le nom de fichier, puis cliquez sur **ajouter**.
- Dans la boîte de dialogue Choisir le contenu du modèle, sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.
- Sélectionnez le **EntitySplitting** connexion dans la liste déroulante et cliquez sur **suivant**.
- Dans la boîte de dialogue Choisir vos objets de base de données, cochez la case à côté du **Tables** noeud. Cette opération ajoute toutes les tables à partir de la **EntitySplitting** base de données au modèle.
- Cliquez sur **Terminer**.

Le Concepteur d'entités, qui fournit une aire de conception pour la modification de votre modèle, s'affiche.

## Mapper une entité à deux Tables

Dans cette étape nous mettrons à jour le **personne** type d'entité à combiner des données à partir de la **personne** et **PersonInfo** tables.

- Sélectionnez le **E-mail** et **téléphone** propriétés de la \*\* PersonInfo \*\* entité, puis appuyez sur **Ctrl + X** clés.
- Sélectionnez le \*\* personne \*\* entité, puis appuyez sur **Ctrl + V** clés.
- Sur l'aire de conception, sélectionnez le **PersonInfo** entité, puis appuyez sur **supprimer** bouton sur le clavier.
- Cliquez sur **non** lorsque vous êtes invité à supprimer le **PersonInfo** table à partir du modèle, nous sommes sur le point de la mapper à la **personne** entité.



Les étapes suivantes requièrent le **détails de Mapping** fenêtre. Si vous ne voyez pas cette fenêtre, cliquez sur l'aire de conception et sélectionnez **détails de Mapping**.

- Sélectionnez le **personne** type d'entité et cliquez sur <**ajouter une Table ou vue**> dans le **détails de Mapping** fenêtre.
- Sélectionnez \*\* PersonInfo \*\* dans la liste déroulante. Le **détails de Mapping** fenêtre est mis à jour avec les mappages de colonnes par défaut, le résultat est parfaits pour notre scénario.

Le **personne** type d'entité est maintenant mappé à la **personne** et **PersonInfo** tables.

A screenshot of the "Mapping Details - Person" window. The window has a tree view on the left and a table view on the right. The tree view shows "Tables" expanded, with "Maps to Person" and "Maps to PersonInfo" selected. The table view shows mappings between columns:

| Column               | Operator | Value / Property   |
|----------------------|----------|--------------------|
| PersonId : int       | =        | PersonId : Int32   |
| FirstName : nvarchar | =        | FirstName : String |
| LastName : nvarchar  | =        | LastName : String  |
| PersonId : int       | =        | PersonId : Int32   |
| Email : nvarchar     | =        | Email : String     |
| Phone : nvarchar     | =        | Phone : String     |

## Utiliser le modèle

- Collez le code suivant dans la méthode Main.

```
using (var context = new EntitySplittingEntities())
{
    var person = new Person
    {
        FirstName = "John",
        LastName = "Doe",
        Email = "john@example.com",
        Phone = "555-555-5555"
    };

    context.People.Add(person);
    context.SaveChanges();

    foreach (var item in context.People)
    {
        Console.WriteLine(item.FirstName);
    }
}
```

- Compilez et exéutez l'application.

Les instructions T-SQL suivantes ont été exécutées par rapport à la base de données suite à l'exécution de cette application.

- Les deux **insérer** instructions ont été exécutées suite à l'exécution de contexte. SaveChanges(). Ils prennent les données à partir de la **personne** entité et la fractionner entre le **personne** et **PersonInfo** tables.

⚡ **ADO.NET:** Execute Reader "insert [dbo].[Person]([FirstName], [LastName]) values (@0, @1) select [PersonId] from [dbo].[Person] where @@ROWCOUNT > 0 and [PersonId] = scope\_identity()"  
The command text "insert [dbo].[Person]([FirstName], [LastName]) values (@0, @1) select [PersonId] from [dbo].[Person] where @@ROWCOUNT > 0 and [PersonId] = scope\_identity()" was executed on connection "data source=(localdb)\v11.0;initial catalog=EntitySplitting;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", building a SqlDataReader.  
**Thread:** Main Thread [2052]  
**Related views:** [Locals](#) [Call Stack](#)

⚡ **ADO.NET:** Execute NonQuery "insert [dbo].[PersonInfo]([PersonId], [Email], [Phone]) values (@0, @1, @2)"  
The command text "insert [dbo].[PersonInfo]([PersonId], [Email], [Phone]) values (@0, @1, @2)" was executed on connection "data source=(localdb)\v11.0;initial catalog=EntitySplitting;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", returning the number of rows affected.  
**Thread:** Main Thread [2052]  
**Related views:** [Locals](#) [Call Stack](#)

- Ce qui suit **sélectionnez** a été exécutée à la suite de l'énumération des personnes de la base de données. Il combine les données à partir de la **personne** et **PersonInfo** table.

**ADO.NET: Execute Reader "SELECT [Extent1].[PersonId] AS [Per**  
The command text "SELECT  
[Extent1].[PersonId] AS [PersonId],  
[Extent2].[FirstName] AS [FirstName],  
[Extent2].[LastName] AS [LastName],  
[Extent1].[Email] AS [Email],  
[Extent1].[Phone] AS [Phone]  
FROM [dbo].[PersonInfo] AS [Extent1]  
INNER JOIN [dbo].[Person] AS [Extent2] ON [Extent1].[PersonId] = [Extent2].[PersonId]" was executed on connection  
"data source=(localdb)\v11.0;initial  
catalog=EntitySplitting;integrated  
security=True;MultipleActiveResultSets=True;App=EntityFram  
ework", building a SqlDataReader.  
**Thread:** Main Thread [2052]

**Related views:** [Locals](#) [Call Stack](#)

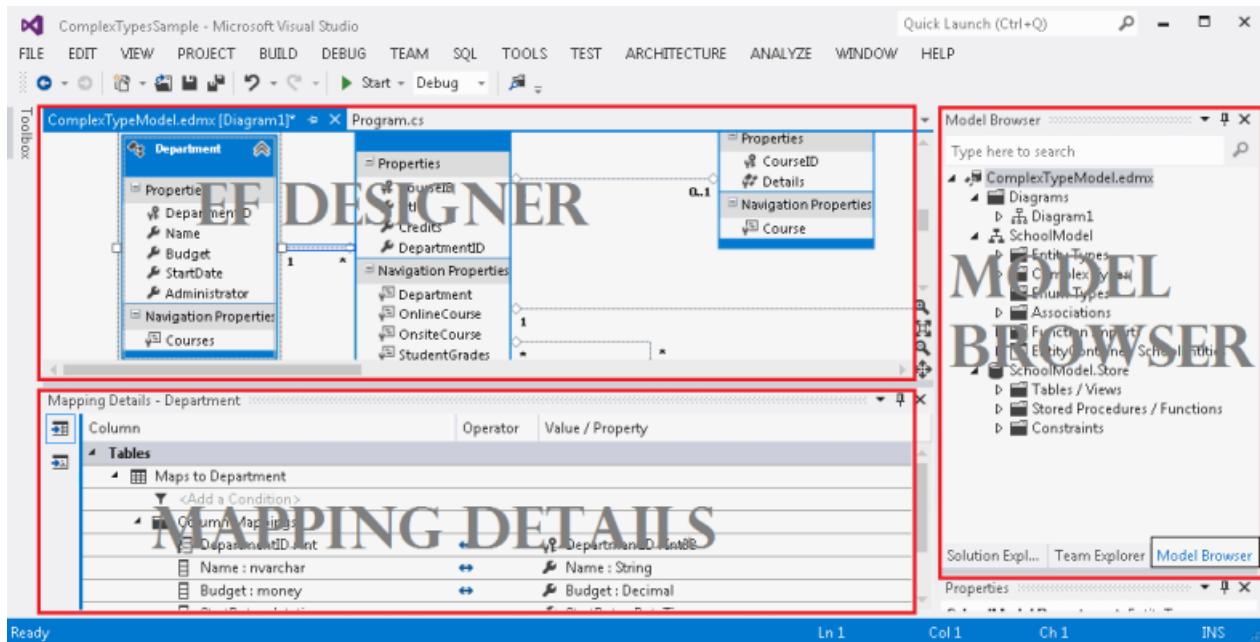
# Fractionnement des tables du concepteur

23/11/2019 • 7 minutes to read

Cette procédure pas à pas montre comment mapper plusieurs types d'entité à une seule table en modifiant un modèle avec le Entity Framework Designer (concepteur EF).

L'une des raisons pour lesquelles vous pouvez souhaiter utiliser le fractionnement de table est de retarder le chargement de certaines propriétés lors de l'utilisation du chargement différé pour charger vos objets. Vous pouvez séparer les propriétés qui peuvent contenir de très grandes quantités de données dans une entité distincte et les charger uniquement lorsque cela est nécessaire.

L'illustration suivante montre les fenêtres principales qui sont utilisées lors de l'utilisation du concepteur EF.



## Configuration requise

Pour exécuter cette procédure pas à pas, vous avez besoin des éléments suivants :

- Une version récente de Visual Studio.
- [Exemple de base de données School](#).

## Configurer le projet

Cette procédure pas à pas utilise Visual Studio 2012.

- Ouvrez Visual Studio 2012.
- Dans le menu **Fichier**, pointez sur **Nouveau**, puis cliquez sur **Projet**.
- Dans le volet gauche, cliquez sur Visual C#, puis sélectionnez le modèle application console.
- Entrez **TableSplittingSample** comme nom du projet, puis cliquez sur **OK**.

## Créer un modèle basé sur la base de données School

- Cliquez avec le bouton droit sur le nom du projet dans Explorateur de solutions, pointez sur **Ajouter**, puis cliquez sur **nouvel élément**.

- Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet modèles.
- Entrez **TableSplittingModel.edmx** comme nom de fichier, puis cliquez sur **Ajouter**.
- Dans la boîte de dialogue choisir le contenu du Model, sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.
- Cliquez sur nouvelle connexion. Dans la boîte de dialogue Propriétés de connexion, entrez le nom du serveur (par exemple, (base de données locale) \mssqllocaldb), sélectionnez la méthode d'authentification, tapez **School** pour le nom de la base de données, puis cliquez sur **OK**. La boîte de dialogue choisir votre connexion de données est mise à jour avec votre paramètre de connexion à la base de données.
- Dans la boîte de dialogue choisir vos objets de base de données, dérouler les **Tables** nœud et vérifier la table **Person**. Cette opération ajoute la table spécifiée au modèle **School**.
- Cliquez sur **Terminer**.

Le Entity Designer, qui fournit une aire de conception pour la modification de votre modèle, est affiché. Tous les objets que vous avez sélectionnés dans la boîte de dialogue choisir vos objets de base de données sont ajoutés au modèle.

## Mapper deux entités à une table unique

Dans cette section, vous allez fractionner l'entité **Person** en deux entités, puis les mapper à une table unique.

### NOTE

L'entité **Person** ne contient pas de propriétés pouvant contenir une grande quantité de données ; elle est utilisée à titre d'exemple.

- Cliquez avec le bouton droit sur une zone vide de l'aire de conception, pointez sur **Ajouter nouveau**, puis cliquez sur **entité**. La boîte de dialogue nouvelle d'entité s'affiche.
- Tapez **HireInfo** pour le nom de l' **entité** et **PersonID** pour le nom de la **propriété de clé** .
- Cliquez sur **OK**.
- Un nouveau type d'entité est créé et affiché sur l'aire de conception.
- Sélectionnez la propriété **hiredate** de la **personne** type d'entité, puis appuyez sur **CTRL + X** .
- Sélectionnez l'entité **HireInfo** et appuyez sur les touches **Ctrl + V** .
- Créer une association entre **Person** et **HireInfo**. Pour ce faire, cliquez avec le bouton droit sur une zone vide de l'aire de conception, pointez sur **Ajouter nouveau**, puis cliquez sur **Association**.
- La boîte de dialogue Ajouter un d' **Association** s'affiche. Le nom **PersonHireInfo** est donné par défaut.
- Spécifiez la multiplicité **1 (un)** aux deux extrémités de la relation.
- Appuyez sur **OK**.

L'étape suivante nécessite la fenêtre **Détails de mappage** . Si vous ne voyez pas cette fenêtre, cliquez avec le bouton droit sur l'aire de conception, puis sélectionnez **Détails de mapping**.

- Sélectionnez le type d'entité **HireInfo** , puis cliquez sur **<ajouter une table ou une vue>** dans la fenêtre des **Détails de mappage** .
- Sélectionnez **Person** dans la liste déroulante **<ajouter une table ou une vue>** champ. La liste contient les tables ou les vues auxquelles l'entité sélectionnée peut être mappée. Les propriétés appropriées doivent être mappées par défaut.

The screenshot shows the 'Mapping Details' window with the following details:

- Tables:** Maps to Person
- Column Mappings:**
  - PersonID : int ↔ PersonID : Int32
  - LastName : nvarchar ↔ ↗
  - FirstName : nvarchar ↔ ↗
  - HireDate : datetime ↔ ↗ HireDate : DateTime
  - EnrollmentDate : datetime ↔ ↗
  - Discriminator : nvarchar ↔ ↗

- Sélectionnez l'Association **PersonHireInfo** sur l'aire de conception.
- Cliquez avec le bouton droit sur l'Association sur l'aire de conception, puis sélectionnez **Propriétés**.
- Dans la fenêtre **Propriétés**, sélectionnez la propriété **contraintes référentielles**, puis cliquez sur le bouton de sélection.
- Sélectionnez **Person** dans la liste déroulante **principal**.
- Appuyez sur **OK**.

## Utiliser le modèle

- Collez le code suivant dans la méthode main.

```
using (var context = new SchoolEntities())
{
    Person person = new Person()
    {
        FirstName = "Kimberly",
        LastName = "Morgan",
        Discriminator = "Instructor",
    };

    person.HireInfo = new HireInfo()
    {
        HireDate = DateTime.Now
    };

    // Add the new person to the context.
    context.People.Add(person);

    // Insert a row into the Person table.
    context.SaveChanges();

    // Execute a query against the Person table.
    // The query returns columns that map to the Person entity.
    var existingPerson = context.People.FirstOrDefault();

    // Execute a query against the Person table.
    // The query returns columns that map to the Instructor entity.
    var hireInfo = existingPerson.HireInfo;

    Console.WriteLine("{0} was hired on {1}",
        existingPerson.LastName, hireInfo.HireDate);
}
```

- Compilez et exéutez l'application.

Les instructions T-SQL suivantes ont été exécutées sur la base de données **School** suite à l'exécution de cette application.

- L' instruction **INSERT** suivante a été exécutée à la suite de l'exécution du contexte. SaveChanges () et combine les données des entités **Person** et **HireInfo**

⚡ **ADO.NET:** Execute Reader "insert [dbo].[Person]([LastName], [FirstName], [HireDate], [EnrollmentDate], [Discriminator]) values (@0, @1, @2, null, @3) select [PersonID] from [dbo].[Person] where @@ROWCOUNT > 0 and [PersonID] = scope\_identity()" was executed on connection "data source=(localdb)\v11.0;initial catalog=School;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", building a SqlDataReader.

- La commande **Select** suivante a été exécutée à la suite de l'exécution du contexte. People. FirstOrDefault () et sélectionne uniquement les colonnes mappées à **Person**

⚡ **ADO.NET:** Execute Reader "SELECT TOP (1) [c].[PersonID], [c].[LastName] AS [LastName], [c].[FirstName] AS [FirstName], [c].[EnrollmentDate] AS [EnrollmentDate], [c].[Discriminator] AS [Discriminator] FROM [dbo].[Person] AS [c]" was executed on connection "data source=(localdb)\v11.0;initial catalog=School;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", building a SqlDataReader.

- La commande **Select** suivante a été exécutée suite à l'accès à la propriété de navigation existingPerson. Instructor et sélectionne uniquement les colonnes mappées à **HireInfo**

⚡ **ADO.NET:** Execute Reader "SELECT [Extent1].[PersonID], [Extent1].[HireDate] AS [HireDate] FROM [dbo].[Person] AS [Extent1] WHERE [Extent1].[PersonID] = @EntityKeyValue1" was executed on connection "data source=(localdb)\v11.0;initial catalog=School;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework", building a SqlDataReader.

# Héritage TPH Concepteur

13/09/2018 • 11 minutes to read

Cette procédure pas à pas montre comment implémenter l'héritage de table par hiérarchie (TPH) dans votre modèle conceptuel avec Entity Framework Designer (Concepteur d'EF). L'héritage TPH utilise une table de base de données pour gérer les données de tous les types d'entités dans une hiérarchie d'héritage.

Dans cette procédure pas à pas, nous allons mapper la table Person à trois types d'entités : Person (type de base), Student (dérive de personne) et Instructor (dérive de personne). Nous allons créer un modèle conceptuel à partir de la base de données (Database First), puis la modifier le modèle pour implémenter l'héritage TPH à l'aide du Concepteur EF.

Il est possible de mapper à un héritage TPH à l'aide de Model First, mais vous seriez obligé d'écrire vos propres flux de travail de génération de base de données qui est complexe. Il vous faut ensuite attribuer ce flux de travail pour le **flux de génération de base de données** propriété dans le Concepteur EF. Une alternative plus simple consiste à utiliser Code First.

## Autres Options d'héritage

Table par Type (TPT) est un autre type d'héritage dans laquelle des tables distinctes de la base de données sont mappées aux entités qui participent à l'héritage. Pour plus d'informations sur le mappage d'héritage Table par Type avec le Concepteur d'Entity Framework, consultez [l'héritage TPT de Concepteur EF](#).

L'héritage de Type de table par concret (TPC) et les modèles d'héritage mixte sont prises en charge par le runtime Entity Framework, mais ne sont pas pris en charge par le Concepteur EF. Si vous souhaitez utiliser TPC ou mixte de l'héritage, vous avez deux options : utiliser Code First, ou modifier manuellement le fichier EDMX. Si vous choisissez d'utiliser le fichier EDMX, la fenêtre Détails de mappage est placée dans le « mode sans échec » et vous ne serez pas en mesure d'utiliser le concepteur pour modifier les mappages.

## Prérequis

Pour exécuter cette procédure pas à pas, vous avez besoin des éléments suivants :

- Une version récente de Visual Studio.
- Le [base de données School exemple](#).

## Configurer le projet

- Ouvrez Visual Studio 2012.
- Sélectionnez **fichier -> nouveau -> projet**
- Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le **Console** modèle.
- Entrez **TPHDBFirstSample** comme nom.
- Sélectionnez **OK**.

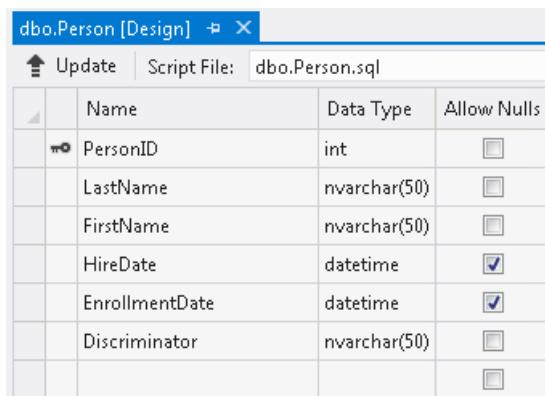
## Créer un modèle

- Cliquez sur le nom du projet dans l'Explorateur de solutions, puis sélectionnez **Add -> un nouvel élément**.
- Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet Modèles.
- Entrez **TPHModel.edmx** pour le nom de fichier, puis cliquez sur **ajouter**.

- Dans la boîte de dialogue Choisir le contenu du modèle, sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.
- Cliquez sur **nouvelle connexion**. Dans la boîte de dialogue Propriétés de connexion, entrez le nom du serveur (par exemple, **(localdb)\mssqllocaldb**), sélectionnez la méthode d'authentification, tapez **School** pour le nom de la base de données, puis Cliquez sur **OK**. La boîte de dialogue Choisir votre connexion de données est mis à jour avec le paramètre de votre connexion de base de données.
- Dans la boîte de dialogue Choisir vos objets de base de données, sous le noeud Tables, sélectionnez le **personne** table.
- Cliquez sur **Terminer**.

Le Concepteur d'entités, qui fournit une aire de conception pour la modification de votre modèle, s'affiche. Tous les objets que vous avez sélectionné dans la boîte de dialogue Choisir vos objets de base de données sont ajoutées au modèle.

Autrement dit la **personne** table se présente dans la base de données.



The screenshot shows the EntityDataSource1 Properties window with the 'ConnectionString' tab selected. It displays the connection string configuration for the EntityDataSource.

| Name                 | Type                                  | Description          |
|----------------------|---------------------------------------|----------------------|
| DefaultContainerName | System.Data.Entity.DbContainer        | DefaultContainerName |
| ConnectionString     | System.Data.Entity.DbConnectionString | ConnectionString     |

ConnectionString Properties

| Name                 | Type                                  | Value                |
|----------------------|---------------------------------------|----------------------|
| DefaultContainerName | System.Data.Entity.DbContainer        | DefaultContainerName |
| ConnectionString     | System.Data.Entity.DbConnectionString | DefaultContainerName |

ConnectionString

| Name           | Type         | Allow Nulls                         |
|----------------|--------------|-------------------------------------|
| PersonID       | int          | <input type="checkbox"/>            |
| LastName       | nvarchar(50) | <input type="checkbox"/>            |
| FirstName      | nvarchar(50) | <input type="checkbox"/>            |
| HireDate       | datetime     | <input checked="" type="checkbox"/> |
| EnrollmentDate | datetime     | <input checked="" type="checkbox"/> |
| Discriminator  | nvarchar(50) | <input type="checkbox"/>            |

## Implémenter l'héritage Table par hiérarchie

Le **personne** table comporte le **discriminateur** colonne, ce qui peut avoir une des deux valeurs : « L'étudiant » et « Instructor ». Selon la valeur la **personne** table sera mappée à la **étudiant** entité ou le **Instructor** entité. Le **personne** table possède également deux colonnes, **HireDate** et **EnrollmentDate**, qui doit être **nullable**, car une personne ne peut pas être un stagiaire et formateur en même temps (au moins pas dans cette procédure pas à pas).

### Ajouter de nouvelles entités

- Ajouter une nouvelle entité. Pour ce faire, avec le bouton droit sur un espace vide de l'aire de conception d'Entity Framework Designer et sélectionnez **Add ->entité**.
- Type **Instructor** pour le **nom de l'entité** et sélectionnez **personne** dans la liste déroulante pour le **type de Base**.
- Cliquez sur **OK**.
- Ajouter une autre nouvelle entité. Type **étudiant** pour le **nom de l'entité** et sélectionnez **personne** dans la liste déroulante pour le **type de Base**.

Deux nouveaux types d'entités ont été ajoutés à l'aire de conception. Une flèche pointe à partir de nouveaux types d'entités pour le **personne** type d'entité ; cela indique que **personne** est le type de base pour les nouveaux types d'entité.

- Avec le bouton droit le **HireDate** propriété de la **personne** entité. Sélectionnez **couper** (ou utilisez la touche Ctrl + X).
- Avec le bouton droit le **Instructor** entité, puis sélectionnez **coller** (ou utilisez la touche Ctrl + V).
- Cliquez sur le **HireDate** propriété et sélectionnez **propriétés**.
- Dans le **propriétés** fenêtre, définissez la **Nullable** propriété **false**.

- Avec le bouton droit le **EnrollmentDate** propriété de la **personne** entité. Sélectionnez **couper** (ou utilisez la touche Ctrl + X).
- Avec le bouton droit le **étudiant** entité, puis sélectionnez **Coller (ou clé d'utilisation le Ctrl + V)**.
- Sélectionnez le **EnrollmentDate** propriété et définissez la **Nullable** propriété **false**.
- Sélectionnez le **personne** type d'entité. Dans le **propriétés** fenêtre, définissez son **abstraite** propriété **true**.
- Supprimer le **discriminateur** propriété à partir de **personne**. La raison pour laquelle qu'il doit être supprimé est expliquée dans la section suivante.

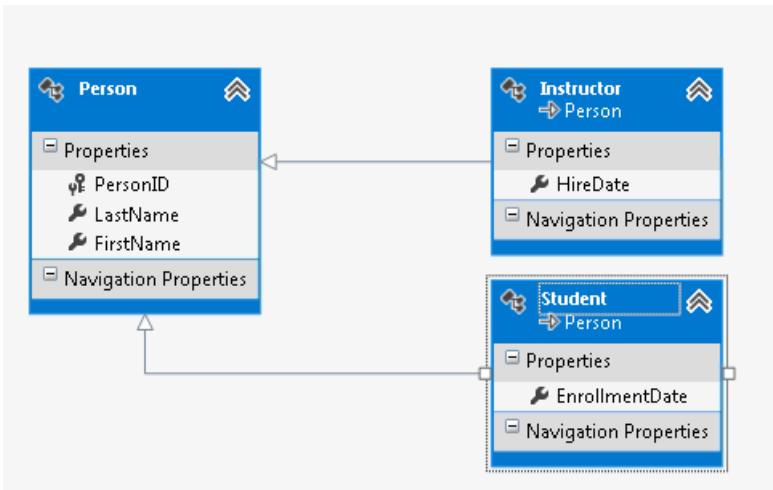
## Mapper les entités

- Avec le bouton droit le **Instructor** et sélectionnez **mappage de Table**. L'entité Instructor est sélectionnée dans la fenêtre Détails de Mapping.
- Cliquez sur **<ajouter une Table ou vue>** dans le **détails de Mapping** fenêtre. Le **<ajouter une Table ou vue>** champ devient une liste déroulante de tables ou des vues, auquel l'entité sélectionnée peut être mappée.
- Sélectionnez **personne** dans la liste déroulante.
- Le **détails de Mapping** fenêtre est mis à jour avec les mappages de colonnes par défaut et une option permettant d'ajouter une condition.
- Cliquez sur **<ajouter une Condition>**. Le **<ajouter une Condition>** champ devient une liste déroulante des colonnes pour lesquelles des conditions peuvent être définies.
- Sélectionnez **discriminateur** dans la liste déroulante.
- Dans le **opérateur** colonne de la **détails de Mapping** fenêtre, sélectionnez **=** dans la liste déroulante.
- Dans le **valeur/propriété** colonne, tapez **Instructor**. Le résultat final doit ressembler à ceci :

| Column                    | Operator | Value / Property    |
|---------------------------|----------|---------------------|
| Tables                    |          |                     |
| Maps to Person            |          |                     |
| When Discriminator        | =        | Instructor          |
| <Add a Condition>         |          |                     |
| Column Mappings           |          |                     |
| HireDate : datetime       | ↔        | HireDate : DateTime |
| EnrollmentDate : datetime | ↔        |                     |
| Discriminator : nvarchar  | ↔        |                     |

- Répétez ces étapes pour le **étudiant** type d'entité, mais vérifiez la condition égale à **étudiant** valeur.  
*La raison pour laquelle nous souhaitons supprimer le **discriminateur** propriété, est, car vous ne pouvez pas mapper une colonne de table plusieurs fois. Cette colonne sera utilisée pour le mappage conditionnel, afin qu'il ne peut pas être utilisé pour le mappage de propriété ainsi. La seule façon, il peut être utilisé pour les deux, si une condition utilise une **Is Null** ou **Is Not Null** comparaison.*

L'héritage TPH (table par hiérarchie) est maintenant implémenté.



## Utiliser le modèle

Ouvrez le **Program.cs** fichier où le **Main** méthode est définie. Collez le code suivant dans le **Main** (fonction). Le code s'exécute trois requêtes. La première requête renvoie tous les **personne** objets. La deuxième requête utilise le **OfType** méthode pour retourner **Instructor** objets. La troisième requête utilise le **OfType** méthode pour retourner **étudiant** objets.

```

using (var context = new SchoolEntities())
{
    Console.WriteLine("All people:");
    foreach (var person in context.People)
    {
        Console.WriteLine("    {0} {1}", person.FirstName, person.LastName);
    }

    Console.WriteLine("Instructors only: ");
    foreach (var person in context.People.OfType<Instructor>())
    {
        Console.WriteLine("    {0} {1}", person.FirstName, person.LastName);
    }

    Console.WriteLine("Students only: ");
    foreach (var person in context.People.OfType<Student>())
    {
        Console.WriteLine("    {0} {1}", person.FirstName, person.LastName);
    }
}

```

# Héritage TPT Concepteur

13/09/2018 • 7 minutes to read

Cette procédure pas à pas montre comment implémenter l'héritage de table par type (TPT) dans votre modèle à l'aide d'Entity Framework Designer (Concepteur d'EF). L'héritage TPT (table par type) utilise une table distincte de la base de données pour maintenir des données des propriétés non héritées et des propriétés de clé pour chaque type de la hiérarchie d'héritage.

Dans cette procédure pas à pas, nous allons mapper le **cours** (type de base), **OnlineCourse** (dérive de cours), et **OnsiteCourse** (dérive **cours**) entités à des tables portant le même nom. Nous allons créer un modèle à partir de la base de données, puis la modifier le modèle pour implémenter l'héritage TPT.

Vous pouvez également commencer par le premier modèle, puis générez la base de données à partir du modèle. Le Concepteur EF utilise la stratégie de TPT par défaut et par conséquent, tout l'héritage dans le modèle sera mappée à des tables distinctes.

## Autres Options d'héritage

Table par hiérarchie (TPH) est un autre type d'héritage dans la base d'une données table est utilisée pour gérer les données de tous les types d'entités dans une hiérarchie d'héritage. Pour plus d'informations sur le mappage d'héritage Table par hiérarchie avec le Concepteur d'entités, consultez [l'héritage TPH de Concepteur EF](#).

Notez que, la Table par concret Type héritage (TPC) et l'héritage mixte modèles sont pris en charge par le runtime Entity Framework, mais ne sont pas pris en charge par le Concepteur EF. Si vous souhaitez utiliser TPC ou mixte de l'héritage, vous avez deux options : utiliser Code First, ou modifier manuellement le fichier EDMX. Si vous choisissez d'utiliser le fichier EDMX, la fenêtre Détails de mappage est placée dans le « mode sans échec » et vous ne serez pas en mesure d'utiliser le concepteur pour modifier les mappages.

## Prérequis

Pour exécuter cette procédure pas à pas, vous avez besoin des éléments suivants :

- Une version récente de Visual Studio.
- Le [base de données School exemple](#).

## Configurer le projet

- Ouvrez Visual Studio 2012.
- Sélectionnez **fichier -> nouveau -> projet**
- Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le **Console** modèle.
- Entrez **TPTDBFirstSample** comme nom.
- Sélectionnez **OK**.

## Créer un modèle

- Cliquez sur le projet dans l'Explorateur de solutions, puis sélectionnez **Add -> un nouvel élément**.
- Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet Modèles.
- Entrez **TPTModel.edmx** pour le nom de fichier, puis cliquez sur **ajouter**.
- Dans la boîte de dialogue Choisir le contenu du modèle, sélectionnez \*\* Générer à partir de base de données

\*\*, puis cliquez sur **suivant**.

- Cliquez sur **nouvelle connexion**. Dans la boîte de dialogue Propriétés de connexion, entrez le nom du serveur (par exemple, **(localdb)\mssqllocaldb**), sélectionnez la méthode d'authentification, tapez **School** pour le nom de la base de données, puis Cliquez sur **OK**. La boîte de dialogue Choisir votre connexion de données est mis à jour avec le paramètre de votre connexion de base de données.
- Dans la boîte de dialogue Choisir vos objets de base de données, sous le nœud Tables, sélectionnez le **département, cours, OnlineCourse et OnsiteCourse** tables.
- Cliquez sur **Terminer**.

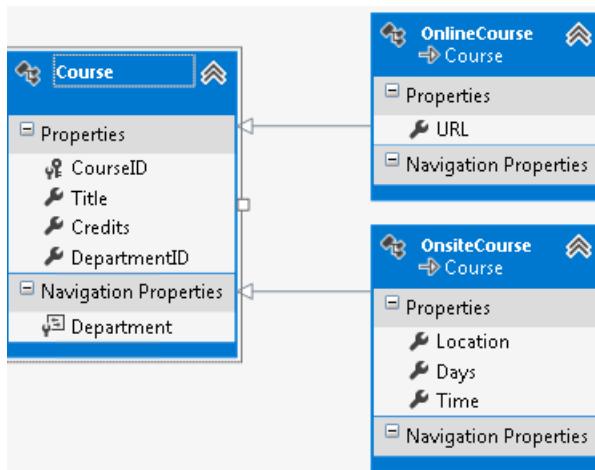
Le Concepteur d'entités, qui fournit une aire de conception pour la modification de votre modèle, s'affiche. Tous les objets que vous avez sélectionné dans la boîte de dialogue Choisir vos objets de base de données sont ajoutées au modèle.

## Implémenter l'héritage Table par Type

- Sur l'aire de conception, cliquez sur le **OnlineCourse** type d'entité, puis sélectionnez **propriétés**.
- Dans le **propriétés** fenêtre, définissez la propriété de Type de Base **cours**.
- Cliquez sur le **OnsiteCourse** type d'entité, puis sélectionnez **propriétés**.
- Dans le **propriétés** fenêtre, définissez la propriété de Type de Base **cours**.
- Avec le bouton droit de l'association (la ligne) entre le **OnlineCourse** et **cours** types d'entité. Sélectionnez **supprimer du modèle**.
- Avec le bouton droit de l'association entre le **OnsiteCourse** et **cours** types d'entité. Sélectionnez **supprimer du modèle**.

Nous va maintenant supprimer le **CourseID** propriété à partir de **OnlineCourse** et **OnsiteCourse** étant donné que ces classes héritent **CourseID** à partir de le **cours** type de base.

- Avec le bouton droit le **CourseID** propriété de la **OnlineCourse** type d'entité, puis **supprimer du modèle**.
- Avec le bouton droit le **CourseID** propriété de la **OnsiteCourse** type d'entité, puis **supprimer du modèle**
- L'héritage TPT (table par type) est maintenant implémenté.



## Utiliser le modèle

Ouvrez le **Program.cs** fichier où le **Main** méthode est définie. Collez le code suivant dans le **Main** (fonction). Le code s'exécute trois requêtes. La première requête renvoie tous les **cours** associées au département spécifié. La deuxième requête utilise le **OfType** méthode pour retourner **OnlineCourses** associées au département spécifié. Retourne la troisième requête **OnsiteCourses**.

```
using (var context = new SchoolEntities())
{
    foreach (var department in context.Departments)
    {
        Console.WriteLine("The {0} department has the following courses:",
                          department.Name);

        Console.WriteLine("    All courses");
        foreach (var course in department.Courses )
        {
            Console.WriteLine("        {0}", course.Title);
        }

        foreach (var course in department.Courses.
                  OfType<OnlineCourse>())
        {
            Console.WriteLine("        Online - {0}", course.Title);
        }

        foreach (var course in department.Courses.
                  OfType<OnsiteCourse>())
        {
            Console.WriteLine("        Onsite - {0}", course.Title);
        }
    }
}
```

# Procédures stockées de requête du concepteur

23/11/2019 • 6 minutes to read

Cette procédure pas à pas explique comment utiliser le Entity Framework Designer (concepteur EF) pour importer des procédures stockées dans un modèle, puis appeler les procédures stockées importées pour récupérer les résultats.

Notez que Code First ne prend pas en charge le mappage à des procédures stockées ou à des fonctions. Toutefois, vous pouvez appeler des procédures stockées ou des fonctions à l'aide de la méthode System. Data. Entity. DbSet. SqlQuery. Exemple :

```
var query = context.Products.SqlQuery("EXECUTE [dbo].[GetAllProducts]");
```

## Configuration requise

Pour exécuter cette procédure pas à pas, vous avez besoin des éléments suivants :

- Une version récente de Visual Studio.
- [Exemple de base de données School](#).

## Configurer le projet

- Ouvrez Visual Studio 2012.
- Sélectionnez **fichier-> nouveau-> projet**
- Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le modèle **console** .
- Entrez **EFwithSProcsSample** comme nom.
- Sélectionnez **OK**.

## Créer un modèle

- Dans Explorateur de solutions, cliquez avec le bouton droit sur le projet, puis sélectionnez **Ajouter-> nouvel élément**.
- Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet modèles.
- Entrez **EFwithSProcsModel. edmx** comme nom de fichier, puis cliquez sur **Ajouter**.
- Dans la boîte de dialogue choisir le contenu du Model, sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.
- Cliquez sur **nouvelle connexion**.

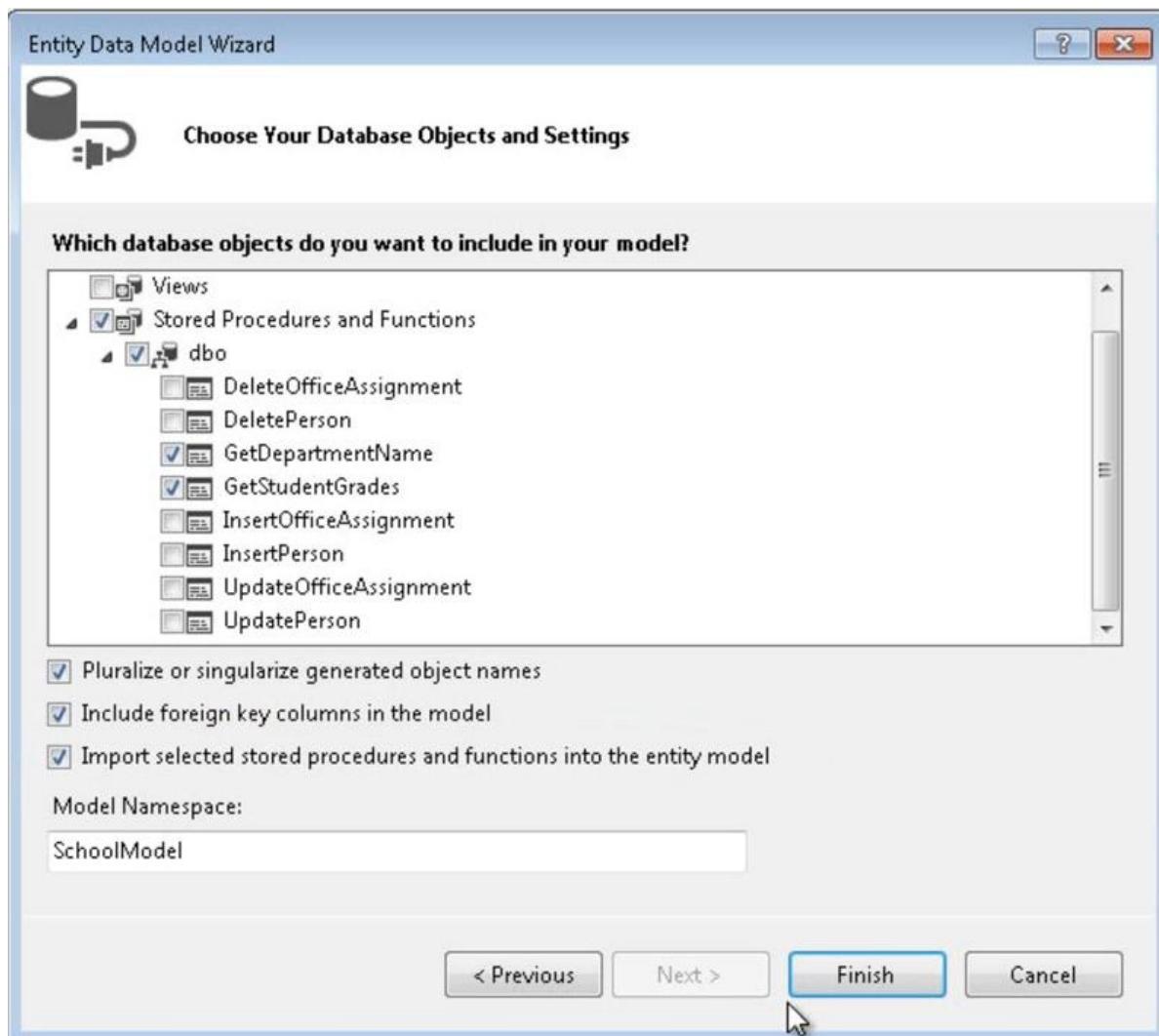
Dans la boîte de dialogue Propriétés de connexion, entrez le nom du serveur (par exemple, (base de données locale )\mssqllocaldb), sélectionnez la méthode d'authentification, tapez **School** pour le nom de la base de données, puis cliquez sur **OK**.

La boîte de dialogue choisir votre connexion de données est mise à jour avec votre paramètre de connexion à la base de données.

- Dans la boîte de dialogue choisir vos objets de base de données, activez la case à cocher **Tables** pour sélectionner toutes les tables.

En outre, sélectionnez les procédures stockées suivantes sous le noeud **procédures stockées et fonctions**

: GetStudentGrades et GetDepartmentName.



À compter de Visual Studio 2012, le concepteur EF prend en charge l'importation en bloc des procédures stockées. L'**importation des procédures stockées et des fonctions sélectionnées dans le modèle** **theentity** est activée par défaut.

- Cliquez sur **Terminer**.

Par défaut, la forme de résultat de chaque procédure stockée ou fonction importée qui retourne plusieurs colonnes devient automatiquement un nouveau type complexe. Dans cet exemple, nous voulons mapper les résultats de la fonction **GetStudentGrades** à l'entité **StudentGrade** et les résultats du **GetDepartmentName** à **None** (**None** est la valeur par défaut).

Pour qu'une importation de fonction retourne un type d'entité, les colonnes renvoyées par la procédure stockée correspondante doivent correspondre exactement aux propriétés scalaires du type d'entité retourné. Une importation de fonction peut également renvoyer des collections de types simples, des types complexes ou aucune valeur.

- Cliquez avec le bouton droit sur l'aire de conception, puis sélectionnez **Explorateur de modèles**.
- Dans l'**Explorateur de modèles**, sélectionnez importation de **fonction**, puis double-cliquez sur la fonction **GetStudentGrades**.
- Dans la boîte de dialogue Modifier l'importation de fonction, sélectionnez **entités**, puis choisissez **StudentGrade**.

La case **Importer une fonction peut être composable** en haut de la boîte de dialogue **importations de fonctions**, qui vous permet de mapper à des fonctions composites. Si vous activez cette case à cocher, seules les fonctions composites (fonctions table) s'affichent dans la liste déroulante nom de la **procédure**

**stockée/fonction**. Si vous n'activez pas cette case à cocher, seules les fonctions non componables seront affichées dans la liste.

## Utiliser le modèle

Ouvrez le fichier **Program.cs** dans lequel la méthode **main** est définie. Ajoutez le code suivant à la fonction main.

Le code appelle deux procédures stockées : **GetStudentGrades** (retourne **StudentGrades** pour le *StudentID* spécifié) et **GetDepartmentName** (retourne le nom du département dans le paramètre de sortie).

```
using (SchoolEntities context = new SchoolEntities())
{
    // Specify the Student ID.
    int studentId = 2;

    // Call GetStudentGrades and iterate through the returned collection.
    foreach (StudentGrade grade in context.GetStudentGrades(studentId))
    {
        Console.WriteLine("StudentID: {0}\tSubject={1}", studentId, grade.Subject);
        Console.WriteLine("Student grade: " + grade.Grade);
    }

    // Call GetDepartmentName.
    // Declare the name variable that will contain the value returned by the output parameter.
    ObjectParameter name = new ObjectParameter("Name", typeof(String));
    context.GetDepartmentName(1, name);
    Console.WriteLine("The department name is {0}", name.Value);

}
```

Compilez et exécutez l'application. Le programme génère la sortie suivante :

```
StudentID: 2
Student grade: 4.00
StudentID: 2
Student grade: 3.50
The department name is Engineering
```

## Paramètres de sortie

Si des paramètres de sortie sont utilisés, leurs valeurs ne sont pas disponibles tant que les résultats n'ont pas été entièrement lus. Cela est dû au comportement sous-jacent de **DbDataReader**. Pour plus d'informations, consultez [récupération de données à l'aide d'un DataReader](#).

# Procédures stockées CUD du concepteur

23/11/2019 • 10 minutes to read

Cette procédure pas à pas explique comment mapper les opérations Create\Insert, Update et Delete (CUD) d'un type d'entité aux procédures stockées à l'aide de l'Entity Framework Designer (concepteur EF). Par défaut, le Entity Framework génère automatiquement les instructions SQL pour les opérations CUD, mais vous pouvez également mapper des procédures stockées à ces opérations.

Notez que Code First ne prend pas en charge le mappage à des procédures stockées ou à des fonctions. Toutefois, vous pouvez appeler des procédures stockées ou des fonctions à l'aide de la méthode System. Data. Entity. DbSet. SqlQuery. Exemple :

```
var query = context.Products.SqlQuery("EXECUTE [dbo].[GetAllProducts]");
```

## Considérations relatives au mappage des opérations CUD à des procédures stockées

Lors du mappage des opérations CUD à des procédures stockées, les considérations suivantes s'appliquent :

- Si vous mappez l'une des opérations CUD à une procédure stockée, mappez-les toutes. Si vous ne mappez pas les trois, les opérations non mappées échouent si elles sont exécutées et une **UpdateException** est levée.
- Vous devez mapper chaque paramètre de la procédure stockée à des propriétés d'entité.
- Si le serveur génère la valeur de clé primaire pour la ligne insérée, vous devez mapper cette valeur à la propriété de clé de l'entité. Dans l'exemple qui suit, la procédure stockée **InsertPerson** retourne la clé primaire nouvellement créée dans le cadre du jeu de résultats de la procédure stockée. La clé primaire est mappée à la clé d'entité (**PersonID**) à l'aide de la **<ajouter des liaisons de résultats>** fonctionnalité du concepteur EF.
- Les appels de procédure stockée sont mappés 1:1 avec les entités dans le modèle conceptuel. Par exemple, si vous implémentez une hiérarchie d'héritage dans votre modèle conceptuel, puis mappez les procédures stockées CUD pour les entités **parent** (base) et **enfant** (dérivées), l'enregistrement des modifications **enfants** appellera uniquement les procédures stockées de l'**enfant**, mais pas les appels des procédures stockées du **parent**.

## Configuration requise

Pour exécuter cette procédure pas à pas, vous avez besoin des éléments suivants :

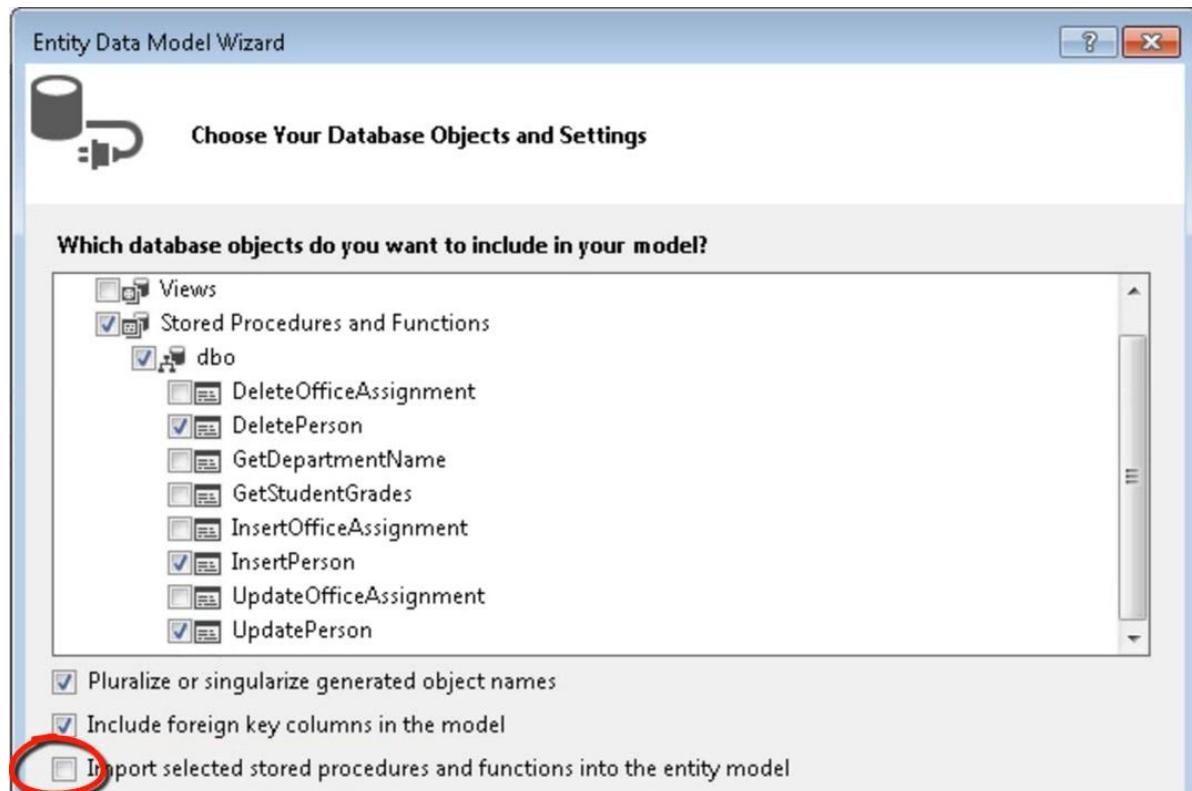
- Une version récente de Visual Studio.
- [Exemple de base de données School](#).

## Configurer le projet

- Ouvrez Visual Studio 2012.
- Sélectionnez **fichier-> nouveau-> projet**
- Dans le volet gauche, cliquez sur **Visual C#** , puis sélectionnez le modèle **console** .
- Entrez **CUDSProcsSample** comme nom.
- Sélectionnez **OK**.

## Créer un modèle

- Dans Explorateur de solutions, cliquez avec le bouton droit sur le nom du projet, puis sélectionnez **Ajouter > nouvel élément**.
- Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet modèles.
- Entrez **CUDSProcs. edmx** comme nom de fichier, puis cliquez sur **Ajouter**.
- Dans la boîte de dialogue choisir le contenu du Model, sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.
- Cliquez sur **nouvelle connexion**. Dans la boîte de dialogue Propriétés de connexion, entrez le nom du serveur (par exemple, (base de données locale )\mssqllocaldb), sélectionnez la méthode d'authentification, tapez **School** pour le nom de la base de données, puis cliquez sur **OK**. La boîte de dialogue choisir votre connexion de données est mise à jour avec votre paramètre de connexion à la base de données.
- Dans la boîte de dialogue choisir vos objets de base de données, sous le nœud **Tables**, sélectionnez la table **Person**.
- En outre, sélectionnez les procédures stockées suivantes sous le nœud **procédures stockées et fonctions** : **DeletePerson, InsertPersonet UpdatePerson**.
- À compter de Visual Studio 2012, le concepteur EF prend en charge l'importation en bloc des procédures stockées. L' **importation des procédures stockées et des fonctions sélectionnées dans le modèle d'entité** est activée par défaut. Étant donné que dans cet exemple nous avons des procédures stockées qui insèrent, mettent à jour et suppriment des types d'entité, nous ne voulons pas les importer et décochent cette case.



- Cliquez sur **Terminer**. Le concepteur EF, qui fournit une aire de conception pour la modification de votre modèle, est affiché.

## Mapper l'entité Person aux procédures stockées

- Cliquez avec le bouton droit sur la **personne** type d'entité, puis sélectionnez **mappage de procédure stockée**.

- Les mappages de procédure stockée s'affichent dans la fenêtre **Détails de mappage**.
- Cliquez <**sélectionnez Insérer une fonction**>. Le champ devient une liste déroulante des procédures stockées dans le modèle de stockage qui peut être mappé aux types d'entité dans le modèle conceptuel. Sélectionnez **InsertPerson** dans la liste déroulante.
- Les mappages par défaut entre les paramètres des procédures stockées et les propriétés d'entité apparaissent. Notez que les flèches indiquent le sens du mappage : les valeurs des propriétés sont fournies aux paramètres des procédures stockées.
- Cliquez sur \*\*<ajouter> de liaison de résultats \*\*.
- Tapez **NewPersonID**, le nom du paramètre retourné par la procédure stockée **InsertPerson**. Veillez à ne pas taper les espaces de début ou de fin.
- Appuyez sur **entrée**.
- Par défaut, **NewPersonID** est mappé à la clé d'entité **PersonID**. Notez qu'une flèche indique le sens du mappage : la valeur de la colonne de résultats est fournie à la propriété.

| Parameter / Column            | Operator | Property                  | Use Original...          | Rows Affected Parameter |
|-------------------------------|----------|---------------------------|--------------------------|-------------------------|
| <b>Functions</b>              |          |                           |                          |                         |
| Insert Using InsertPerson     |          |                           |                          |                         |
| <b>Parameters</b>             |          |                           |                          |                         |
| LastName : nvarchar           | ←        | Lastname : String         | <input type="checkbox"/> |                         |
| FirstName : nvarchar          | ←        | FirstName : String        | <input type="checkbox"/> |                         |
| HireDate : datetime           | ←        | HireDate : DateTime       | <input type="checkbox"/> |                         |
| EnrollmentDate : datetime     | ←        | EnrollmentDate : DateTime | <input type="checkbox"/> |                         |
| Discriminator : nvarchar      | ←        | Discriminator : String    | <input type="checkbox"/> |                         |
| <b>Result Column Bindings</b> |          |                           |                          |                         |
| NewPersonID                   | →        | PersonID : Int32          |                          |                         |

- Cliquez sur <**sélectionnez mettre à jour la fonction**>, puis sélectionnez **UpdatePerson** dans la liste déroulante résultante.
- Les mappages par défaut entre les paramètres des procédures stockées et les propriétés d'entité apparaissent.
- Cliquez sur <**sélectionnez Supprimer la fonction**>, puis sélectionnez **DeletePerson** dans la liste déroulante résultante.
- Les mappages par défaut entre les paramètres des procédures stockées et les propriétés d'entité apparaissent.

Les opérations d'insertion, de mise à jour et de suppression de la **personne** type d'entité sont maintenant mappées à des procédures stockées.

Si vous souhaitez activer le contrôle d'accès concurrentiel lors de la mise à jour ou de la suppression d'une entité avec des procédures stockées, utilisez l'une des options suivantes :

- Utilisez un paramètre de **sortie** pour retourner le nombre de lignes affectées à partir de la procédure stockée et cochez la case **lignes affectées** la case à cocher en regard du nom du paramètre. Si la valeur retournée est zéro lorsque l'opération est appelée, une **OptimisticConcurrencyException** est levée.
- Cochez la case **utiliser la valeur d'origine** à côté d'une propriété que vous souhaitez utiliser pour la vérification de l'accès concurrentiel. Lors d'une tentative de mise à jour, la valeur de la propriété qui a été lue à l'origine dans la base de données sera utilisée lors de l'écriture de données dans la base de données. Si la valeur ne correspond pas à la valeur de la base de données, une **OptimisticConcurrencyException** est levée.

## Utiliser le modèle

Ouvrez le fichier **Program.cs** dans lequel la méthode **main** est définie. Ajoutez le code suivant à la fonction main.

Le code crée un nouvel objet **Person**, puis met à jour l'objet, puis supprime l'objet.

```
using (var context = new SchoolEntities())
{
    var newInstructor = new Person
    {
        FirstName = "Robyn",
        LastName = "Martin",
        HireDate = DateTime.Now,
        Discriminator = "Instructor"
    }

    // Add the new object to the context.
    context.People.Add(newInstructor);

    Console.WriteLine("Added {0} {1} to the context.",
                      newInstructor.FirstName, newInstructor.LastName);

    Console.WriteLine("Before SaveChanges, the PersonID is: {0}",
                      newInstructor.PersonID);

    // SaveChanges will call the InsertPerson sproc.
    // The PersonID property will be assigned the value
    // returned by the sproc.
    context.SaveChanges();

    Console.WriteLine("After SaveChanges, the PersonID is: {0}",
                      newInstructor.PersonID);

    // Modify the object and call SaveChanges.
    // This time, the UpdatePerson will be called.
    newInstructor.FirstName = "Rachel";
    context.SaveChanges();

    // Remove the object from the context and call SaveChanges.
    // The DeletePerson sproc will be called.
    context.People.Remove(newInstructor);
    context.SaveChanges();

    Person deletedInstructor = context.People.
        Where(p => p.PersonID == newInstructor.PersonID).
        FirstOrDefault();

    if (deletedInstructor == null)
        Console.WriteLine("A person with PersonID {0} was deleted.",
                          newInstructor.PersonID);
}
```

- Compilez et exéutez l'application. Le programme produit la sortie suivante \*

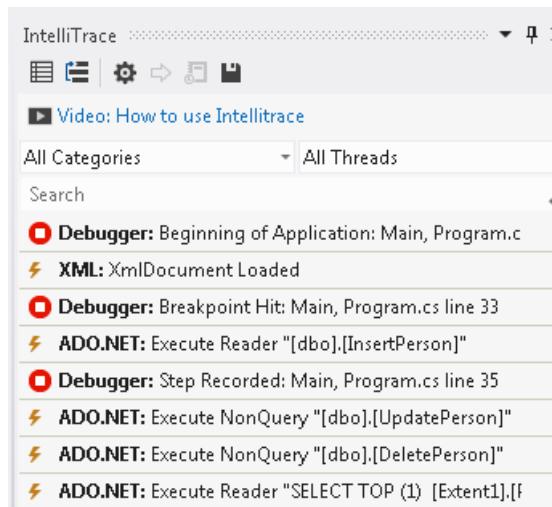
#### NOTE

PersonID est généré automatiquement par le serveur. vous verrez probablement un autre numéro \*

```
Added Robyn Martin to the context.
Before SaveChanges, the PersonID is: 0
After SaveChanges, the PersonID is: 51
A person with PersonID 51 was deleted.
```

Si vous utilisez la version finale de Visual Studio, vous pouvez utiliser IntelliTrace avec le débogueur pour voir les

instructions SQL qui sont exécutées.



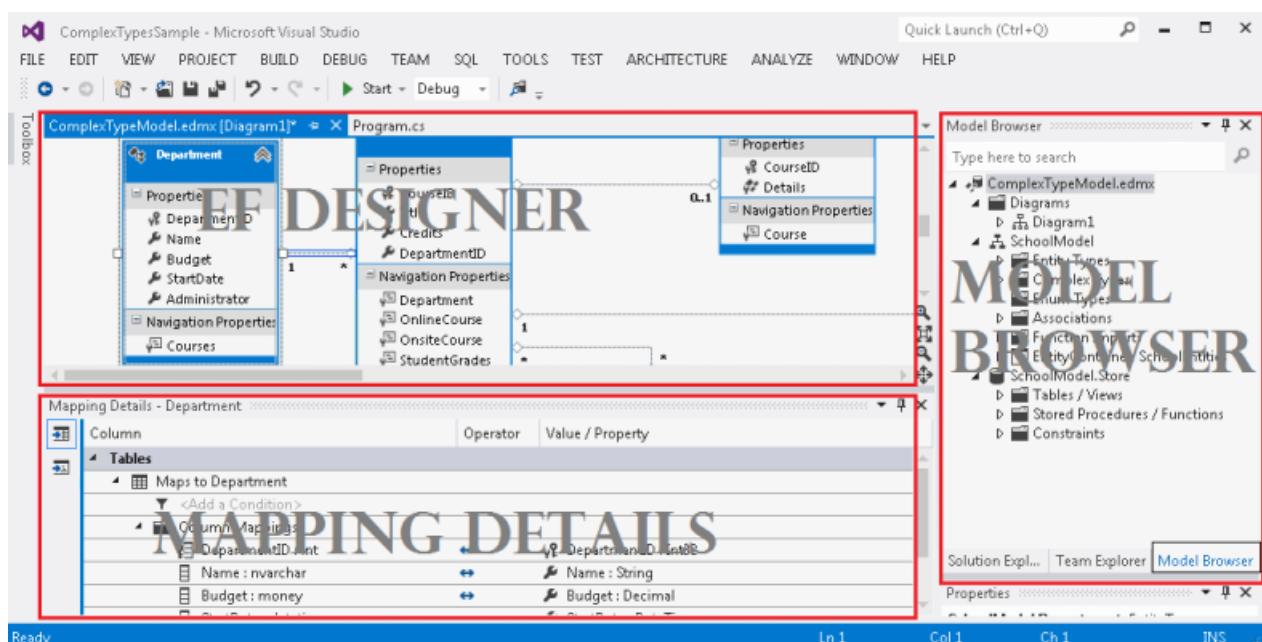
# Relations - Entity Framework Designer

13/09/2018 • 11 minutes to read

## NOTE

Cette page fournit des informations sur la définition des relations dans votre modèle à l'aide du Concepteur EF. Pour obtenir des informations générales sur les relations dans Entity Framework et comment accéder à et manipuler des données à l'aide de relations, consultez [relations & Propriétés de Navigation](#).

Les associations définissent les relations entre les types d'entité dans un modèle. Cette rubrique montre comment mapper des associations avec Entity Framework Designer (Concepteur d'EF). L'illustration suivante montre les principales fenêtres qui sont utilisées lorsque vous travaillez avec le Concepteur EF.



## NOTE

Lorsque vous générez le modèle conceptuel, les avertissements sur les entités non mappées et les associations peuvent apparaître dans la liste d'erreurs. Vous pouvez ignorer ces avertissements, car une fois que vous choisissez de générer la base de données à partir du modèle, les erreurs disparaîtront.

## Vue d'ensemble d'associations

Lorsque vous concevez votre modèle à l'aide de l'Entity Framework Designer, un fichier .edmx représente votre modèle. Dans le fichier .edmx, un **Association** élément définit une relation entre deux types d'entités. Une association doit spécifier les types d'entités impliqués dans la relation et le nombre possible de types d'entités à chaque terminaison de la relation, appelé « **multiplicité** ». La multiplicité d'une terminaison d'association peut avoir une valeur d'un (1), zéro ou un (0.. 1) ou plusieurs (\*). Ces informations sont spécifiées dans les deux enfants **fin** éléments.

Au moment de l'exécution, instances de type d'entité à une extrémité d'une association sont accessible via les propriétés de navigation ou de clés étrangères (si vous choisissez d'exposer les clés étrangères dans vos entités). Avec des clés étrangères exposées, la relation entre les entités est gérée avec un **ReferentialConstraint** élément (un élément enfant de le **Association** élément). Il est recommandé de toujours exposer les clés étrangères pour

les relations dans vos entités.

#### NOTE

Dans plusieurs-à-plusieurs (\*:\*) vous ne pouvez pas ajouter de clés étrangères aux entités. Dans un \*: \* relation, les informations d'association est gérée avec un objet indépendant.

Pour plus d'informations sur les éléments du langage CSDL (**ReferentialConstraint**, **Association**, etc.) consultez le [spécification CSDL](#).

## Créer et supprimer des Associations

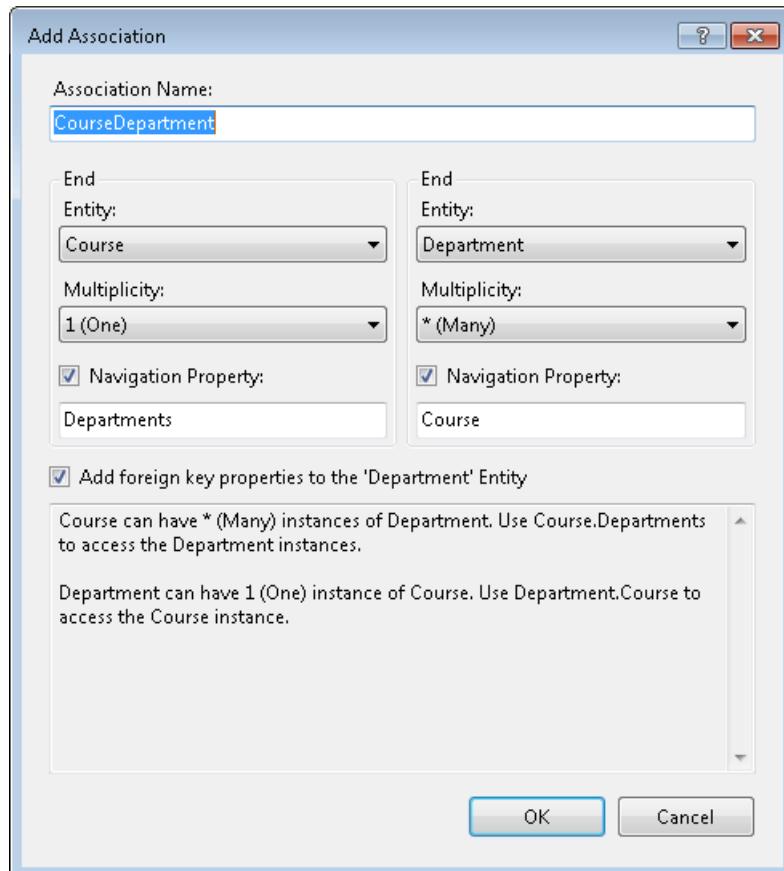
Création d'une association avec les mises à jour du Concepteur EF le contenu du modèle du fichier .edmx. Après avoir créé une association, vous devez créer les mappages pour l'association (décris plus loin dans cette rubrique).

#### NOTE

Cette section suppose que vous avez déjà ajouté les entités que vous souhaitez créer une association entre à votre modèle.

### Pour créer une association

1. Cliquez sur une zone vide de l'aire de conception, pointez sur **Ajouter nouveau**, puis sélectionnez **Association...**
2. Définissez les paramètres pour l'association dans le **ajouter une Association** boîte de dialogue.



#### NOTE

Vous pouvez choisir de ne pas ajouter les propriétés de navigation ou les propriétés de clé étrangère aux entités situées aux terminaisons de l'association en désactivant le \*\* propriété de Navigation \*\* et \*\* ajouter des propriétés de clé étrangères pour la <nom de type d'entité> entité \*\* cases à cocher. Si vous ajoutez une seule propriété de navigation, l'association n'est parcourable que dans une seule direction. Si vous n'ajoutez pas de propriétés de navigation, vous devez ajouter des propriétés de clé étrangère pour accéder aux entités au niveau des terminaisons de l'association.

3. Cliquez sur **OK**.

#### Pour supprimer une association

Pour supprimer une association, procédez comme suit :

- Avec le bouton droit sur le Concepteur EF aire de conception et sélectionnez l'association de **supprimer**.
- OR :
- Sélectionnez une ou plusieurs associations et appuyez sur la touche SUPPR.

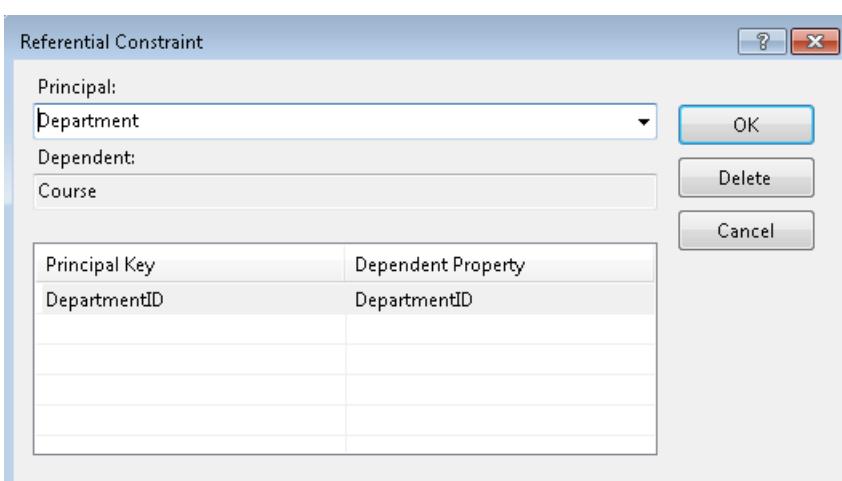
#### Inclure les propriétés de clé étrangère dans vos entités (contraintes référentielles)

Il est recommandé de toujours exposer les clés étrangères pour les relations dans vos entités. Entity Framework utilise une contrainte référentielle pour identifier qu'une propriété agit comme la clé étrangère d'une relation.

Si vous avez coché la **ajouter des propriétés de clé étrangères pour la <nom de type d'entité> entité** case à cocher lorsque vous créez une relation, cette contrainte référentielle a été ajoutée pour vous.

Lorsque vous utilisez le Concepteur EF pour ajouter ou modifier une contrainte référentielle, le Concepteur EF ajoute ou modifie un **ReferentialConstraint** élément dans le contenu CSDL du fichier .edmx.

- Double-cliquez sur l'association à modifier. Le **contrainte référentielle** boîte de dialogue s'affiche.
- À partir de la **Principal** liste déroulante, sélectionnez l'entité principale dans la contrainte référentielle. Propriétés de clé de l'entité sont ajoutées à la **clé du principal du** liste dans la boîte de dialogue.
- À partir de la **dépendants** liste déroulante, sélectionnez l'entité dépendante dans la contrainte référentielle.
- Pour chaque clé principale qui a une clé dépendante, sélectionnez une clé dépendante correspondante dans les listes déroulantes dans la **clé dépendante** colonne.



- Cliquez sur **OK**.

# créer et modifier des mappages d'association

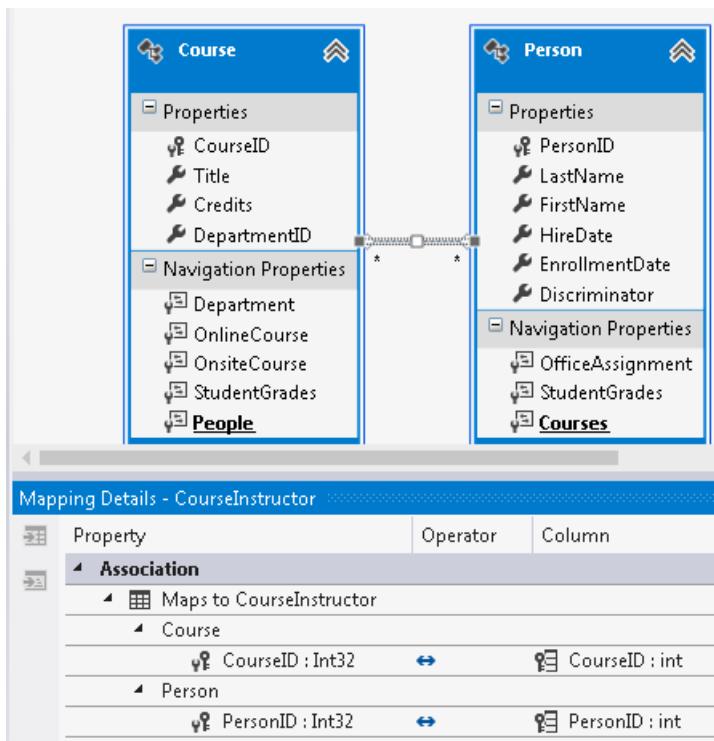
Vous pouvez spécifier comment une association est mappé à la base de données dans le **détails de Mapping** fenêtre du Concepteur EF.

## NOTE

Vous ne pouvez mapper que des détails pour les associations qui n'ont pas une contrainte référentielle spécifiée. Si une contrainte référentielle n'est spécifiée une propriété de clé étrangère est incluse dans l'entité, puis vous pouvez utiliser les détails de mappage pour l'entité au contrôle de la clé étrangère de la colonne qui mappe à.

## Créer un mappage d'association

- Cliquez sur une association dans l'aire de conception et sélectionnez **mappage de Table**. Cette opération affiche le mappage d'association dans le **détails de Mapping** fenêtre.
- Cliquez sur **ajouter une Table ou vue**. Une liste déroulante s'affiche. Elle contient toutes les tables du modèle de stockage.
- Sélectionnez la table à laquelle l'association doit être mappée. Le **détails de Mapping** fenêtre affiche les deux terminaisons de l'association et les propriétés de clé pour le type d'entité à chaque **fin**.
- Pour chaque propriété de clé, cliquez sur le **colonne** champ, puis sélectionnez la colonne à laquelle la propriété doit être mappée.



## Modifier un mappage d'association

- Cliquez sur une association dans l'aire de conception et sélectionnez **mappage de Table**. Cette opération affiche le mappage d'association dans le **détails de Mapping** fenêtre.
- Cliquez sur **est mappé à <nom de la Table>**. Une liste déroulante s'affiche. Elle contient toutes les tables du modèle de stockage.
- Sélectionnez la table à laquelle l'association doit être mappée. Le **détails de Mapping** fenêtre affiche les deux terminaisons de l'association et les propriétés de clé pour le type d'entité à chaque extrémité.
- Pour chaque propriété de clé, cliquez sur le **colonne** champ, puis sélectionnez la colonne à laquelle la propriété doit être mappée.

# Modifier et supprimer les propriétés de Navigation

Propriétés de navigation sont des propriétés de raccourci utilisées pour rechercher les entités situées aux terminaisons d'une association dans un modèle. Il est possible de créer des propriétés de navigation lorsque vous créez une association entre deux types d'entité.

## Pour modifier les propriétés de navigation

- Sélectionnez une propriété de navigation sur l'aire du Concepteur d'Entity Framework. Informations sur la propriété de navigation s'affichent dans Visual Studio **propriétés** fenêtre.
- Modifier les paramètres de propriété dans le **propriétés** fenêtre.

## Supprimer les propriétés de navigation

- Si les clés étrangères ne sont pas exposées sur les types d'entité dans le modèle conceptuel, le fait de supprimer une propriété de navigation peut rendre l'association correspondante parcourable dans une seule direction ou pas parcourable du tout.
- Avec le bouton droit sur le Concepteur EF aire de conception et sélectionnez une propriété de navigation **supprimer**.

# Plusieurs diagrammes par modèle

27/09/2018 • 8 minutes to read

## NOTE

**EF5 et versions ultérieures uniquement** -les fonctionnalités, API, etc. abordés dans cette page ont été introduits dans Entity Framework 5. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Cette page et la vidéo montre comment fractionner un modèle en plusieurs diagrammes à l'aide d'Entity Framework Designer (Concepteur d'EF). Vous souhaiterez peut-être utiliser cette fonctionnalité lorsque votre modèle devient trop volumineux pour afficher ou modifier.

Dans les versions antérieures du Concepteur EF peut uniquement avoir un seul diagramme par le fichier EDMX. À compter de Visual Studio 2012, vous pouvez utiliser le Concepteur EF pour fractionner votre fichier EDMX dans plusieurs diagrammes.

## Regardez la vidéo

Cette vidéo montre comment fractionner un modèle en plusieurs diagrammes à l'aide d'Entity Framework Designer (Concepteur d'EF). Vous souhaiterez peut-être utiliser cette fonctionnalité lorsque votre modèle devient trop volumineux pour afficher ou modifier.

**Présenté par:** Julia Kornich

**Vidéo:** [WMV](#) | [MP4](#) | [WMV \(ZIP\)](#)

## Vue d'ensemble de Concepteur EF

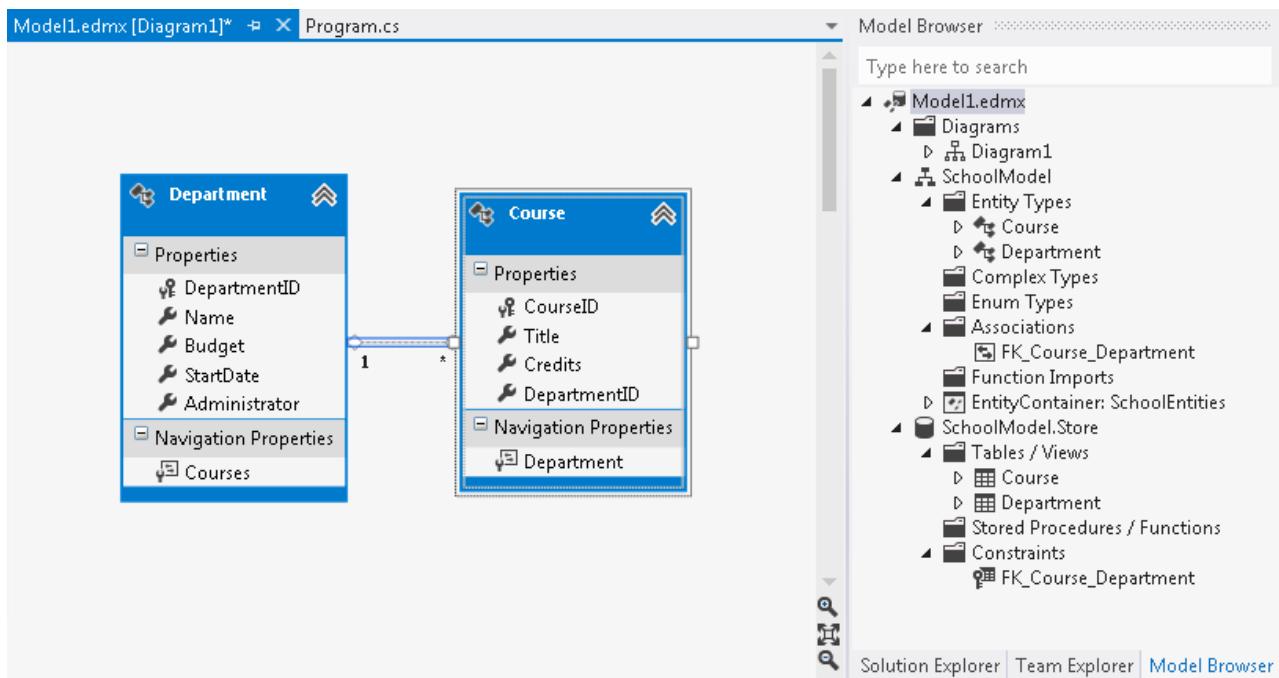
Lorsque vous créez un modèle à l'aide de l'Assistant du Concepteur EF Entity Data Model, un fichier .edmx est créé et ajouté à votre solution. Ce fichier définit la forme de vos entités et comment elles correspondent à la base de données.

Le Concepteur EF comprend les composants suivants :

- Une aire de conception visuelle pour la modification du modèle. Vous pouvez créer, modifier ou supprimer des entités et des associations.
- Un **Explorateur de modèles** fenêtre qui fournit des vues de l'arborescence du modèle. Les entités et leurs associations sont situées sous le `[ModelName]` dossier. Les tables de base de données et les contraintes sont situés sous le `[ModelName]`. Dossier de Store.
- Un **détails de Mapping** fenêtre pour afficher et modifier des mappages. Vous pouvez mapper des types d'entités ou des associations à des tables, colonnes ou procédures stockées de base de données.

La fenêtre de surface de conception visuelle s'ouvre automatiquement à l'issue de l'Assistant Entity Data Model. Si l'Explorateur de modèles n'est pas visible, cliquez sur la principale aire de conception et sélectionnez **Explorateur de modèles**.

La capture d'écran suivante montre un fichier .edmx ouvert dans le Concepteur EF. La capture d'écran montre l'aire de conception visuelle (à gauche) et le **Explorateur de modèles** fenêtre (à droite).



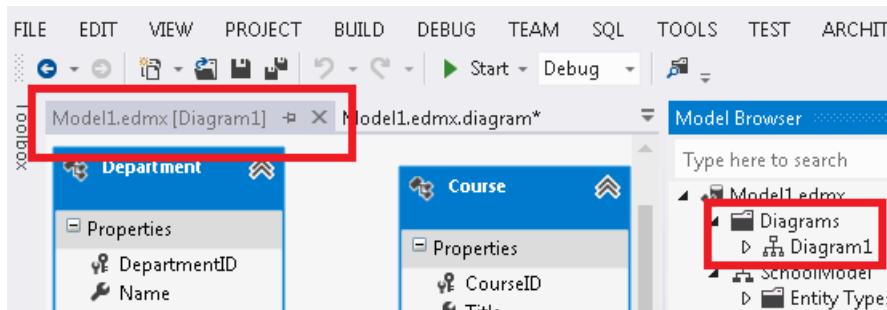
Pour annuler une opération effectuée dans le Concepteur EF, cliquez sur Ctrl-Z.

## Utilisation des diagrammes

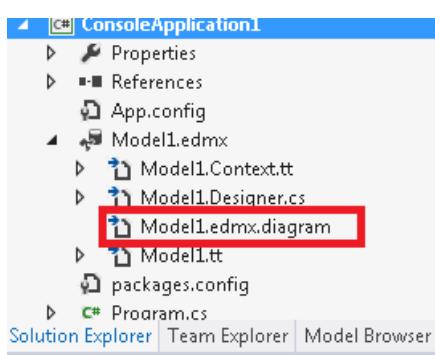
Par défaut, le Concepteur EF crée un diagramme appelé Diagram1. Si vous avez un diagramme composé d'un grand nombre d'entités et associations, sera plus que souhaitez diviser logiquement. À compter de Visual Studio 2012, vous pouvez afficher votre modèle conceptuel dans plusieurs diagrammes.

Lorsque vous ajoutez de nouveaux schémas, ils apparaissent sous le dossier de diagrammes dans la fenêtre Explorateur de modèles. Pour renommer un diagramme : sélectionnez le diagramme dans la fenêtre du navigateur de modèle et cliquez sur une seule fois sur le nom, tapez le nouveau nom. Vous pouvez également cliquer sur le nom de diagramme et sélectionner **renommer**.

Le nom du diagramme s'affiche en regard du nom de fichier .edmx, dans l'éditeur Visual Studio. Par exemple Model1.edmx[Diagram1].



Le contenu de diagrammes (forme et la couleur des entités et associations) est stocké dans le .edmx.diagram fichier. Pour afficher ce fichier, sélectionnez l'Explorateur de solutions et dérouler le fichier .edmx.



Vous ne devez pas modifier le .edmx.diagram de fichiers manuellement, le contenu de ce fichier peut-être remplacé par le Concepteur EF.

## Fractionnement d'entités et Associations dans un nouveau diagramme

Vous pouvez sélectionner les entités sur le schéma existant (maintenez la touche MAJ enfoncée pour sélectionner plusieurs entités). Cliquez sur le bouton droit de la souris et sélectionnez **déplacer vers le nouveau diagramme**. Le nouveau diagramme est créé et les entités sélectionnées et leurs associations sont déplacées vers le diagramme.

Ou bien, vous pouvez cliquez sur le dossier de diagrammes dans l'Explorateur de modèles et sélectionnez **Ajouter nouveau diagramme**. Vous pouvez ensuite glisser les entités figurant dans le dossier de Types d'entités dans l'Explorateur de modèles sur l'aire de conception.

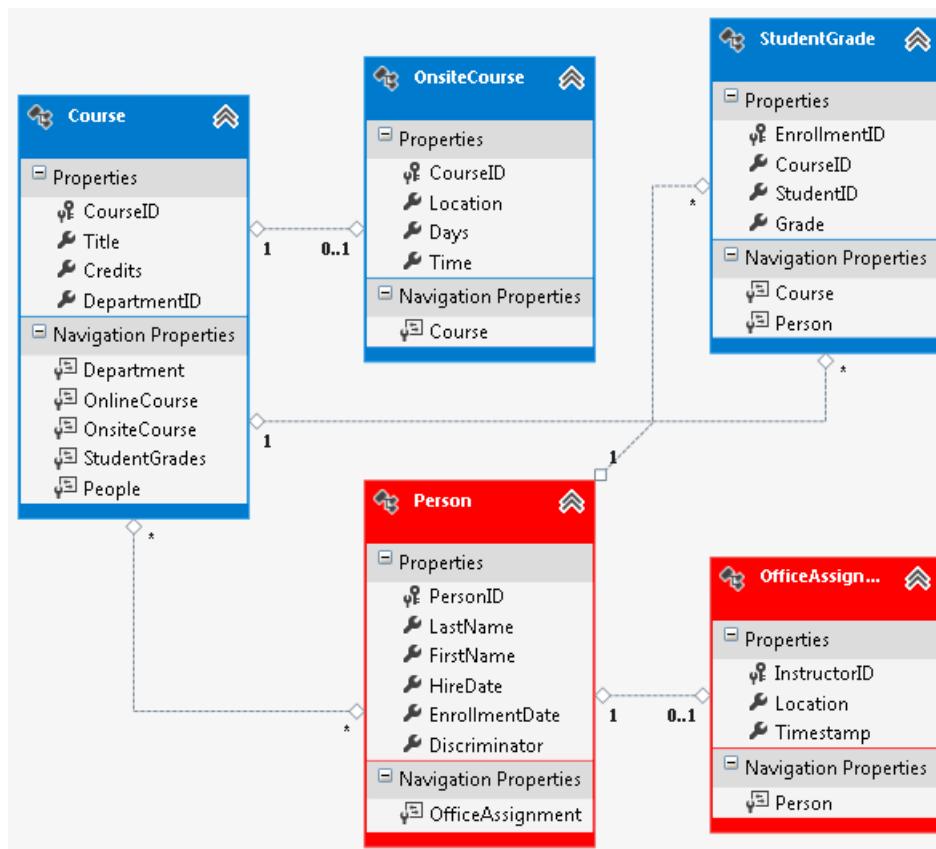
Vous pouvez également couper ou copier des entités (à l'aide des touches Ctrl + X ou Ctrl-C) à partir d'un diagramme et Coller (à l'aide de la clé de Ctrl + V) sur l'autre. Si le diagramme dans lequel vous effectuez un collage une entité déjà contient une entité portant le même nom, une nouvelle entité est créée et ajoutée au modèle. Par exemple : schéma 2 contient l'entité Department. Ensuite, vous collez un autre service sur le diagramme 2. L'entité Department1 est créée et ajoutée au modèle conceptuel.

Pour inclure les entités connexes dans un diagramme, cliquez l'entité et sélectionnez **inclure connexes**. Cela va créer une copie des entités associées et des associations dans le diagramme spécifié.

## Modification de la couleur d'entités

En plus du fractionnement d'un modèle dans plusieurs diagrammes, vous pouvez également modifier les couleurs de vos entités.

Pour modifier la couleur, sélectionnez une entité (ou plusieurs entités) sur l'aire de conception. Ensuite, cliquez sur le bouton droit de la souris et sélectionnez **propriétés**. Dans la fenêtre Propriétés, sélectionnez le **couleur de remplissage** propriété. Spécifiez la couleur à l'aide d'un nom de couleur valide (par exemple, rouge) ou un valide RVB (par exemple, 255, 128, 128).



## Récapitulatif

Dans cette rubrique, nous avons vu comment fractionner un modèle en plusieurs diagrammes et également comment spécifier une couleur différente pour une entité à l'aide d'Entity Framework Designer.

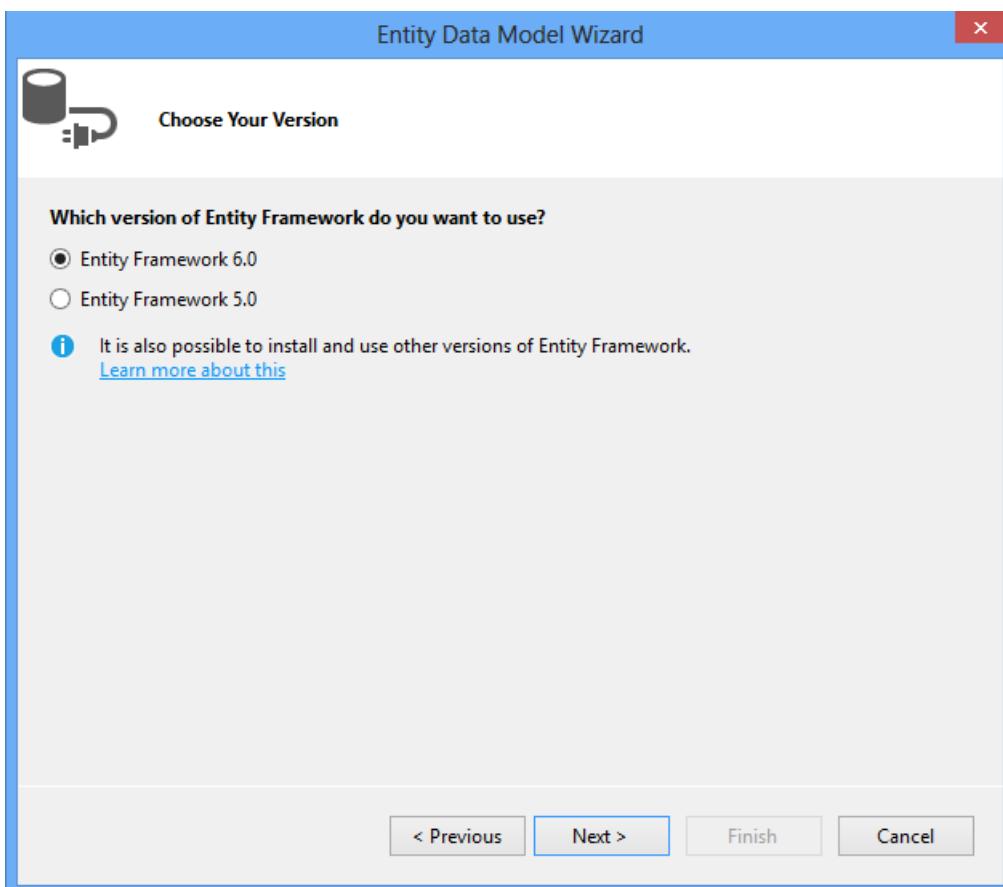
# En sélectionnant la Version du Runtime Entity Framework pour les modèles de Concepteur EF

13/09/2018 • 3 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

L'écran suivant depuis EF6 a été ajoutée au Concepteur d'Entity Framework vous permettent de sélectionner la version du runtime à cibler lors de la création d'un modèle. L'écran s'affiche lorsque la dernière version d'Entity Framework n'est pas déjà installée dans le projet. Si la version la plus récente est déjà installée, il sera uniquement utilisé par défaut.



## Ciblage EF6.x

Vous pouvez choisir d'EF6 à partir de l'écran « Choisir votre Version » pour ajouter le runtime EF6 à votre projet. Une fois que vous avez ajouté EF6, vous allez cesser de se voir cet écran dans le projet actuel.

EF6 va être désactivé si vous avez déjà une version antérieure d'EF installé (étant donné que vous ne pouvez pas cibler plusieurs versions du runtime à partir du même projet). Si l'option d'EF6 n'est pas activée ici, suivez ces étapes pour mettre à niveau votre projet EF6 :

1. Avec le bouton droit sur votre projet dans l'Explorateur de solutions et sélectionnez **gérer les Packages NuGet...**

2. Sélectionnez **mises à jour**
3. Sélectionnez **EntityFramework** (Assurez-vous qu'elle va mettre à jour vers la version souhaitée)
4. Cliquez sur **mise à jour**

## Ciblage EF5.x

Vous pouvez choisir d'EF5 à partir de l'écran « Choisir votre Version » pour ajouter le runtime EF5 à votre projet. Une fois que vous avez ajouté EF5, vous continuerez à voir l'écran avec l'option EF6 désactivée.

Si vous avez une version EF4.x déjà installée du runtime vous verrez que la version d'EF répertorié dans l'écran, plutôt que dans EF5. Dans ce cas, vous pouvez mettre à niveau vers EF5 en procédant comme suit :

1. Sélectionnez **Tools -> Library Package Manager -> Console du Gestionnaire de Package**
2. Exécutez **Install-Package EntityFramework-version 5.0.0**

## Ciblage EF4.x

Vous pouvez installer le runtime EF4.x à votre projet en procédant comme suit :

1. Sélectionnez **Tools -> Library Package Manager -> Console du Gestionnaire de Package**
2. Exécutez **Install-Package EntityFramework-version 4.3.0**

# Modèles de génération de code de concepteur

19/09/2018 • 14 minutes to read

Quand vous créez un modèle à l'aide d'Entity Framework Designer, vos classes et le contexte dérivé sont générés automatiquement pour vous. En plus de la génération de code par défaut, nous fournissons également un nombre de modèles pouvant servir à personnaliser le code généré. Ces modèles sont fournis sous forme de modèles de texte T4, ce qui vous permet de les personnaliser si nécessaire.

Le code généré par défaut dépend de la version de Visual Studio dans laquelle vous avez créé votre modèle :

- Les modèles créés dans Visual Studio 2012 et 2013 génèrent des classes d'entité OCT simples et un contexte qui dérive de la classe DbContext simplifiée.
- Les modèles créés dans Visual Studio 2010 génèrent des classes d'entité qui dérivent d'EntityObject et un contexte qui dérive d'ObjectContext.

## NOTE

Nous vous recommandons de passer au modèle DbContext Generator après l'ajout de votre modèle.

Cette page décrit les modèles disponibles et fournit des instructions pour ajouter un modèle à votre modèle.

## Modèles disponibles

Les modèles suivants sont fournis par l'équipe Entity Framework :

### DbContext Generator

Ce modèle génère des classes d'entité OCT simples et un contexte qui dérive de DbContext à l'aide d'EF6. Il s'agit du modèle recommandé, sauf si vous avez besoin d'utiliser un des autres modèles répertoriés ci-dessous. C'est également le modèle de génération de code que vous obtenez par défaut si vous utilisez les versions récentes de Visual Studio (Visual Studio 2013 et versions ultérieures) : quand vous créez un modèle, ce modèle est utilisé par défaut et les fichiers T4 (.tt) sont imbriqués sous votre fichier .edmx.

### Versions précédentes de Visual Studio

- **Visual Studio 2012** : Pour obtenir les modèles **EF 6.x DbContextGenerator**, vous devez installer la dernière version **d'Entity Framework Tools pour Visual Studio** (consultez la page [Obtenir Entity Framework](#) pour plus d'informations).
- **Visual Studio 2010** : Les modèles **EF 6.x DbContextGenerator** ne sont pas disponibles pour Visual Studio 2010.

### DbContext Generator pour EF 5.x

Si vous utilisez une version antérieure du package NuGet EntityFramework (c'est-à-dire avec une version majeure égale à 5), vous devez utiliser le modèle **EF 5.x DbContext Generator**.

Si vous utilisez Visual Studio 2013 ou 2012, ce modèle est déjà installé.

Si vous utilisez Visual Studio 2010, vous devez sélectionner l'onglet **En ligne** quand vous ajoutez le modèle pour le télécharger à partir de la galerie Visual Studio. Vous pouvez aussi préinstaller le modèle directement à partir de la galerie Visual Studio. Comme les modèles sont inclus dans les versions ultérieures de Visual Studio, les versions de la galerie peuvent uniquement être installées sur Visual Studio 2010.

- [EF 5.x DbContext Generator for C#](#)

- [EF 5.x DbContext Generator for C# Web Sites](#)
- [EF 5.x DbContext Generator for VB.NET](#)
- [EF 5.x DbContext Generator for VB.NET Web Sites](#)

#### DbContext Generator pour EF 4.x

Si vous utilisez une version antérieure du package NuGet EntityFramework (c'est-à-dire avec une version majeure égale à 4), vous devez utiliser le modèle **EF 4.x DbContext Generator**. Il est disponible sous l'onglet **En ligne** quand vous ajoutez le modèle, ou vous pouvez le préinstaller directement à partir de la galerie Visual Studio.

- [EF 4.x DbContext Generator for C#](#)
- [EF 4.x DbContext Generator for C# Web Sites](#)
- [EF 4.x DbContext Generator for VB.NET](#)
- [EF 4.x DbContext Generator for VB.NET Web Sites](#)

#### EntityObject Generator

Ce modèle génère des classes d'entité qui dérivent d'EntityObject et un contexte qui dérive d'ObjectContext.

##### NOTE

Utilisation de DbContext Generator

DbContext Generator est désormais le modèle recommandé pour les nouvelles applications. DbContext Generator tire parti de l'API DbContext plus simple. EntityObject Generator reste disponible pour prendre en charge les applications existantes.

#### Visual Studio 2010, 2012 et 2013

Vous devez sélectionner l'onglet **En ligne** quand vous ajoutez le modèle pour le télécharger à partir de la galerie Visual Studio. Vous pouvez aussi préinstaller le modèle directement à partir de la galerie Visual Studio.

- [EF 6.x EntityObject Generator for C#](#)
- [EF 6.x EntityObject Generator for C# Web Sites](#)
- [EF 6.x EntityObject Generator for VB.NET](#)
- [EF 6.x EntityObject Generator for VB.NET Web Sites](#)

#### EntityObject Generator pour EF 5.x

Si vous utilisez Visual Studio 2012 ou 2013, vous devez sélectionner l'onglet **En ligne** quand vous ajoutez le modèle pour le télécharger à partir de la galerie Visual Studio. Vous pouvez aussi préinstaller le modèle directement à partir de la galerie Visual Studio. Comme les modèles sont inclus dans Visual Studio 2010, les versions de la galerie peuvent uniquement être installées sur Visual Studio 2012 et 2013.

- [EF 5.x EntityObject Generator for C#](#)
- [EF 5.x EntityObject Generator for C# Web Sites](#)
- [EF 5.x EntityObject Generator for VB.NET](#)
- [EF 5.x EntityObject Generator for VB.NET Web Sites](#)

Si vous voulez utiliser la génération de code ObjectContext sans avoir à modifier le modèle, vous pouvez [revenir à la génération de code EntityObject](#).

Si vous utilisez Visual Studio 2010, ce modèle est déjà installé. Si vous créez un modèle dans Visual Studio 2010, il est utilisé par défaut, mais les fichiers .tt ne sont pas inclus dans votre projet. Pour personnaliser le modèle, vous devez l'ajouter à votre projet.

#### Self-Tracking Entities (STE) Generator

Ce modèle génère des classes d'entité de suivi automatique et un contexte qui dérive d'ObjectContext. Dans une application EF, c'est le contexte qui est chargé de suivre les changements des entités. Toutefois, dans les scénarios multicouches, le contexte risque de ne pas être disponible sur la couche qui modifie les entités. Les entités de suivi automatique vous aident à suivre les changements de n'importe quelle couche. Pour plus d'informations, consultez [Entités de suivi automatique](#).

**NOTE**

Modèle des entités de suivi automatique non recommandé

Nous ne recommandons plus d'utiliser le modèle des entités de suivi automatique dans les nouvelles applications, mais il reste disponible pour prendre en charge les applications existantes. Consultez [l'article sur les entités déconnectées](#) afin de connaître les autres options que nous recommandons pour les scénarios multicouches.

**NOTE**

Le modèle des entités de suivi automatique n'a pas de version EF 6.x.

**NOTE**

Le modèle des entités de suivi automatique n'a pas de version Visual Studio 2013.

## Visual Studio 2012

Si vous utilisez Visual Studio 2012, vous devez sélectionner l'onglet **En ligne** quand vous ajoutez le modèle pour le télécharger à partir de la galerie Visual Studio. Vous pouvez aussi préinstaller le modèle directement à partir de la galerie Visual Studio. Comme les modèles sont inclus dans Visual Studio 2010, les versions de la galerie peuvent uniquement être installées sur Visual Studio 2012.

- [EF 5.x STE Generator for C#](#)
- [EF 5.x STE Generator for C# Web Sites](#)
- [EF 5.x STE Generator for VB.NET](#)
- [EF 5.x STE Generator for VB.NET Web Sites](#)

## Visual Studio 2010\*\*

Si vous utilisez Visual Studio 2010, ce modèle est déjà installé.

## POCO Entity Generator

Ce modèle génère des classes d'entité OCT et un contexte qui dérive d'ObjectContext

**NOTE**

Utilisation de DbContext Generator

DbContext Generator est désormais le modèle recommandé pour générer des classes OCT dans les nouvelles applications. DbContext Generator tire parti de la nouvelle API DbContext et peut générer des classes OCT plus simples. POCO Entity Generator reste disponible pour prendre en charge les applications existantes.

**NOTE**

Le modèle OCT n'a pas de version EF 5.x ou EF 6.x.

#### NOTE

Le modèle OCT n'a pas de version Visual Studio 2013.

#### Visual Studio 2012 et Visual Studio 2010

Vous devez sélectionner l'onglet **En ligne** quand vous ajoutez le modèle pour le télécharger à partir de la galerie Visual Studio. Vous pouvez aussi préinstaller le modèle directement à partir de la galerie Visual Studio.

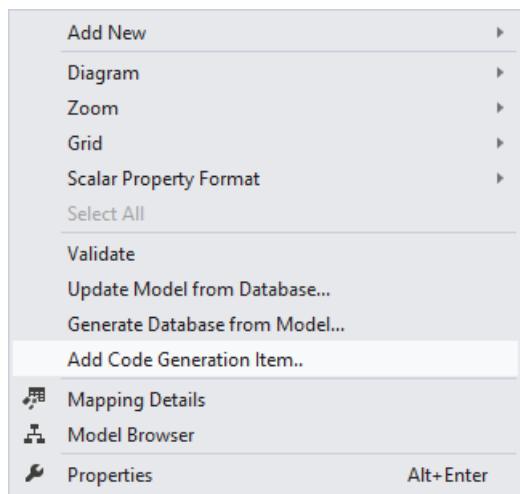
- [EF 4.x POCO Generator for C#](#)
- [EF 4.x POCO Generator for C# Web Sites](#)
- [EF 4.x POCO Generator for VB.NET](#)
- [EF 4.x POCO Generator for VB.NET Web Sites](#)

#### Que sont les modèles « Web Sites »

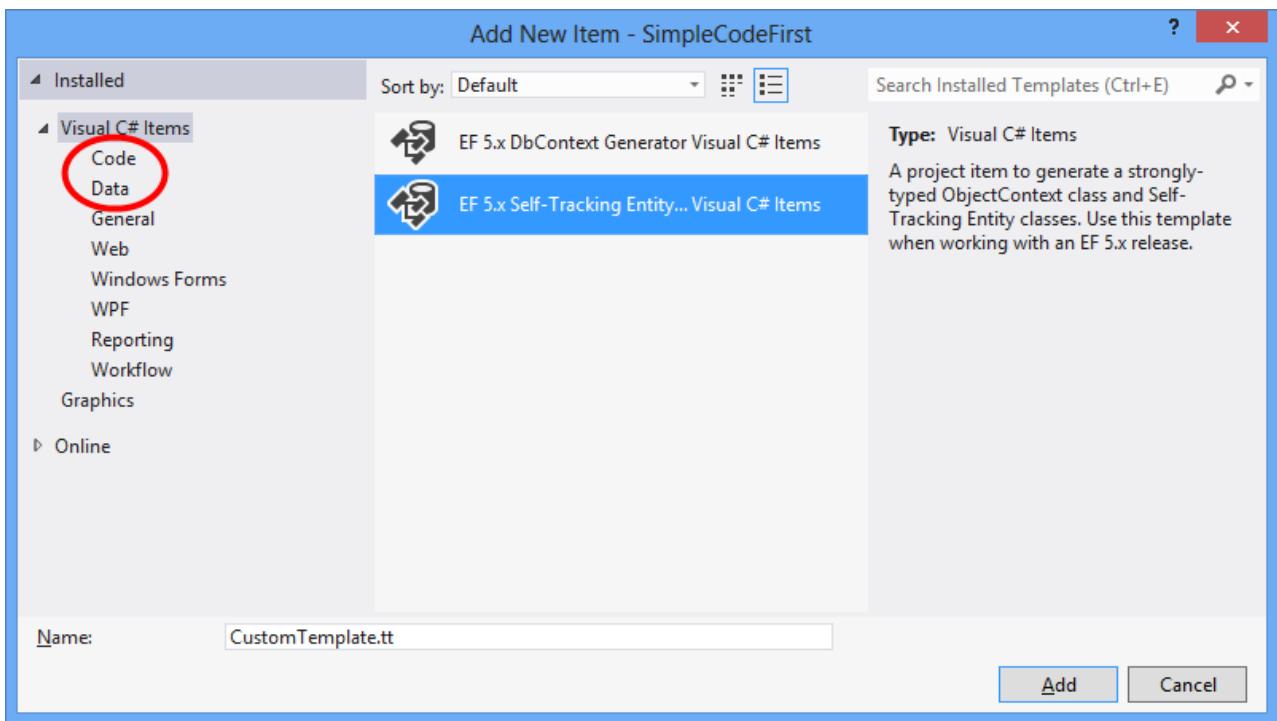
Les modèles « Web Sites » (par ex., **EF 5.x DbContext Generator for C# Web Sites**) sont destinés aux projets de site web créés via **Fichier -> Nouveau -> Site web...**. Ils diffèrent des applications web, créées via **Fichier -> Nouveau -> Projet...**, qui utilisent les modèles standard. Nous fournissons des modèles distincts, car le système de modèle d'élément dans Visual Studio le demande.

## Utilisation d'un modèle

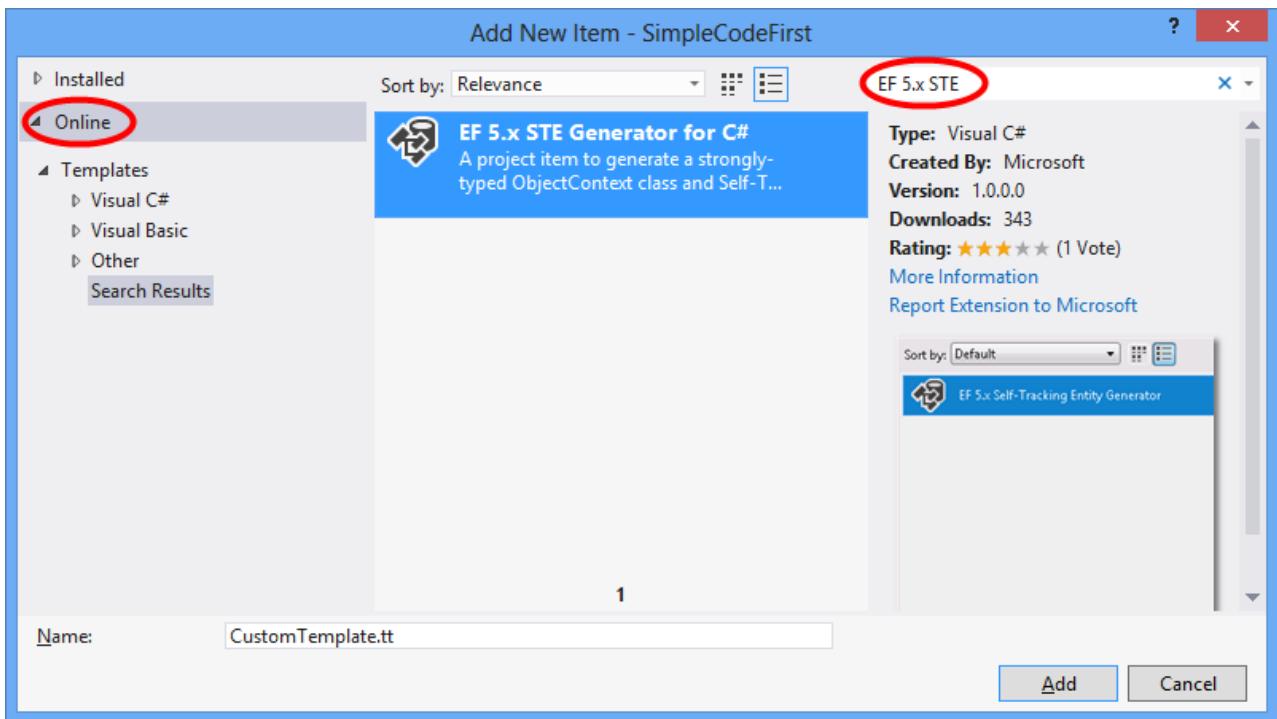
Pour commencer à utiliser un modèle de génération de code, cliquez avec le bouton droit sur un emplacement vide dans l'aire de conception dans EF Designer et sélectionnez **Ajouter un élément de génération de code...**.



Si vous avez déjà installé le modèle à utiliser (ou qu'il a été inclus dans Visual Studio), il est disponible sous la section **Code** ou **Données** dans le menu à gauche.



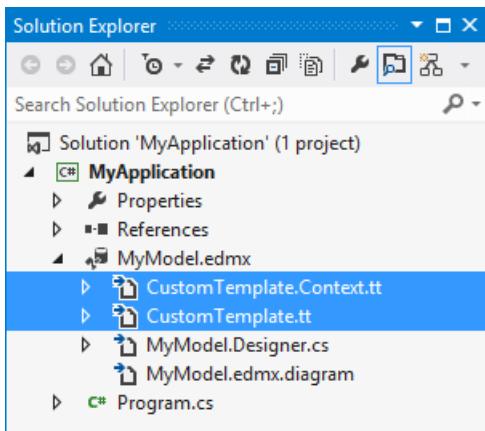
Si vous n'avez pas installé le modèle, sélectionnez **En ligne** dans le menu à gauche et recherchez le modèle souhaité.



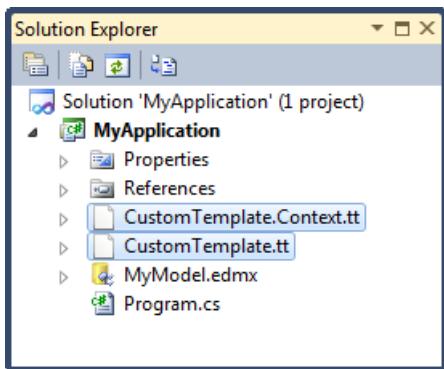
Si vous utilisez Visual Studio 2012, les nouveaux fichiers .tt sont imbriqués sous le fichier .edmx.\*

#### NOTE

Pour les modèles créés dans Visual Studio 2012, vous devez supprimer les modèles utilisés pour la génération de code par défaut, sinon vous obtenez des classes et un contexte en double. Les fichiers par défaut sont <nom du modèle>.tt et <nom du modèle>.context.tt.



Si vous utilisez Visual Studio 2010, les fichiers tt sont ajoutés directement à votre projet.



# Retour à ObjectContext dans Entity Framework Designer

13/09/2018 • 2 minutes to read

Avec une version précédente d'un modèle créé avec le Concepteur EF de Entity Framework génère un contexte dérivé ObjectContext et les classes d'entité dérivé à partir d'EntityObject.

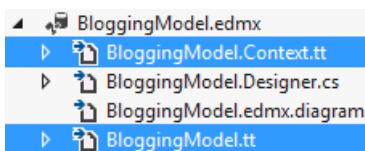
En commençant par EF4.1 recommandé de passer en un modèle de génération de code qui génère un contexte dérivant des classes d'entité DbContext et POCO.

Dans Visual Studio 2012, vous obtenez code DbContext généré par défaut pour tous les nouveaux modèles créés avec le Concepteur EF. Les modèles existants continueront à générer du code d'ObjectContext en fonction, sauf si vous décidez d'échange pour le Générateur de code en fonction de DbContext.

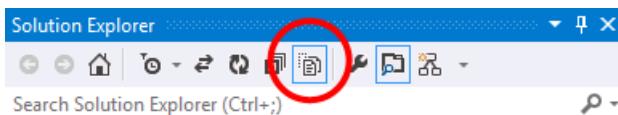
## Retour à la génération de Code d'ObjectContext

### 1. Désactiver la génération de Code de DbContext

Génération des classes DbContext et POCO dérivées est gérée par deux fichiers .tt dans votre projet, si vous développez le fichier .edmx dans l'Explorateur de solutions, vous verrez ces fichiers. Supprimez ces deux fichiers de votre projet.



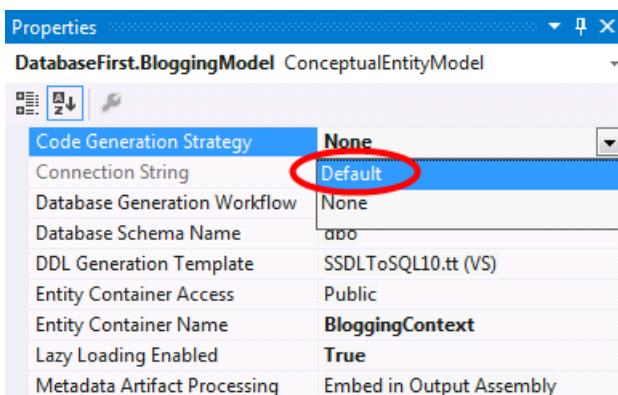
Si vous utilisez VB.NET, vous devrez sélectionner le **afficher tous les fichiers** bouton pour afficher les fichiers imbriqués.



### 2. Réactiver la génération de Code de ObjectContext

Ouvrir dans le Concepteur EF, de modèle avec le bouton droit sur une section vide de l'aire de conception et sélectionnez **propriétés**.

Dans la modification de la fenêtre Propriétés la **stratégie de génération de Code** de aucun à par défaut.



# Spécification CSDL

11/10/2019 • 117 minutes to read

Le langage CSDL (Conceptual Schema Definition Language) est un langage basé sur XML qui décrit les entités, relations et fonctions qui composent un modèle conceptuel d'une application pilotée par les données. Ce modèle conceptuel peut être utilisé par le Entity Framework ou WCF Data Services. Les métadonnées qui sont décrites avec le langage CSDL sont utilisées par le Entity Framework pour mapper les entités et les relations définies dans un modèle conceptuel à une source de données. Pour plus d'informations, consultez [spécification SSDL](#) et [spécification MSL](#).

Le langage CSDL est l'implémentation de la Entity Framework du Entity Data Model.

Dans une application Entity Framework, les métadonnées de modèle conceptuel sont chargées à partir d'un fichier. CSDL (écrit en CSDL) dans une instance de System. Data. Metadata. Edm. EdmItemCollection et sont accessibles à l'aide des méthodes de la Classe System. Data. Metadata. Edm. MetadataWorkspace. Entity Framework utilise les métadonnées du modèle conceptuel pour traduire les requêtes sur le modèle conceptuel en commandes spécifiques à la source de données.

Le concepteur EF stocke les informations de modèle conceptuel dans un fichier. edmx au moment de la conception. Au moment de la génération, le concepteur EF utilise les informations d'un fichier. edmx pour créer le fichier. csdl requis par Entity Framework au moment de l'exécution.

Les versions de CSDL sont différencierées par les espaces de noms XML.

| VERSION CSDL | ESPACE DE NOMS XML  |
|--------------|---|
| CSDL v1      | <a href="https://schemas.microsoft.com/ado/2006/04/edm">https://schemas.microsoft.com/ado/2006/04/edm</a> |
| CSDL v2      | <a href="https://schemas.microsoft.com/ado/2008/09/edm">https://schemas.microsoft.com/ado/2008/09/edm</a> |
| CSDL v3      | <a href="https://schemas.microsoft.com/ado/2009/11/edm">https://schemas.microsoft.com/ado/2009/11/edm</a> |

## Association, élément (CSDL)

Un élément **Association** définit une relation entre deux types d'entités. Une association doit spécifier les types d'entités impliqués dans la relation et le nombre possible de types d'entités à chaque terminaison de la relation, appelé « multiplicité ». La multiplicité d'une terminaison d'association peut avoir la valeur un (1), zéro ou un (0.. 1), ou plusieurs (\*). Ces informations sont spécifiées dans deux éléments finaux enfants.

Il est possible d'accéder aux instances de type d'entité au niveau d'une terminaison d'une association via les propriétés de navigation ou les clés étrangères, si elles sont exposées sur un type d'entité.

Dans une application, une instance d'une association représente une association spécifique entre des instances de types d'entités. Les instances d'association sont regroupées logiquement dans un ensemble d'associations.

Un élément **Association** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- End (exactement 2 éléments)
- ReferentialConstraint (zéro ou un élément)

- Éléments d'annotation (zéro, un ou plusieurs éléments)

## Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **Association**.

| NOM D'ATTRIBUT | REQUIS | VALUE                 |
|----------------|--------|-----------------------|
| <b>Nom</b>     | Oui    | Nom de l'association. |

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Association**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

## Exemple

L'exemple suivant montre un élément **Association** qui définit l'Association **customerOrders** lorsque les clés étrangères n'ont pas été exposées sur les types d'entités **Customer** et **Order**. Les valeurs de **multiplicité** pour chaque **terminaison** de l'Association indiquent que de nombreuses **commandes** peuvent être associées à un **client**, mais qu'un seul **client** peut être associé à une **commande**. En outre, l'élément **OnDelete** indique que toutes les **commandes** associées à un **client** particulier et qui ont été chargées dans ObjectContext seront supprimées si le **client** est supprimé.

```
<Association Name="CustomerOrders">
  <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" >
    <OnDelete Action="Cascade" />
  </End>
  <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
</Association>
```

L'exemple suivant montre un élément **Association** qui définit l'Association **customerOrders** lorsque des clés étrangères ont été exposées sur les types d'entités **Customer** et **Order**. Avec les clés étrangères exposées, la relation entre les entités est gérée à l'aide d'un élément **ReferentialConstraint**. Un élément **AssociationSetMapping** correspondant n'est pas nécessaire pour mapper cette association à la source de données.

```
<Association Name="CustomerOrders">
  <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" >
    <OnDelete Action="Cascade" />
  </End>
  <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customer">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Order">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

## AssociationSet, élément (CSDL)

L'élément **AssociationSet** en Conceptual Schema Definition Language (CSDL) est un conteneur logique pour les instances d'association du même type. Un ensemble d'associations fournit une définition pour le regroupement d'instances d'association afin qu'elles puissent être mappées à une source de données.

L'élément **AssociationSet** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément autorisé)
- End (exactement deux éléments requis)
- Éléments d'annotation (zéro, un ou plusieurs éléments autorisés)

L'attribut **Association** spécifie le type d'association contenu dans un ensemble d'associations. Les jeux d'entités qui composent les terminaisons d'un ensemble d'associations sont spécifiés avec exactement deux éléments **finaux** enfants.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **AssociationSet**.

| NOM D'ATTRIBUT     | REQUIS | VALUE   |
|--------------------|--------|---|
| <b>Nom</b>         | Oui    | Nom du jeu d'entités. La valeur de l'attribut <b>Name</b> ne peut pas être la même que la valeur de l'attribut <b>Association</b> .   |
| <b>Association</b> | Oui    | Nom qualifié complet de l'association dont l'ensemble d'associations contient des instances. L'association doit être dans le même espace de noms que l'ensemble d'associations. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **AssociationSet**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **EntityContainer** avec deux éléments **AssociationSet** :

```

<EntityContainer Name="BooksContainer" >
  <EntityType Name="Books" EntityType="BooksModel.Book" />
  <EntityType Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntityType Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>

```

## Élément CollectionType (CSDL)

L'élément **CollectionType** en Conceptual Schema Definition Language (CSDL) spécifie qu'un paramètre de fonction ou un type de retour de fonction est une collection. L'élément **CollectionType** peut être un enfant de l'élément parameter ou de l'élément ReturnType (Function). Le type de collection peut être spécifié à l'aide de l'attribut **type** ou de l'un des éléments enfants suivants :

- **CollectionType**
- **ReferenceType** ;
- **RowType** ;
- **TypeRef**

### NOTE

Un modèle ne sera pas validé si le type d'une collection est spécifié avec l'attribut **type** et un élément enfant.

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **CollectionType**. Notez que les attributs **DefaultValue**, **MaxLength**, **multiple**, **PRECISION**, **Scale**, **Unicode**et **collation** sont uniquement applicables aux collections de **EDMSimpleTypes**.

| NOM D'ATTRIBUT  | REQUIS | VALUE  |
|---|--------|--|
| <b>Type</b>   | Non    | Type de la collection.   |
| <b>Nullable</b>   | Non    | <b>True</b> (valeur par défaut) ou <b>false</b> selon que la propriété peut avoir une valeur null ou non.<br>[!NOTE] |
| > Dans le CSDL v1, une propriété de type complexe doit avoir<br><code>Nullable="False"</code> . |        |  |
| <b>DefaultValue</b>   | Non    | Valeur par défaut de la propriété.   |

| NOM D'ATTRIBUT    | REQUIS | VALUE  |
|-------------------|--------|--|
| <b>MaxLength</b>  | Non    | Longueur maximale de la valeur de propriété.   |
| <b>Multiple</b>   | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne de longueur fixe.   |
| <b>Précision</b>  | Non    | Précision de la valeur de propriété.   |
| <b>Échelle</b>    | Non    | Échelle de la valeur de propriété.   |
| <b>SRID</b>       | Non    | Identificateur de référence système spatial. Valide uniquement pour les propriétés des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> |
| <b>Unicode</b>    | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne Unicode.  |
| <b>Classement</b> | Non    | Chaîne qui spécifie l'ordre de tri à utiliser dans la source de données.   |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **CollectionType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre une fonction définie par modèle qui utilise un élément **CollectionType** pour spécifier que la fonction retourne une collection de types d'entité **Person** (comme spécifié avec l'attribut **ElementType**).

```
<Function Name="LastNamesAfter">
    <Parameter Name="someString" Type="Edm.String"/>
    <ReturnType>
        <CollectionType ElementType="SchoolModel.Person"/>
    </ReturnType>
    <DefiningExpression>
        SELECT VALUE p
        FROM SchoolEntities.People AS p
        WHERE p.LastName >= someString
    </DefiningExpression>
</Function>
```

L'exemple suivant montre une fonction définie par modèle qui utilise un élément **CollectionType** pour spécifier que la fonction retourne une collection de lignes (comme spécifié dans l'élément **RowType**).

```

<Function Name="LastNamesAfter">
    <Parameter Name="someString" Type="Edm.String" />
    <ReturnType>
        <CollectionType>
            <RowType>
                <Property Name="FirstName" Type="Edm.String" Nullable="false" />
                <Property Name="LastName" Type="Edm.String" Nullable="false" />
            </RowType>
        </CollectionType>
    </ReturnType>
    <DefiningExpression>
        SELECT VALUE ROW(p.FirstName, p.LastName)
        FROM SchoolEntities.People AS p
        WHERE p.LastName &gt;= somestring
    </DefiningExpression>
</Function>

```

L'exemple suivant montre une fonction définie par modèle qui utilise l'élément **CollectionType** pour spécifier que la fonction accepte comme paramètre une collection de types d'entités **Department**.

```

<Function Name="GetAvgBudget">
    <Parameter Name="Departments">
        <CollectionType>
            <TypeRef Type="SchoolModel.Department"/>
        </CollectionType>
    </Parameter>
    <ReturnType Type="Collection(Edm.Decimal)"/>
    <DefiningExpression>
        SELECT VALUE AVG(d.Budget) FROM Departments AS d
    </DefiningExpression>
</Function>

```

## ComplexType, élément (CSDL)

Un élément **complexType** définit une structure de données composée de propriétés **type EDMSSimpleType** ou d'autres types complexes. Un type complexe peut être une propriété d'un type d'entité ou d'un autre type complexe. Un type complexe est semblable à un type d'entité du fait qu'un type complexe définit des données. Toutefois, il existe des différences clés entre les types complexes et les types d'entités :

- Les types complexes n'ont pas d'identité (ni de clés), par conséquent, ils ne peuvent pas exister indépendamment. Les types complexes peuvent uniquement exister en tant que propriétés de types d'entités ou d'autres types complexes.
- Les types complexes ne peuvent pas participer aux associations. Aucune des terminaisons d'une association ne peut être un type complexe ; par conséquent, les propriétés de navigation ne peuvent pas être définies pour des types complexes.
- Une propriété de type complexe ne peut pas avoir de valeur NULL, bien que les propriétés scalaires d'un type complexe puissent chacune être définie sur NULL.

Un élément **complexType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Property (zéro, un ou plusieurs éléments)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **complexType** .

| NOM D'ATTRIBUT   | REQUIS | VALUE  |
|--|--------|--|
| Name   | Oui    | Nom du type complexe. Le nom d'un type complexe ne peut pas être identique au nom d'un autre type complexe, d'un type d'entité ou d'une association qui figure dans l'étendue du modèle. |
| BaseType   | Non    | Nom d'un autre type complexe qui est le type de base du type complexe en cours de définition.<br>[!NOTE]   |
| > Cet attribut n'est pas applicable dans CSDL v1. L'héritage pour les types complexes n'est pas pris en charge dans cette version. |        |  |
| Abstraction  | Non    | <b>True</b> ou <b>false</b> (valeur par défaut), selon que le type complexe est un type abstrait.<br>[!NOTE]   |
| > Cet attribut n'est pas applicable dans CSDL v1. Les types complexes dans cette version ne peuvent pas être des types abstraits.  |        |  |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **complexType** . Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un type complexe, **Address**, avec les propriétés type EDMSimpleType **StreetAddress**, **City**, **StateOrProvince**, **Country** et **PostalCode**.

```
<ComplexType Name="Address" >
  <Property Type="String" Name="StreetAddress" Nullable="false" />
  <Property Type="String" Name="City" Nullable="false" />
  <Property Type="String" Name="StateOrProvince" Nullable="false" />
  <Property Type="String" Name="Country" Nullable="false" />
  <Property Type="String" Name="PostalCode" Nullable="false" />
</ComplexType>
```

Pour définir l' **adresse** de type complexe (ci-dessus) en tant que propriété d'un type d'entité, vous devez déclarer le type de propriété dans la définition du type d'entité. L'exemple suivant illustre la propriété **Address** en tant que

type complexe sur un type d'entité (**éditeur**) :

```
<EntityType Name="Publisher">
  <Key>
    <PropertyRef Name="Id" />
  </Key>
  <Property Type="Int32" Name="Id" Nullable="false" />
  <Property Type="String" Name="Name" Nullable="false" />
  <Property Type="BooksModel.Address" Name="Address" Nullable="false" />
  <NavigationProperty Name="Books" Relationship="BooksModel.PublishedBy"
    FromRole="Publisher" ToRole="Book" />
</EntityType>
```

## DefiningExpression, élément (CSDL)

L'élément **DefiningExpression** en Conceptual Schema Definition Language (CSDL) contient une expression Entity SQL qui définit une fonction dans le modèle conceptuel.

### NOTE

À des fins de validation, un élément **DefiningExpression** peut contenir du contenu arbitraire. Toutefois, Entity Framework lèvera une exception au moment de l'exécution si un élément **DefiningExpression** ne contient pas d'Entity SQL valide.

### Attributs applicables

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **DefiningExpression**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant utilise un élément **DefiningExpression** pour définir une fonction qui retourne le nombre d'années écoulées depuis la publication d'un livre. Le contenu de l'élément **DefiningExpression** est écrit en Entity SQL.

```
<Function Name="GetYearsInPrint" ReturnType="Edm.Int32" >
  <Parameter Name="book" Type="BooksModel.Book" />
  <DefiningExpression>
    Year(CurrentDateTime()) - Year(cast(book.PublishedDate as DateTime))
  </DefiningExpression>
</Function>
```

## Dependent, élément (CSDL)

L'élément **dépendant** en Conceptual Schema Definition Language (CSDL) est un élément enfant de l'élément **ReferentialConstraint** et définit la terminaison dépendante d'une contrainte référentielle. Un élément **ReferentialConstraint** définit des fonctionnalités qui sont semblables à une contrainte d'intégrité référentielle

dans une base de données relationnelle. De la même manière qu'une ou plusieurs colonnes d'une table de base de données peuvent référencer la clé primaire d'une autre table, une ou plusieurs propriétés d'un type d'entité peuvent référencer la clé d'entité d'un autre type d'entité. Le type d'entité référencé est appelé *terminaison principale* de la contrainte. Le type d'entité qui référence la terminaison principale est appelé *terminaison dépendante* de la contrainte. Les éléments **PropertyRef** sont utilisés pour spécifier les clés qui réfèrent à la terminaison principale.

L'élément **dépendant** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- PropertyRef (un ou plusieurs éléments)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **dépendant**.

| NOM D'ATTRIBUT | REQUIS | VALUE   |
|----------------|--------|---|
| Rôle           | Oui    | Nom du type d'entité au niveau de la terminaison dépendante de l'association. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **dépendant**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **ReferentialConstraint** utilisé dans le cadre de la définition de l'Association **PublishedBy**. La propriété **PublisherId** du type **d'entité Book** compose la terminaison dépendante de la contrainte référentielle.

```
<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" />
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

## Documentation, élément (CSDL)

L'élément **documentation** en Conceptual Schema Definition Language (CSDL) peut être utilisé pour fournir des

informations sur un objet défini dans un élément parent. Dans un fichier. edmx, lorsque l'élément **documentation** est un enfant d'un élément qui apparaît sous la forme d'un objet sur l'aire de conception du concepteur EF (tel qu'une entité, une association ou une propriété), le contenu de l'élément de **documentation** s'affiche dans la Fenêtre **Propriétés** de Visual Studio pour l'objet.

L'élément **documentation** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- **Résumé**: Brève description de l'élément parent. (zéro ou un élément).
- **LongDescription**: Description complète de l'élément parent. (zéro ou un élément).
- Éléments d'annotation. (zéro, un ou plusieurs éléments).

### Attributs applicables

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **documentation**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre l'élément de **documentation** en tant qu'élément enfant d'un élément EntityType. Si l'extrait de code ci-dessous se trouve dans le contenu CSDL d'un fichier. edmx, le contenu des éléments **Summary** et **LongDescription** apparaît dans la fenêtre **Propriétés** de Visual Studio lorsque vous cliquez sur le type d'entité **Customer**.

```
<EntityType Name="Customer">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Type="Int32" Name="CustomerId" Nullable="false" />
  <Property Type="String" Name="Name" Nullable="false" />
</EntityType>
```

## End, élément (CSDL)

L'élément **end** en Conceptual Schema Definition Language (CSDL) peut être un enfant de l'élément Association ou de l'élément AssociationSet. Dans chaque cas, le rôle de l'élément de **fin** est différent et les attributs applicables sont différents.

### Élément End comme enfant de l'élément Association

Un élément **end** (en tant qu'enfant de l'élément **Association**) identifie le type d'entité à une terminaison d'une association et le nombre d'instances de type d'entité qui peuvent exister à cette terminaison d'une association. Les terminaisons d'association sont définies dans le cadre d'une association ; une association doit avoir exactement deux terminaisons d'association. Il est possible d'accéder aux instances de type d'entité au niveau de la terminaison d'une association via les propriétés de navigation ou les clés étrangères si elles sont exposées sur un type d'entité.

Un élément **end** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- OnDelete (zéro ou un élément)

- Éléments d'annotation (zéro, un ou plusieurs éléments)

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **final** lorsqu'il est l'enfant d'un élément **Association**.

| NOM D'ATTRIBUT      | REQUIS | VALUE  |
|---------------------|--------|--|
| <b>Type</b>         | Oui    | Nom du type d'entité au niveau de la terminaison de l'association.   |
| <b>Rôle</b>         | Non    | Nom de la terminaison de l'association. Si aucun nom n'est fourni, le nom du type d'entité au niveau de la terminaison de l'association sera utilisé.  |
| <b>Multiplicité</b> | Oui    | <p><b>1, 0.. 1</b> ou * selon le nombre d'instances de type d'entité qui peuvent être à la fin de l'Association.</p> <p><b>1</b> indique qu'il existe exactement une instance de type d'entité au niveau de la terminaison d'association.</p> <p><b>0.. 1</b> indique que zéro ou une instance de type d'entité existe au niveau de la terminaison d'association.</p> <p>* indique qu'il existe zéro, une ou plusieurs instances de type d'entité au niveau de la terminaison d'association.</p> |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **fin**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **Association** qui définit l'Association **customerOrders**. Les valeurs de **multiplicité** pour chaque **terminaison** de l'Association indiquent que de nombreuses **commandes** peuvent être associées à un **client**, mais qu'un seul **client** peut être associé à une **commande**. En outre, l'élément **OnDelete** indique que toutes les **commandes** associées à un **client** particulier et qui ont été chargées dans ObjectContext sont supprimées si le **client** est supprimé.

```
<Association Name="CustomerOrders">
  <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1" />
  <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
    <OnDelete Action="Cascade" />
  </End>
</Association>
```

#### Élément End comme enfant de l'élément AssociationSet

L'élément **end** spécifie une extrémité d'un ensemble d'associations. L'élément **AssociationSet** doit contenir deux

éléments **end**. Les informations contenues dans un élément **end** sont utilisées dans le mappage d'un ensemble d'associations à une source de données.

Un élément **end** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

#### NOTE

Les éléments d'annotation doivent figurer après tous les autres éléments enfants. Les éléments d'annotation sont autorisés uniquement dans CSDL v2 et versions ultérieures.

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **end** lorsqu'il est l'enfant d'un élément

#### AssociationSet .

| NOM D'ATTRIBUT   | REQUIS | VALUE   |
|------------------|--------|---|
| <b>EntitySet</b> | Oui    | Nom de l'élément <b>EntitySet</b> qui définit une extrémité de l'élément <b>AssociationSet</b> parent. L'élément <b>EntitySet</b> doit être défini dans le même conteneur d'entités que l'élément <b>AssociationSet</b> parent. |
| <b>Rôle</b>      | Non    | Nom de la terminaison de l'ensemble d'associations. Si l'attribut <b>role</b> n'est pas utilisé, le nom de la terminaison de l'ensemble d'associations sera le nom du jeu d'entités.  |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **fin**.

Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **EntityContainer** avec deux éléments **AssociationSet**, chacun avec deux éléments **end** :

```

<EntityContainer Name="BooksContainer" >
  <EntityType Name="Books" EntityType="BooksModel.Book" />
  <EntityType Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntityType Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>

```

## EntityContainer, élément (CSDL)

L'élément **EntityContainer** en Conceptual Schema Definition Language (CSDL) est un conteneur logique pour les jeux d'entités, les ensembles d'associations et les importations de fonction. Un conteneur d'entités de modèle conceptuel est mappé à un conteneur d'entités de modèle de stockage par le biais de l'élément **EntityContainerMapping**. Un conteneur d'entités de modèle de stockage décrit la structure de la base de données : les jeux d'entités décrivent les tables, les ensembles d'associations décrivent les contraintes de clé étrangère et les importations de fonction décrivent les procédures stockées dans une base de données.

Un élément **EntityContainer** peut avoir zéro ou un élément documentation. Si un élément de **documentation** est présent, il doit précéder tous les éléments **EntityType**, **AssociationSet** et **FunctionImport**.

Un élément **EntityContainer** peut avoir zéro ou plusieurs des éléments enfants suivants (dans l'ordre indiqué) :

- **EntityType** ;
- **AssociationSet** ;
- **FunctionImport** ;
- éléments d'annotation.

Vous pouvez étendre un élément **EntityContainer** pour inclure le contenu d'un autre **EntityContainer** qui se trouve dans le même espace de noms. Pour inclure le contenu d'un autre **EntityContainer**, dans l'élément **EntityContainer** de référence, définissez la valeur de l'attribut **extends** sur le nom de l'élément **EntityContainer** que vous souhaitez inclure. Tous les éléments enfants de l'élément **EntityContainer** inclus sont traités comme des éléments enfants de l'élément **EntityContainer** de référence.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **using**.

| NOM D'ATTRIBUT | REQUIS | VALUE   |
|----------------|--------|---|
| <b>Nom</b>     | Oui    | Nom du conteneur d'entités.   |
| <b>Dure</b>    | Non    | Le nom d'un autre conteneur d'entités au sein du même espace de noms. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **EntityContainer**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **EntityContainer** qui définit trois jeux d'entités et deux ensembles d'associations.

```
<EntityContainer Name="BooksContainer" >
  <EntityType Name="Books" EntityType="BooksModel.Book" />
  <EntityType Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntityType Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>
```

## EntityType, élément (CSDL)

L'élément **EntityType** dans Conceptual Schema Definition Language est un conteneur logique pour les instances d'un type d'entité et les instances de tout type dérivé de ce type d'entité. La relation entre un type d'entité et un jeu d'entités est analogue à la relation entre une ligne et une table dans une base de données relationnelle. Comme une ligne, un type d'entité définit un jeu de données connexes et, comme une table, un jeu d'entités contient des instances de cette définition. Un jeu d'entités fournit une construction pour le regroupement d'instances du type d'entité, afin qu'elles puissent être mappées aux structures de données associées dans une source de données.

Plusieurs jeux d'entités peuvent être définis pour un type d'entité particulier.

#### NOTE

Le concepteur EF ne prend pas en charge les modèles conceptuels qui contiennent plusieurs jeux d'entités par type.

L'élément **EntityType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Élément documentation (zéro ou un élément autorisé)
- Éléments d'annotation (zéro, un ou plusieurs éléments autorisés)

#### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **EntityType**.

| NOM D'ATTRIBUT    | REQUIS | VALUE  |
|-------------------|--------|--|
| <b>Nom</b>        | Oui    | Nom du jeu d'entités.  |
| <b>EntityType</b> | Oui    | Nom qualifié complet du type d'entité pour lequel le jeu d'entités contient des instances. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **EntitySet**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **EntityContainer** avec trois éléments **EntitySet** :

```
<EntityContainer Name="BooksContainer" >
  <EntitySet Name="Books" EntityType="BooksModel.Book" />
  <EntitySet Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntitySet Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="WrittenBy" Association="BooksModel.WrittenBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>
```

Il est possible de définir des jeux d'entités multiples par type (MEST). L'exemple suivant définit un conteneur d'entités avec deux jeux d'entités pour le type **d'entité Book** :

```
<EntityContainer Name="BooksContainer" >
  <EntitySet Name="Books" EntityType="BooksModel.Book" />
  <EntitySet Name="FictionBooks" EntityType="BooksModel.Book" />
  <EntitySet Name="Publishers" EntityType="BooksModel.Publisher" />
  <EntitySet Name="Authors" EntityType="BooksModel.Author" />
  <AssociationSet Name="PublishedBy" Association="BooksModel.PublishedBy">
    <End Role="Book" EntitySet="Books" />
    <End Role="Publisher" EntitySet="Publishers" />
  </AssociationSet>
  <AssociationSet Name="BookAuthor" Association="BooksModel.BookAuthor">
    <End Role="Book" EntitySet="Books" />
    <End Role="Author" EntitySet="Authors" />
  </AssociationSet>
</EntityContainer>
```

## EntityType, élément (CSDL)

L'élément **EntityType** représente la structure d'un concept de niveau supérieur, tel qu'un client ou une commande, dans un modèle conceptuel. Un type d'entité est un modèle pour des instances de types d'entités dans une application. Chaque modèle contient les informations suivantes :

- Nom unique. (Obligatoire.)
- Clé d'entité définie par une ou plusieurs propriétés. (Obligatoire.)
- Propriétés pour contenir des données. (Facultatif)
- Propriétés de navigation qui permettent de naviguer d'une terminaison d'une association à l'autre terminaison. (Facultatif)

Dans une application, une instance d'un type d'entité représente un objet spécifique (tel qu'un client ou une commande spécifique). Chaque instance d'un type d'entité doit avoir une clé d'entité unique dans un jeu d'entités.

Deux instances de type d'entité sont considérées égales seulement si elles sont du même type et si les valeurs de leurs clés d'entité sont identiques.

Un élément **EntityType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Key (zéro ou un élément)
- Property (zéro, un ou plusieurs éléments)
- NavigationProperty (zéro, un ou plusieurs éléments)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **EntityType**.

| NOM D'ATTRIBUT  | REQUIS | VALUE  |
|---|--------|--|
| <b>Nom</b>  | Oui    | Nom du type d'entité.  |
| <b>BaseType</b>   | Non    | Nom d'un autre type d'entité qui est le type de base du type d'entité en cours de définition.  |
| <b>Abstraction</b>  | Non    | <b>True</b> ou <b>false</b> , selon que le type d'entité est un type abstrait ou non.          |
| <b>OpenType</b>   | Non    | <b>True</b> ou <b>false</b> selon que le type d'entité est un type d'entité ouvert.<br>[!NOTE] |
| > L'attribut <b>OpenType</b> s'applique uniquement aux types d'entité définis dans les modèles conceptuels utilisés avec ADO.NET Data Services. |        |  |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **EntityType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **EntityType** avec trois éléments de **propriété** et deux éléments **NavigationProperty** :

```
<EntityType Name="Book">
    <Key>
        <PropertyRef Name="ISBN" />
    </Key>
    <Property Type="String" Name="ISBN" Nullable="false" />
    <Property Type="String" Name="Title" Nullable="false" />
    <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
    <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
        FromRole="Book" ToRole="Publisher" />
    <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
        FromRole="Book" ToRole="Author" />
</EntityType>
```

## EnumType, élément (CSDL)

L'élément **enumType** représente un type énuméré.

Un élément **enumType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Membre (zéro, un ou plusieurs éléments)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

#### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **enumType**.

| NOM D'ATTRIBUT        | REQUIS | VALUE  |
|-----------------------|--------|--|
| <b>Nom</b>            | Oui    | Nom du type d'entité.  |
| <b>IsFlags</b>        | Non    | <b>True</b> ou <b>false</b> , selon que le type enum peut être utilisé comme un ensemble d'indicateurs. La valeur par défaut est <b>false</b> .  |
| <b>UnderlyingType</b> | Non    | <b>Edm. Byte</b> , <b>Edm. Int16</b> , <b>Edm. Int32</b> , <b>Edm. Int64</b> ou <b>Edm. SByte</b> définissant la plage de valeurs du type. Le type sous-jacent par défaut des éléments d'énumération est <b>Edm. Int32</b> . |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **enumType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **enumType** avec trois éléments **membres** :

```
<EnumType Name="Color" IsFlags="false" UnderlyingTyp="Edm.Byte">
  <Member Name="Red" />
  <Member Name="Green" />
  <Member Name="Blue" />
</EntityType>
```

## Function, élément (CSDL)

L'élément de **fonction** en Conceptual Schema Definition Language (CSDL) est utilisé pour définir ou déclarer des fonctions dans le modèle conceptuel. Une fonction est définie à l'aide d'un élément **DefiningExpression**.

Un élément **Function** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Paramètre (zéro, un ou plusieurs éléments)
- **DefiningExpression** (zéro ou un élément)
- **ReturnType** (**Function**) (zéro ou un élément)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

Un type de retour pour une fonction doit être spécifié avec l'élément **ReturnType** (**Function**) ou **ReturnType** (voir ci-dessous), mais pas les deux. Les types de retour possibles correspondent à tout type **EdmSimpleType**, type d'entité, type complexe, type de ligne ou type **REF** (ou à une collection de l'un de ces types).

#### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **Function**.

| NOM D'ATTRIBUT    | REQUIS | VALUE                          |
|-------------------|--------|--------------------------------|
| <b>Nom</b>        | Oui    | Nom de la fonction.            |
| <b>ReturnType</b> | Non    | Type retourné par la fonction. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément de **fonction**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant utilise un élément **Function** pour définir une fonction qui retourne le nombre d'années écoulées depuis l'embauche d'un formateur.

```
<Function Name="YearsSince" ReturnType="Edm.Int32">
  <Parameter Name="date" Type="Edm.DateTime" />
  <DefiningExpression>
    Year(CurrentDateTime()) - Year(date)
  </DefiningExpression>
</Function>
```

## FunctionImport, élément (CSDL)

L'élément **FunctionImport** en Conceptual Schema Definition Language (CSDL) représente une fonction qui est définie dans la source de données, mais qui est disponible pour les objets via le modèle conceptuel. Par exemple, un élément Function dans le modèle de stockage peut être utilisé pour représenter une procédure stockée dans une base de données. Un élément **FunctionImport** du modèle conceptuel représente la fonction correspondante dans une Entity Framework application et est mappée à la fonction de modèle de stockage à l'aide de l'élément FunctionImportMapping. Lorsque la fonction est appelée dans l'application, la procédure stockée correspondante est exécutée dans la base de données.

L'élément **FunctionImport** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément autorisé)
- Paramètre (zéro, un ou plusieurs éléments autorisés)
- Éléments d'annotation (zéro, un ou plusieurs éléments autorisés)
- ReturnType (FunctionImport) (zéro, un ou plusieurs éléments autorisés)

Un élément de **paramètre** doit être défini pour chaque paramètre que la fonction accepte.

Un type de retour pour une fonction doit être spécifié avec l'élément **ReturnType** (FunctionImport) ou **ReturnType** (voir ci-dessous), mais pas les deux. La valeur du type de retour doit être une collection de type EDMSimpleType, d'EntityType ou de ComplexType.

#### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **FunctionImport**.

| NOM D'ATTRIBUT | REQUIS | VALUE                        |
|----------------|--------|------------------------------|
| <b>Nom</b>     | Oui    | Nom de la fonction importée. |

| NOM D'ATTRIBUT      | REQUIS | VALUE  |
|---------------------|--------|--|
| <b>ReturnType</b>   | Non    | Type retourné par la fonction. N'utilisez pas cet attribut si la fonction ne retourne pas de valeur. Dans le cas contraire, la valeur doit être une collection de ComplexType, EntityType ou type EDMSimpleType.                     |
| <b>EntitySet</b>    | Non    | Si la fonction retourne une collection de types d'entités, la valeur de l' <b>EntitySet</b> doit être le jeu d'entités auquel la collection appartient. Dans le cas contraire, l'attribut <b>EntitySet</b> ne doit pas être utilisé. |
| <b>IsComposable</b> | Non    | Si la valeur est définie sur true, la fonction est composable (fonction table) et peut être utilisée dans une requête LINQ. La valeur par défaut est <b>false</b> .  |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **FunctionImport**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **FunctionImport** qui accepte un paramètre et retourne une collection de types d'entités :

```
<FunctionImport Name="GetStudentGrades"
    EntitySet="StudentGrade"
    ReturnType="Collection(SchoolModel.StudentGrade)">
    <Parameter Name="StudentID" Mode="In" Type="Int32" />
</FunctionImport>
```

## Key, élément (CSDL)

L'élément **Key** est un élément enfant de l'élément **EntityType** et définit une *clé d'entité* (une propriété ou un ensemble de propriétés d'un type d'entité qui détermine l'identité). Les propriétés qui composent une clé d'entité sont choisies au moment du design. Les valeurs des propriétés de clé d'entité doivent identifier de façon unique une instance de type d'entité dans un jeu d'entités au moment de l'exécution. Les propriétés qui composent une clé d'entité doivent être choisies pour garantir l'unicité des instances dans un jeu d'entités. L'élément **Key** définit une clé d'entité en référençant une ou plusieurs des propriétés d'un type d'entité.

L'élément **Key** peut avoir les éléments enfants suivants :

- **PropertyRef** (un ou plusieurs éléments)

- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Key**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple ci-dessous définit un type **d'** entité nommé Book. La clé d'entité est définie en référençant la propriété **ISBN** du type d'entité.

```
<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>
```

La propriété **ISBN** est un bon choix pour la clé d'entité, car un numéro ISBN (International Standard Book Number) identifie de manière unique un livre.

L'exemple suivant montre un type d'entité (**auteur**) qui a une clé d'entité composée de deux propriétés, **nom** et **adresse**.

```
<EntityType Name="Author">
  <Key>
    <PropertyRef Name="Name" />
    <PropertyRef Name="Address" />
  </Key>
  <Property Type="String" Name="Name" Nullable="false" />
  <Property Type="String" Name="Address" Nullable="false" />
  <NavigationProperty Name="Books" Relationship="BooksModel.WrittenBy"
    FromRole="Author" ToRole="Book" />
</EntityType>
```

L'utilisation du **nom** et de l' **adresse** pour la clé d'entité est un choix raisonnable, car il est peu probable que deux auteurs du même nom vivent à la même adresse. Toutefois, ce choix pour une clé d'entité ne garantit pas vraiment l'unicité des clés d'entité dans un jeu d'entités. L'ajout d'une **propriété**, **telle que** la réutilisation, qui pourrait être utilisée pour identifier un auteur de manière unique, est recommandé dans ce cas.

## Member, élément (CSDL)

L'élément **member** est un élément enfant de l'élément enumType et définit un membre du type énuméré.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **FunctionImport**.

| NOM D'ATTRIBUT | REQUIS | VALUE  |
|----------------|--------|--|
| <b>Nom</b>     | Oui    | Nom du membre.   |
| <b>Valeur</b>  | Non    | Valeur du membre. Par défaut, le premier membre a la valeur 0, et la valeur de chaque énumérateur successif est incrémentée de 1. Plusieurs membres ayant les mêmes valeurs peuvent exister. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **FunctionImport**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **enumType** avec trois éléments **membres** :

```
<EnumType Name="Color">
  <Member Name="Red" Value="1"/>
  <Member Name="Green" Value="3" />
  <Member Name="Blue" Value="5"/>
</EntityType>
```

## NavigationProperty, élément (CSDL)

Un élément **NavigationProperty** définit une propriété de navigation, qui fournit une référence à l'autre terminaison d'une association. Contrairement aux propriétés définies avec l'élément **Property**, les propriétés de navigation ne définissent pas la forme et les caractéristiques des données. Elles fournissent un moyen d'explorer une association entre deux types d'entités.

Notez que les propriétés de navigation sont facultatives sur les deux types d'entités au niveau des terminaisons d'une association. Si vous définissez une propriété de navigation sur un type d'entité au niveau de la terminaison d'une association, il n'est pas nécessaire de définir une propriété de navigation sur le type d'entité à l'autre terminaison de l'association.

Le type de données retourné par une propriété de navigation est déterminé par la multiplicité de sa terminaison d'association distante. Par exemple, supposons qu'une propriété de navigation, **OrdersNavProp**, existe sur un type d'entité **Customer** et navigue vers une association un-à-plusieurs entre **Customer** et **Order**. Étant donné que la terminaison d'association distante pour la propriété de navigation a une multiplicité de plusieurs (\*), son type de données est une collection ( **ordre** ). De même, si une propriété de navigation, **CustomerNavProp**, existe sur le type d'entité **Order** , son type de données est **Customer** , car la multiplicité de la terminaison distante est un (1).

Un élément **NavigationProperty** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)

- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **NavigationProperty**.

| NOM D'ATTRIBUT  | REQUIS | VALUE   |
|-----------------|--------|---|
| <b>Nom</b>      | Oui    | Nom de la propriété de navigation.  |
| <b>Relation</b> | Oui    | Nom d'une association figurant dans l'étendue du modèle.  |
| <b>ToRole</b>   | Oui    | Terminaison de l'association à laquelle la navigation prend fin. La valeur de l'attribut <b>ToRole</b> doit être identique à la valeur de l'un des attributs de <b>rôle</b> définis sur l'une des terminaisons d'Association (définie dans l'élément AssociationEnd). |
| <b>FromRole</b> | Oui    | Terminaison de l'association où la navigation commence. La valeur de l'attribut <b>FromRole</b> doit être identique à la valeur de l'un des attributs de <b>rôle</b> définis sur l'une des terminaisons d'Association (définie dans l'élément AssociationEnd).        |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **NavigationProperty**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant définit un type d'entité (**Book**) avec deux propriétés de navigation (**PublishedBy** et **WrittenBy**) :

```
<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>
```

## OnDelete, élément (CSDL)

L'élément **OnDelete** en Conceptual Schema Definition Language (CSDL) définit le comportement qui est connecté avec une association. Si l'attribut **action** est défini sur **cascade** à une terminaison d'une association, les types d'entités associés à l'autre terminaison de l'Association sont supprimés lorsque le type d'entité de la première terminaison est supprimé. Si l'association entre deux types d'entités est une relation clé primaire-clé primaire, un objet dépendant chargé est supprimé lorsque l'objet principal de l'autre terminaison de l'Association est supprimé, quelle que soit la spécification de **OnDelete**.

### NOTE

L'élément **OnDelete** affecte uniquement le comportement au moment de l'exécution d'une application ; elle n'affecte pas le comportement dans la source de données. Le comportement défini dans la source de données doit être le même que le comportement défini dans l'application.

Un élément **OnDelete** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **OnDelete**.

| NOM D'ATTRIBUT | REQUIS | VALUE   |
|----------------|--------|---|
| Action         | Oui    | <b>Cascade</b> ou <b>None</b> . Si vous disposez d'une <b>cascade</b> , les types d'entités dépendants sont supprimés lorsque le type d'entité principal est supprimé. Si <b>aucune</b> valeur n'est, les types d'entités dépendants ne sont pas supprimés lorsque le type d'entité principal est supprimé. |

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Association**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **Association** qui définit l'Association **customerOrders**. L'élément **OnDelete** indique que toutes les **commandes** associées à un **client** particulier et qui ont été chargées dans **ObjectContext** seront supprimées lors de la suppression du **client**.

```

<Association Name="CustomerOrders">
  <End Type="ExampleModel.Customer" Role="Customer" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Type="ExampleModel.Order" Role="Order" Multiplicity="*" />
</Association>

```

## Parameter, élément (CSDL)

L'élément **Parameter** en Conceptual Schema Definition Language (CSDL) peut être un enfant de l'élément **FunctionImport** ou de l'élément **Function**.

### Application de l'élément **FunctionImport**

Un élément **Parameter** (en tant qu'enfant de l'élément **FunctionImport**) est utilisé pour définir les paramètres d'entrée et de sortie des importations de fonctions déclarées dans le langage CSDL.

L'élément **Parameter** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément autorisé)
- Éléments d'annotation (zéro, un ou plusieurs éléments autorisés)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **Parameter**.

| NOM D'ATTRIBUT   | REQUIS | VALUE  |
|------------------|--------|--|
| <b>Nom</b>       | Oui    | Nom du paramètre.  |
| <b>Type</b>      | Oui    | Type du paramètre. La valeur doit être un <b>type EDMSimpleType</b> ou un type complexe qui se trouve dans l'étendue du modèle.  |
| <b>Mode</b>      | Non    | <b>In</b> , <b>out</b> ou <b>INOUT</b> selon que le paramètre est un paramètre d'entrée, de sortie ou d'entrée/sortie.   |
| <b>MaxLength</b> | Non    | Longueur maximale autorisée du paramètre.  |
| <b>Précision</b> | Non    | Précision du paramètre.  |
| <b>Échelle</b>   | Non    | Échelle du paramètre.  |
| <b>SRID</b>      | Non    | Identificateur de référence système spatial. Valide uniquement pour les paramètres des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> . |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Parameter**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **FunctionImport** avec un élément enfant **Parameter**. La fonction accepte un paramètre d'entrée et retourne une collection de types d'entités.

```
<FunctionImport Name="GetStudentGrades"
    EntitySet="StudentGrade"
    ReturnType="Collection(SchoolModel.StudentGrade)">
    <Parameter Name="StudentID" Mode="In" Type="Int32" />
</FunctionImport>
```

#### Application de l'élément Function

Un élément **Parameter** (en tant qu'enfant de l'élément **Function**) définit des paramètres pour les fonctions qui sont définies ou déclarées dans un modèle conceptuel.

L'élément **Parameter** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- CollectionType (zéro ou un élément)
- ReferenceType (zéro ou un élément)
- RowType (zéro ou un élément)

#### NOTE

Seul l'un des éléments **CollectionType**, **ReferenceType** ou **RowType** peut être un élément enfant d'un élément de **propriété**.

- Éléments d'annotation (zéro, un ou plusieurs éléments autorisés)

#### NOTE

Les éléments d'annotation doivent figurer après tous les autres éléments enfants. Les éléments d'annotation sont autorisés uniquement dans CSDL v2 et versions ultérieures.

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **Parameter**.

| NOM D'ATTRIBUT | REQUIS | VALUE             |
|----------------|--------|-------------------|
| <b>Nom</b>     | Oui    | Nom du paramètre. |

| NOM D'ATTRIBUT      | REQUIS | VALUE  |
|---------------------|--------|--|
| <b>Type</b>         | Non    | Type du paramètre. Un paramètre peut correspondre à l'un quelconque des types suivants (ou à des collections de ces types) :<br><b>Type EDMSimpleType</b><br>type d'entité<br>type complexe<br>type de ligne<br>type référence |
| <b>Nullable</b>     | Non    | <b>True</b> (valeur par défaut) ou <b>false</b> selon que la propriété peut avoir une valeur <b>null</b> ou non.   |
| <b>DefaultValue</b> | Non    | Valeur par défaut de la propriété.   |
| <b>MaxLength</b>    | Non    | Longueur maximale de la valeur de propriété.   |
| <b>Multiple</b>     | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne de longueur fixe.   |
| <b>Précision</b>    | Non    | Précision de la valeur de propriété.   |
| <b>Échelle</b>      | Non    | Échelle de la valeur de propriété.   |
| <b>SRID</b>         | Non    | Identificateur de référence système spatial. Valide uniquement pour les propriétés des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> .                         |
| <b>Unicode</b>      | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne Unicode.  |
| <b>Classement</b>   | Non    | Chaîne qui spécifie l'ordre de tri à utiliser dans la source de données.   |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Parameter**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **Function** qui utilise un élément enfant **Parameter** pour définir un paramètre de fonction.

```

<Function Name="GetYearsEmployed" ReturnType="Edm.Int32">
<Parameter Name="Instructor" Type="SchoolModel.Person" />
<DefiningExpression>
Year(CurrentDateTime()) - Year(cast(Instructor.HireDate as DateTime))
</DefiningExpression>
</Function>

```

## Principal, élément (CSDL)

L'élément **principal** en Conceptual Schema Definition Language (CSDL) est un élément enfant de l'élément **ReferentialConstraint** qui définit la terminaison principale d'une contrainte référentielle. Un élément **ReferentialConstraint** définit des fonctionnalités qui sont semblables à une contrainte d'intégrité référentielle dans une base de données relationnelle. De la même manière qu'une ou plusieurs colonnes d'une table de base de données peuvent référencer la clé primaire d'une autre table, une ou plusieurs propriétés d'un type d'entité peuvent référencer la clé d'entité d'un autre type d'entité. Le type d'entité référencé est appelé *terminaison principale* de la contrainte. Le type d'entité qui référence la terminaison principale est appelé *terminaison dépendante* de la contrainte. Les éléments **PropertyRef** sont utilisés pour spécifier les clés qui sont référencées par la terminaison dépendante.

L'élément **principal** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- **PropertyRef** (un ou plusieurs éléments)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **principal**.

| NOM D'ATTRIBUT | REQUIS | VALUE   |
|----------------|--------|---|
| Rôle           | Oui    | Nom du type d'entité au niveau de la terminaison principale de l'association. |

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **principal**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **ReferentialConstraint** qui fait partie de la définition de l'Association **PublishedBy**. La propriété **ID** du type d'entité du serveur de **publication** compose la terminaison principale de la contrainte référentielle.

```

<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" />
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>

```

## Property, élément (CSDL)

L'élément de **propriété** en Conceptual Schema Definition Language (CSDL) peut être un enfant de l'élément EntityType, de l'élément complexType ou de l'élément RowType.

### Applications des éléments EntityType et ComplexType

Les éléments de **propriété** (en tant qu'enfants des éléments **EntityType** ou **complexType**) définissent la forme et les caractéristiques des données qu'une instance de type d'entité ou une instance de type complexe contiendra. Les propriétés dans un modèle conceptuel sont analogues aux propriétés qui sont définies sur une classe. De même que les propriétés sur une classe définissent la forme de la classe et acheminent des informations sur les objets, les propriétés dans un modèle conceptuel définissent la forme d'un type d'entité et acheminent des informations sur les instances de type d'entité.

L'élément **Property** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Élément documentation (zéro ou un élément autorisé)
- Éléments d'annotation (zéro, un ou plusieurs éléments autorisés)

Les facettes suivantes peuvent être appliquées à un élément **Property** : **Nullable**, **DefaultValue**, **MaxLength**, **multiple**, **Precision**, **Scale**, **Unicode**, **collation**, **ConcurrencyMode**. Les facettes sont des attributs XML qui fournissent des informations sur la manière dont les valeurs de propriété sont stockées dans la banque de données.

#### NOTE

Les facettes ne peuvent être appliquées qu'à des propriétés de type **type EDMSSimpleType**.

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **Property**.

| NOM D'ATTRIBUT | REQUIS | VALUE                |
|----------------|--------|----------------------|
| <b>Nom</b>     | Oui    | Nom de la propriété. |

| NOM D'ATTRIBUT   | REQUIS | VALUE   |
|--|--------|---|
| <b>Type</b>  | Oui    | Type de la valeur de propriété. Le type de valeur de propriété doit être un <b>type EDMSimpleType</b> ou un type complexe (indiqué par un nom qualifié complet) qui se trouve dans l'étendue du modèle. |
| <b>Nullable</b>  | Non    | <b>True</b> (valeur par défaut) ou <b>false</b> selon que la propriété peut avoir une valeur null ou non.<br>[!NOTE]  |
| > Dans le langage CSDL v1, une propriété de type complexe doit avoir <code>Nullable="False"</code> . |        |   |
| <b>DefaultValue</b>  | Non    | Valeur par défaut de la propriété.  |
| <b>MaxLength</b>   | Non    | Longueur maximale de la valeur de propriété.  |
| <b>Multiple</b>  | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne de longueur fixe.  |
| <b>Précision</b>   | Non    | Précision de la valeur de propriété.  |
| <b>Échelle</b>   | Non    | Échelle de la valeur de propriété.  |
| <b>SRID</b>  | Non    | Identificateur de référence système spatial. Valide uniquement pour les propriétés des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> .  |
| <b>Unicode</b>   | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne Unicode.   |
| <b>Classement</b>  | Non    | Chaîne qui spécifie l'ordre de tri à utiliser dans la source de données.  |
| <b>ConcurrencyMode</b>   | Non    | <b>None</b> (valeur par défaut) ou <b>fixed</b> . Si la valeur est définie sur <b>fixed</b> , la valeur de la propriété sera utilisée dans les contrôles d'accès concurrentiel optimiste.               |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliquée à l'élément **Property**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

## Exemple

L'exemple suivant montre un élément **EntityType** avec trois éléments **Property** :

```
<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>
```

L'exemple suivant montre un élément **complexType** avec cinq éléments de **propriété** :

```
<ComplexType Name="Address" >
  <Property Type="String" Name="StreetAddress" Nullable="false" />
  <Property Type="String" Name="City" Nullable="false" />
  <Property Type="String" Name="StateOrProvince" Nullable="false" />
  <Property Type="String" Name="Country" Nullable="false" />
  <Property Type="String" Name="PostalCode" Nullable="false" />
</ComplexType>
```

## Application de l'élément RowType

Les éléments de **propriété** (comme les enfants d'un élément **RowType**) définissent la forme et les caractéristiques des données qui peuvent être passées ou retournées à partir d'une fonction définie par modèle.

L'élément **Property** peut avoir exactement l'un des éléments enfants suivants :

- CollectionType ;
- ReferenceType ;
- RowType ;

L'élément **Property** peut avoir n'importe quel nombre d'éléments d'annotation enfants.

### NOTE

Les éléments d'annotation sont autorisés uniquement dans CSDL v2 et versions ultérieures.

## Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **Property**.

| NOM D'ATTRIBUT | REQUIS | VALUE                           |
|----------------|--------|---------------------------------|
| <b>Nom</b>     | Oui    | Nom de la propriété.            |
| <b>Type</b>    | Oui    | Type de la valeur de propriété. |

| NOM D'ATTRIBUT   | REQUIS | VALUE  |
|--|--------|--|
| <b>Nullable</b>  | Non    | <b>True</b> (valeur par défaut) ou <b>false</b> selon que la propriété peut avoir une valeur null ou non.<br>[!NOTE]   |
| > Dans CSDL v1, une propriété de type complexe doit avoir<br><code>Nullable="False"</code> . |        |  |
| <b>DefaultValue</b>  | Non    | Valeur par défaut de la propriété.   |
| <b>MaxLength</b>   | Non    | Longueur maximale de la valeur de propriété.   |
| <b>Multiple</b>  | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne de longueur fixe.   |
| <b>Précision</b>   | Non    | Précision de la valeur de propriété.   |
| <b>Échelle</b>   | Non    | Échelle de la valeur de propriété.   |
| <b>SRID</b>  | Non    | Identificateur de référence système spatial. Valide uniquement pour les propriétés des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> . |
| <b>Unicode</b>   | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne Unicode.  |
| <b>Classement</b>  | Non    | Chaîne qui spécifie l'ordre de tri à utiliser dans la source de données.   |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Property**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre les éléments de **propriété** utilisés pour définir la forme du type de retour d'une fonction définie par modèle.

```

<Function Name="LastNamesAfter">
  <Parameter Name="someString" Type="Edm.String" />
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="FirstName" Type="Edm.String" Nullable="false" />
        <Property Name="LastName" Type="Edm.String" Nullable="false" />
      </RowType>
    </CollectionType>
  </ReturnType>
  <DefiningExpression>
    SELECT VALUE ROW(p.FirstName, p.LastName)
    FROM SchoolEntities.People AS p
    WHERE p.LastName &gt;= somestring
  </DefiningExpression>
</Function>

```

## PropertyRef, élément (CSDL)

L'élément **PropertyRef** en Conceptual Schema Definition Language (CSDL) fait référence à une propriété d'un type d'entité pour indiquer que la propriété effectuera l'un des rôles suivants :

- Partie de la clé de l'entité (une propriété ou un jeu de propriétés d'un type d'entité qui détermine l'identité). Un ou plusieurs éléments **PropertyRef** peuvent être utilisés pour définir une clé d'entité.
- Terminaison dépendante ou principale d'une contrainte référentielle.

L'élément **PropertyRef** peut uniquement comporter des éléments d'annotation (zéro ou plus) en tant qu'éléments enfants.

### NOTE

Les éléments d'annotation sont autorisés uniquement dans CSDL v2 et versions ultérieures.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **PropertyRef**.

| NOM D'ATTRIBUT | REQUIS | VALUE                           |
|----------------|--------|---------------------------------|
| <b>Nom</b>     | Oui    | Nom de la propriété référencée. |

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **PropertyRef**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple ci-dessous définit un type d'entité (Book). La clé d'entité est définie en référençant la propriété **ISBN** du type d'entité.

```
<EntityType Name="Book">
  <Key>
    <PropertyRef Name="ISBN" />
  </Key>
  <Property Type="String" Name="ISBN" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" />
  <Property Type="Decimal" Name="Revision" Nullable="false" Precision="29" Scale="29" />
  <NavigationProperty Name="Publisher" Relationship="BooksModel.PublishedBy"
    FromRole="Book" ToRole="Publisher" />
  <NavigationProperty Name="Authors" Relationship="BooksModel.WrittenBy"
    FromRole="Book" ToRole="Author" />
</EntityType>
```

Dans l'exemple suivant, deux éléments **PropertyRef** sont utilisés pour indiquer que deux propriétés (**ID** et **PublisherId**) sont les terminaisons principale et dépendante d'une contrainte référentielle.

```
<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" />
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

## Élément ReferenceType (CSDL)

L'élément **ReferenceType** en Conceptual Schema Definition Language (CSDL) spécifie une référence à un type d'entité. L'élément **ReferenceType** peut être un enfant des éléments suivants :

- ReturnType (fonction)
- Paramètre
- CollectionType ;

L'élément **ReferenceType** est utilisé lors de la définition d'un paramètre ou d'un type de retour pour une fonction.

Un élément **ReferenceType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **ReferenceType**.

| NOM D'ATTRIBUT | REQUIS | VALUE                           |
|----------------|--------|---------------------------------|
| Type           | Oui    | Nom du type d'entité référencé. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **ReferenceType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre l'élément **ReferenceType** utilisé comme enfant d'un élément **Parameter** dans une fonction définie par modèle qui accepte une référence à un type d'entité **Person** :

```
<Function Name="GetYearsEmployed" ReturnType="Edm.Int32">
    <Parameter Name="instructor">
        <ReferenceType Type="SchoolModel.Person" />
    </Parameter>
    <DefiningExpression>
        Year(CurrentDateTime()) - Year(cast(instructor.HireDate as DateTime))
    </DefiningExpression>
</Function>
```

L'exemple suivant montre l'élément **ReferenceType** utilisé comme enfant d'un élément **ReturnType** (Function) dans une fonction définie par modèle qui retourne une référence à un type d'entité **Person** :

```
<Function Name="GetPersonReference">
    <Parameter Name="p" Type="SchoolModel.Person" />
    <ReturnType>
        <ReferenceType Type="SchoolModel.Person" />
    </ReturnType>
    <DefiningExpression>
        REF(p)
    </DefiningExpression>
</Function>
```

## ReferentialConstraint, élément (CSDL)

Un élément **ReferentialConstraint** en Conceptual Schema Definition Language (CSDL) définit des fonctionnalités qui sont semblables à une contrainte d'intégrité référentielle dans une base de données relationnelle. De la même manière qu'une ou plusieurs colonnes d'une table de base de données peuvent référencer la clé primaire d'une autre table, une ou plusieurs propriétés d'un type d'entité peuvent référencer la clé d'entité d'un autre type d'entité. Le type d'entité référencé est appelé *terminaison principale* de la contrainte. Le type d'entité qui référence la terminaison principale est appelé *terminaison dépendante* de la contrainte.

Si une clé étrangère exposée sur un type d'entité fait référence à une propriété sur un autre type d'entité, l'élément

**ReferentialConstraint** définit une association entre les deux types d'entité. Étant donné que l'élément **ReferentialConstraint** fournit des informations sur la façon dont deux types d'entité sont associés, aucun élément **AssociationSetMapping** correspondant n'est nécessaire dans le Mapping Specification Language (MSL). Une association entre deux types d'entités qui n'ont pas de clés étrangères exposées doit avoir un élément **AssociationSetMapping** correspondant pour mapper les informations d'association à la source de données.

Si une clé étrangère n'est pas exposée sur un type d'entité, l'élément **ReferentialConstraint** ne peut définir qu'une contrainte de clé primaire-clé primaire entre le type d'entité et un autre type d'entité.

Un élément **ReferentialConstraint** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Principal (exactement un élément)
- Dependent (un seul élément)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

L'élément **ReferentialConstraint** peut avoir n'importe quel nombre d'attributs d'annotation (attributs XML personnalisés). Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **ReferentialConstraint** utilisé dans le cadre de la définition de l'Association **PublishedBy** .

```
<Association Name="PublishedBy">
  <End Type="BooksModel.Book" Role="Book" Multiplicity="*" >
  </End>
  <End Type="BooksModel.Publisher" Role="Publisher" Multiplicity="1" />
  <ReferentialConstraint>
    <Principal Role="Publisher">
      <PropertyRef Name="Id" />
    </Principal>
    <Dependent Role="Book">
      <PropertyRef Name="PublisherId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

## ReturnType (Function), élément (CSDL)

L'élément **ReturnType** (Function) en Conceptual Schema Definition Language (CSDL) spécifie le type de retour pour une fonction définie dans un élément Function. Un type de retour de fonction peut également être spécifié avec un attribut **ReturnType** .

Les types de retour peuvent être n'importe quel **type EDMSSimpleType**, type d'entité, type complexe, type de ligne, type ref ou une collection de l'un de ces types.

Le type de retour d'une fonction peut être spécifié avec l'attribut de **type** de l'élément **ReturnType** (Function), ou avec l'un des éléments enfants suivants :

- CollectionType ;
- ReferenceType ;

- RowType ;

#### NOTE

Un modèle ne sera pas validé si vous spécifiez un type de retour de fonction avec à la fois l'attribut de **type** de l'élément **ReturnType** (Function) et l'un des éléments enfants.

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **ReturnType** (Function).

| NOM D'ATTRIBUT    | REQUIS | VALUE                          |
|-------------------|--------|--------------------------------|
| <b>ReturnType</b> | Non    | Type retourné par la fonction. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **ReturnType** (Function). Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant utilise un élément **Function** pour définir une fonction qui retourne le nombre d'années pendant lequel un livre a été imprimé. Notez que le type de retour est spécifié par l'attribut **type** d'un élément **ReturnType** (Function).

```
<Function Name="GetYearsInPrint">
  <ReturnType Type=="Edm.Int32">
    <Parameter Name="book" Type="BooksModel.Book" />
    <DefiningExpression>
      Year(CurrentDateTime()) - Year(cast(book.PublishedDate as DateTime))
    </DefiningExpression>
  </ReturnType>
</Function>
```

## ReturnType (FunctionImport), élément (CSDL)

L'élément **ReturnType** (FunctionImport) de Conceptual Schema Definition Language (CSDL) spécifie le type de retour pour une fonction définie dans un élément FunctionImport. Un type de retour de fonction peut également être spécifié avec un attribut **ReturnType**.

Les types de retour peuvent être n'importe quelle collection de type d'entité, de type complexe ou de **type EDMSimpleType**.

Le type de retour d'une fonction est spécifié avec l'attribut de **type** de l'élément **ReturnType** (FunctionImport).

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **ReturnType** (FunctionImport).

| NOM D'ATTRIBUT   | REQUIS | VALUE  |
|------------------|--------|--|
| <b>Type</b>      | Non    | Type retourné par la fonction. La valeur doit être une collection de ComplexType, EntityType ou type EDMSimpleType.  |
| <b>EntitySet</b> | Non    | Si la fonction retourne une collection de types d'entités, la valeur de l' <b>EntitySet</b> doit être le jeu d'entités auquel la collection appartient. Dans le cas contraire, l'attribut <b>EntitySet</b> ne doit pas être utilisé. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **ReturnType** (FunctionImport). Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant utilise un **FunctionImport** qui retourne des livres et des serveurs de publication. Notez que la fonction retourne deux jeux de résultats et, par conséquent, deux éléments **ReturnType** (FunctionImport) sont spécifiés.

```
<FunctionImport Name="GetBooksAndPublishers">
  <ReturnType Type=="Collection(BooksModel.Book )" EntitySet="Books">
  <ReturnType Type=="Collection(BooksModel.Publisher)" EntitySet="Publishers">
</FunctionImport>
```

## Élément RowType (CSDL)

Un élément **RowType** en Conceptual Schema Definition Language (CSDL) définit une structure sans nom en tant que paramètre ou type de retour pour une fonction définie dans le modèle conceptuel.

Un élément **RowType** peut être l'enfant des éléments suivants :

- CollectionType ;
- Paramètre
- ReturnType (fonction)

Un élément **RowType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Property (un ou plusieurs)
- Éléments d'annotation (zéro ou plus)

#### Attributs applicables

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **RowType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre une fonction définie par modèle qui utilise un élément **CollectionType** pour spécifier que la fonction retourne une collection de lignes (comme spécifié dans l'élément **RowType**).

```
<Function Name="LastNamesAfter">
  <Parameter Name="someString" Type="Edm.String" />
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="FirstName" Type="Edm.String" Nullable="false" />
        <Property Name="LastName" Type="Edm.String" Nullable="false" />
      </RowType>
    </CollectionType>
  </ReturnType>
  <DefiningExpression>
    SELECT VALUE ROW(p.FirstName, p.LastName)
    FROM SchoolEntities.People AS p
    WHERE p.LastName &gt;= somestring
  </DefiningExpression>
</Function>
```

## Schema, élément (CSDL)

L'élément **Schema** est l'élément racine d'une définition de modèle conceptuel. Il contient des définitions pour les objets, fonctions et conteneurs qui composent un modèle conceptuel.

L'élément **Schema** peut contenir zéro, un ou plusieurs des éléments enfants suivants :

- Using
- EntityContainer
- EntityType
- EnumType
- Association
- ComplexType
- Fonction

Un élément de **schéma** peut contenir zéro ou un élément d'annotation.

#### NOTE

L'élément de **fonction** et les éléments d'annotation sont autorisés uniquement dans CSDL v2 et versions ultérieures.

L'élément **Schema** utilise l'attribut **namespace** pour définir l'espace de noms pour le type d'entité, le type complexe et les objets Association dans un modèle conceptuel. Dans un espace de noms, deux objets ne peuvent pas avoir le même nom. Les espaces de noms peuvent s'étendre sur plusieurs éléments de **schéma** et plusieurs fichiers. csdl.

Un espace de noms de modèle conceptuel est différent de l'espace de noms XML de l'élément de **schéma**. Un espace de noms de modèle conceptuel (tel que défini par l'attribut **namespace**) est un conteneur logique pour les

types d'entité, les types complexes et les types d'association. L'espace de noms XML (indiqué par l'attribut **xmlns**) d'un élément de **schéma** est l'espace de noms par défaut pour les éléments enfants et les attributs de l'élément de **schéma**. Les espaces de noms XML de la forme <https://schemas.microsoft.com/ado/YYYY/MM/edm> (où aaa et MM représentent respectivement une année et un mois) sont réservés au langage CSDL. Des éléments et attributs personnalisés ne peuvent pas être dans des espaces de noms de cette forme.

## Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **Schema**.

| NOM D'ATTRIBUT        | REQUIS | VALUE   |
|-----------------------|--------|---|
| <b>Espace de noms</b> | Oui    | Espace de noms du modèle conceptuel.<br>La valeur de l'attribut d' <b>espace de noms</b> est utilisée pour former le nom qualifié complet d'un type. Par exemple, si un <b>EntityType</b> nommé <i>Customer</i> est dans l'espace de noms simple. example. <i>Model</i> , le nom qualifié complet de l' <b>EntityType</b> est <i>SimpleExampleModel.Customer</i> .<br>Les chaînes suivantes ne peuvent pas être utilisées comme valeur pour l'attribut d' <b>espace de noms</b> : <b>Système</b> , <b>transitoire</b> ou <b>EDM</b> . La valeur de l'attribut d' <b>espace de noms</b> ne peut pas être la même que la valeur de l'attribut d' <b>espace de noms</b> dans l'élément de schéma SSDL. |
| <b>Alias</b>          | Non    | Identificateur utilisé à la place du nom de l'espace de noms. Par exemple, si un <b>EntityType</b> nommé <i>Customer</i> est dans l'espace de noms simple. example. <i>Model</i> et que la valeur de l'attribut <b>alias</b> est <i>Model</i> , vous pouvez utiliser <i>Model.Customer</i> comme nom qualifié complet de l' <b>EntityType</b> .   |

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément de **schéma**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

## Exemple

L'exemple suivant illustre un élément de **schéma** qui contient un élément **EntityContainer**, deux éléments **EntityType** et un élément **Association**.

```

<Schema xmlns="https://schemas.microsoft.com/ado/2009/11/edm"
    xmlns:cg="https://schemas.microsoft.com/ado/2009/11/codegeneration"
    xmlns:store="https://schemas.microsoft.com/ado/2009/11/edm/EntityStoreSchemaGenerator"
    Namespace="ExampleModel" Alias="Self">
    <EntityTypeContainer Name="ExampleModelContainer">
        <EntityTypeSet Name="Customers"
            EntityType="ExampleModel.Customer" />
        <EntityTypeSet Name="Orders" EntityType="ExampleModel.Order" />
        <AssociationSet
            Name="CustomerOrder"
            Association="ExampleModel.CustomerOrders">
            <End Role="Customer" EntitySet="Customers" />
            <End Role="Order" EntitySet="Orders" />
        </AssociationSet>
    </EntityTypeContainer>
    <EntityType Name="Customer">
        <Key>
            <PropertyRef Name="CustomerId" />
        </Key>
        <Property Type="Int32" Name="CustomerId" Nullable="false" />
        <Property Type="String" Name="Name" Nullable="false" />
        <NavigationProperty
            Name="Orders"
            Relationship="ExampleModel.CustomerOrders"
            FromRole="Customer" ToRole="Order" />
    </EntityType>
    <EntityType Name="Order">
        <Key>
            <PropertyRef Name="OrderId" />
        </Key>
        <Property Type="Int32" Name="OrderId" Nullable="false" />
        <Property Type="Int32" Name="ProductId" Nullable="false" />
        <Property Type="Int32" Name="Quantity" Nullable="false" />
        <NavigationProperty
            Name="Customer"
            Relationship="ExampleModel.CustomerOrders"
            FromRole="Order" ToRole="Customer" />
        <Property Type="Int32" Name="CustomerId" Nullable="false" />
    </EntityType>
    <Association Name="CustomerOrders">
        <End Type="ExampleModel.Customer"
            Role="Customer" Multiplicity="1" />
        <End Type="ExampleModel.Order"
            Role="Order" Multiplicity="*" />
        <ReferentialConstraint>
            <Principal Role="Customer">
                <PropertyRef Name="CustomerId" />
            </Principal>
            <Dependent Role="Order">
                <PropertyRef Name="CustomerId" />
            </Dependent>
        </ReferentialConstraint>
    </Association>
</Schema>

```

## Élément TypeRef (CSDL)

L'élément **TypeRef** en Conceptual Schema Definition Language (CSDL) fournit une référence à un type nommé existant. L'élément **TypeRef** peut être un enfant de l'élément **CollectionType**, qui est utilisé pour spécifier qu'une fonction a une collection en tant que paramètre ou type de retour.

Un élément **TypeRef** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **TypeRef**. Notez que les attributs **DefaultValue**, **MaxLength**, **multiple**, **PRECISION**, **Scale**, **Unicode** et **collation** sont uniquement applicables à **EDMSimpleTypes**.

| NOM D'ATTRIBUT   | REQUIS | VALUE  |
|--|--------|--|
| <b>Type</b>  | Non    | Nom du type référencé.   |
| <b>Nullable</b>  | Non    | <b>True</b> (valeur par défaut) ou <b>false</b> selon que la propriété peut avoir une valeur null ou non.<br>[!NOTE]   |
| > Dans CSDL v1, une propriété de type complexe doit avoir<br><code>Nullable="False"</code> . |        |  |
| <b>DefaultValue</b>  | Non    | Valeur par défaut de la propriété.   |
| <b>MaxLength</b>   | Non    | Longueur maximale de la valeur de propriété.   |
| <b>Multiple</b>  | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne de longueur fixe.   |
| <b>Précision</b>   | Non    | Précision de la valeur de propriété.   |
| <b>Échelle</b>   | Non    | Échelle de la valeur de propriété.   |
| <b>SRID</b>  | Non    | Identificateur de référence système spatial. Valide uniquement pour les propriétés des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> . |
| <b>Unicode</b>   | Non    | <b>True</b> ou <b>false</b> selon que la valeur de la propriété sera stockée ou non comme une chaîne Unicode.  |
| <b>Classement</b>  | Non    | Chaîne qui spécifie l'ordre de tri à utiliser dans la source de données.   |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **CollectionType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

## Exemple

L'exemple suivant montre une fonction définie par modèle qui utilise l'élément **TypeRef** (en tant qu'enfant d'un élément **CollectionType**) pour spécifier que la fonction accepte une collection de types d'entités **Department** .

```
<Function Name="GetAvgBudget">
    <Parameter Name="Departments">
        <CollectionType>
            <TypeRef Type="SchoolModel.Department"/>
        </CollectionType>
    </Parameter>
    <ReturnType Type="Collection(Edm.Decimal)"/>
    <DefiningExpression>
        SELECT VALUE AVG(d.Budget) FROM Departments AS d
    </DefiningExpression>
</Function>
```

## Using, élément (CSDL)

L'élément **using** de Conceptual Schema Definition Language (CSDL) importe le contenu d'un modèle conceptuel qui existe dans un espace de noms différent. En définissant la valeur de l'attribut d'**espace de noms**, vous pouvez faire référence à des types d'entités, des types complexes et des types d'associations définis dans un autre modèle conceptuel. Plus d'un élément **using** peut être un enfant d'un élément **Schema** .

### NOTE

L'élément **using** dans CSDL ne fonctionne pas exactement comme une instruction **using** dans un langage de programmation. En important un espace de noms avec une instruction **using** dans un langage de programmation, vous n'affectez pas les objets de l'espace de noms d'origine. Dans le langage CSDL, un espace de noms importé peut contenir un type d'entité dérivé d'un type d'entité figurant dans l'espace de noms d'origine. Cela peut affecter les jeux d'entités déclarés dans l'espace de noms d'origine.

L'élément **using** peut avoir les éléments enfants suivants :

- Documentation (zéro ou un élément autorisé)
- Éléments d'annotation (zéro, un ou plusieurs éléments autorisés)

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **using** .

| NOM D'ATTRIBUT        | REQUIS | VALUE  |
|-----------------------|--------|--|
| <b>Espace de noms</b> | Oui    | Nom de l'espace de noms importé.   |
| <b>Alias</b>          | Oui    | Identificateur utilisé à la place du nom de l'espace de noms. Bien que cet attribut soit obligatoire, il n'est pas nécessaire qu'il soit utilisé à la place du nom de l'espace de noms pour qualifier les noms d'objets. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **using**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant illustre l'**utilisation** de l'élément Using pour importer un espace de noms défini ailleurs. Notez que l'espace de noms de l'élément de schéma indiqué est `BooksModel`. La propriété `Address` sur l'**EntityType** `Publisher` est un type complexe défini dans l'espace de noms `ExtendedBooksModel` (importé avec l'élément **using**).

```
<Schema xmlns="https://schemas.microsoft.com/ado/2009/11/edm"
         xmlns:cg="https://schemas.microsoft.com/ado/2009/11/codegeneration"
         xmlns:store="https://schemas.microsoft.com/ado/2009/11/edm/EntityStoreSchemaGenerator"
         Namespace="BooksModel" Alias="Self">

    <Using Namespace="BooksModel.Extended" Alias="BMExt" />

    <EntityContainer Name="BooksContainer" >
        <EntityType Name="Publishers" EntityType="BooksModel.Publisher" />
    </EntityContainer>

    <EntityType Name="Publisher">
        <Key>
            <PropertyRef Name="Id" />
        </Key>
        <Property Type="Int32" Name="Id" Nullable="false" />
        <Property Type="String" Name="Name" Nullable="false" />
        <Property Type="BMExt.Address" Name="Address" Nullable="false" />
    </EntityType>
</Schema>
```

## Attributs d'annotation (CSDL)

Les attributs d'annotation dans le langage CSDL (Conceptual Schema Definition Language) sont des attributs XML personnalisés dans le modèle conceptuel. En plus d'avoir une structure XML valide, les attributs d'annotation doivent satisfaire les critères suivants :

- Les attributs d'annotation ne doivent pas figurer dans un espace de noms XML réservé pour le langage CSDL.
- Plusieurs attributs d'annotation peuvent être appliqués à un élément CSDL donné.
- Les noms qualifiés complets de deux attributs d'annotation ne doivent pas être identiques.

Les attributs d'annotation peuvent être utilisés pour fournir des métadonnées supplémentaires sur des éléments dans un modèle conceptuel. Vous pouvez accéder aux métadonnées contenues dans les éléments d'annotation au moment de l'exécution à l'aide des classes de l'espace de noms System. Data. Metadata. Edm.

#### Exemple

L'exemple suivant montre un élément **EntityType** avec un attribut d'annotation (**CustomAttribute**). L'exemple fait également apparaître un élément d'annotation appliqué à l'élément de type d'entité.

```

<Schema Namespace="SchoolModel" Alias="Self"
    xmlns:annotation="https://schemas.microsoft.com/ado/2009/02/edm/annotation"
    xmlns="https://schemas.microsoft.com/ado/2009/11/edm">
    <EntityContainer Name="SchoolEntities" annotation:LazyLoadingEnabled="true">
        <EntityType Name="Person" EntityType="SchoolModel.Person" />
    </EntityContainer>
    <EntityType Name="Person" xmlns:p="http://CustomNamespace.com"
        p:CustomAttribute="Data here.">
        <Key>
            <PropertyRef Name="PersonID" />
        </Key>
        <Property Name="PersonID" Type="Int32" Nullable="false"
            annotation:StoreGeneratedPattern="Identity" />
        <Property Name="LastName" Type="String" Nullable="false"
            MaxLength="50" Unicode="true" FixedLength="false" />
        <Property Name="FirstName" Type="String" Nullable="false"
            MaxLength="50" Unicode="true" FixedLength="false" />
        <Property Name="HireDate" Type="DateTime" />
        <Property Name="EnrollmentDate" Type="DateTime" />
        <p:CustomElement>
            Custom metadata.
        </p:CustomElement>
    </EntityType>
</Schema>

```

Le code suivant récupère les métadonnées dans l'attribut d'annotation et les écrit dans la console :

```

EdmItemCollection collection = new EdmItemCollection("School.csdl");
MetadataWorkspace workspace = new MetadataWorkspace();
workspace.RegisterItemCollection(collection);
EdmType contentType;
workspace.TryGetType("Person", "SchoolModel", DataSpace.CSpace, out contentType);
if (contentType.MetadataProperties.Contains("http://CustomNamespace.com:CustomAttribute"))
{
    MetadataProperty annotationProperty =
        contentType.MetadataProperties["http://CustomNamespace.com:CustomAttribute"];
    object annotationValue = annotationProperty.Value;
    Console.WriteLine(annotationValue.ToString());
}

```

Le code ci-dessus suppose que le fichier `School.csdl` se trouve dans le répertoire de sortie du projet et que vous avez ajouté les instructions `Imports` et `Using` suivantes à votre projet :

```
using System.Data.Metadata.Edm;
```

## Éléments d'annotation (CSDL)

Les éléments d'annotation dans le langage CSDL (Conceptual Schema Definition Language) sont des éléments XML personnalisés dans le modèle conceptuel. En plus d'avoir une structure XML valide, les éléments d'annotation doivent satisfaire les critères suivants :

- Les éléments d'annotation ne doivent pas figurer dans un espace de noms XML réservé pour le langage CSDL.

- Plusieurs éléments d'annotation peuvent être des enfants d'un élément CSDL donné.
- Les noms qualifiés complets de deux éléments d'annotation ne doivent pas être identiques.
- Les éléments d'annotation doivent apparaître après tous les autres éléments enfants d'un élément CSDL donné.

Les éléments d'annotation permettent de fournir des métadonnées supplémentaires sur les éléments dans un modèle conceptuel. À partir de la .NET Framework version 4, les métadonnées contenues dans les éléments d'annotation sont accessibles au moment de l'exécution à l'aide des classes de l'espace de noms System.Data.Metadata.Edm.

## Exemple

L'exemple suivant montre un élément **EntityType** avec un élément annotation (**customelement**). L'exemple fait également apparaître un attribut d'annotation appliqué à l'élément de type d'entité.

```
<Schema Namespace="SchoolModel" Alias="Self"
    xmlns:annotation="https://schemas.microsoft.com/ado/2009/02/edm/annotation"
    xmlns="https://schemas.microsoft.com/ado/2009/11/edm">
<EntityContainer Name="SchoolEntities" annotation:LazyLoadingEnabled="true">
    <EntityType Name="Person" EntityType="SchoolModel.Person" />
</EntityContainer>
<EntityType Name="Person" xmlns:p="http://CustomNamespace.com"
    p:CustomAttribute="Data here.">
    <Key>
        <PropertyRef Name="PersonID" />
    </Key>
    <Property Name="PersonID" Type="Int32" Nullable="false"
        annotation:StoreGeneratedPattern="Identity" />
    <Property Name="LastName" Type="String" Nullable="false"
        MaxLength="50" Unicode="true" FixedLength="false" />
    <Property Name="FirstName" Type="String" Nullable="false"
        MaxLength="50" Unicode="true" FixedLength="false" />
    <Property Name="HireDate" Type="DateTime" />
    <Property Name="EnrollmentDate" Type="DateTime" />
    <p:CustomElement>
        Custom metadata.
    </p:CustomElement>
</EntityType>
</Schema>
```

Le code suivant récupère les métadonnées dans l'élément d'annotation et les écrit dans la console :

```
EdmItemCollection collection = new EdmItemCollection("School.csdl");
MetadataWorkspace workspace = new MetadataWorkspace();
workspace.RegisterItemCollection(collection);
EdmType contentType;
workspace.TryGetType("Person", "SchoolModel", DataSpace.CSpace, out contentType);
if (contentType.MetadataProperties.Contains("http://CustomNamespace.com:CustomElement"))
{
    MetadataProperty annotationProperty =
        contentType.MetadataProperties["http://CustomNamespace.com:CustomElement"];
    object annotationValue = annotationProperty.Value;
    Console.WriteLine(annotationValue.ToString());
}
```

Le code ci-dessus suppose que le fichier School.csdl se trouve dans le répertoire de sortie du projet et que vous avez ajouté les instructions `Imports` et `Using` suivantes à votre projet :

```
using System.Data.Metadata.Edm;
```

## Types de modèles conceptuels (CSDL)

Le langage CSDL (Conceptual Schema Definition Language) prend en charge un ensemble de types de données primitifs abstraits, appelés **EDMSimpleTypes**, qui définissent des propriétés dans un modèle conceptuel. Les **EDMSimpleTypes** sont des proxys pour les types de données primitifs pris en charge dans l'environnement de stockage ou d'hébergement.

Le tableau suivant répertorie les types de données primitifs pris en charge par CSDL. Le tableau répertorie également les facettes qui peuvent être appliquées à chaque **type EDMSimpleType**.

| TYPE EDMSIMPLETYPE         | DESCRIPTION   | FACETTES APPLICABLES                      |
|----------------------------|---|---|
| <b>EDM. binaire</b>        | Contient des données binaires.  | MaxLength, FixedLength, Nullable, Default |
| <b>EDM. Boolean</b>        | Contient la valeur <b>true</b> ou <b>false</b> .  | Nullable, Default                         |
| <b>EDM. Byte</b>           | Contient une valeur d'entier 8 bits non signé.  | Precision, Nullable, Default              |
| <b>EDM. DateTime</b>       | Représente une date et une heure.   | Precision, Nullable, Default              |
| <b>EDM. DateTimeOffset</b> | Contient une date et une heure en tant que décalage en minutes par rapport à l'heure GMT. | Precision, Nullable, Default              |
| <b>EDM. Decimal</b>        | Contient une valeur numérique avec une précision et une échelle fixes.                    | Precision, Nullable, Default              |
| <b>EDM. double</b>         | Contient un nombre à virgule flottante avec une précision de 15 chiffres                  | Precision, Nullable, Default              |
| <b>EDM. float</b>          | Contient un nombre à virgule flottante avec une précision de 7 chiffres.                  | Precision, Nullable, Default              |
| <b>EDM. Guid</b>           | Contient un identificateur unique de 16 octets.   | Precision, Nullable, Default              |
| <b>EDM. Int16</b>          | Contient une valeur d'entier 16 bits signé.   | Precision, Nullable, Default              |
| <b>EDM. Int32</b>          | Contient une valeur d'entier 32 bits signé.   | Precision, Nullable, Default              |
| <b>EDM. Int64</b>          | Contient une valeur d'entier 64 bits signé.   | Precision, Nullable, Default              |
| <b>EDM. SByte</b>          | Contient une valeur d'entier 8 bits signé.  | Precision, Nullable, Default              |

| TYPE EDM SIMPLE TYPE                 | DESCRIPTION                      | FACETTES APPLICABLES   |
|--------------------------------------|----------------------------------|--|
| <b>EDM. String</b>                   | Contient des données caractères. | Unicode, FixedLength, MaxLength, Collation, Precision, Nullable, Default |
| <b>EDM. Time</b>                     | Contient une heure.              | Precision, Nullable, Default   |
| <b>EDM. Geography</b>                |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeographyPoint</b>           |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeographyLineString</b>      |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeographyPolygon</b>         |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeographyMultiPoint</b>      |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeographyMultiLineString</b> |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeographyMultiPolygon</b>    |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeographyCollection</b>      |                                  | Nullable, par défaut, SRID   |
| <b>EDM. Geometry</b>                 |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeometryPoint</b>            |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeometryLineString</b>       |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeometryPolygon</b>          |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeometryMultiPoint</b>       |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeometryMultiLineString</b>  |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeometryMultiPolygon</b>     |                                  | Nullable, par défaut, SRID   |
| <b>EDM. GeometryCollection</b>       |                                  | Nullable, par défaut, SRID   |

## Facettes (CSDL)

Les facettes dans le langage CSDL (Conceptual Schema Definition Language) représentent des contraintes sur les propriétés de types d'entités et de types complexes. Les facettes apparaissent comme des attributs XML sur les éléments CSDL suivants :

- Propriété
- TypeRef
- Paramètre

Le tableau ci-dessous décrit les facettes prises en charge dans le langage CSDL. Toutes les facettes sont facultatives. Certaines facettes répertoriées ci-dessous sont utilisées par la Entity Framework lors de la génération d'une base de données à partir d'un modèle conceptuel.

**NOTE**

Pour plus d'informations sur les types de données dans un modèle conceptuel, consultez types de modèles conceptuels (CSDL).

| <b>FACETTE</b>         | <b>DESCRIPTION</b>  | <b>S'APPLIQUE À</b>  | <b>UTILISÉE POUR LA GÉNÉRATION DE BASE DE DONNÉES.</b> | <b>UTILISÉE PAR LE RUNTIME.</b> |
|------------------------|---|--|--|---------------------------------|
| <b>Classement</b>      | Spécifie la table de classement ou ordre de tri à utiliser lors de l'exécution d'opérations de comparaison et de tri sur des valeurs de la propriété.   | <b>EDM. String</b>   | Oui  | Non                             |
| <b>ConcurrencyMode</b> | Indique que la valeur de la propriété doit être utilisée pour des contrôles d'accès concurrentiel optimiste.  | Toutes les propriétés <b>type EDMSimpleType</b>                    | Non  | Oui                             |
| <b>Default</b>         | Spécifie la valeur par défaut de la propriété si aucune valeur n'est fournie en cas d'instanciation.  | Toutes les propriétés <b>type EDMSimpleType</b>                    | Oui  | Oui                             |
| <b>Multiple</b>        | Spécifie si la longueur de la valeur de propriété peut varier.  | <b>Edm. Binary, Edm. String</b>                                    | Oui  | Non                             |
| <b>MaxLength</b>       | Spécifie la longueur maximale de la valeur de propriété.  | <b>Edm. Binary, Edm. String</b>                                    | Oui  | Non                             |
| <b>Nullable</b>        | Spécifie si la propriété peut avoir une valeur <b>null</b> .  | Toutes les propriétés <b>type EDMSimpleType</b>                    | Oui  | Oui                             |
| <b>Précision</b>       | Pour les propriétés de type <b>Decimal</b> , spécifie le nombre de chiffres qu'une valeur de propriété peut avoir. Pour les propriétés de type <b>Time</b> , <b>Date Time</b> et <b>Date Time Offset</b> , spécifie le nombre de chiffres pour la partie fractionnaire des secondes de la valeur de la propriété. | <b>Edm. DateTime, Edm. DateTimeOffset, Edm. Decimal, Edm. Time</b> | Oui  | Non                             |

| Facette | Description  | S'applique à  | Utilisée pour la génération de base de données. | Utilisée par le runtime. |
|---------|--|---|---|--------------------------|
| Échelle | Spécifie le nombre de chiffres à droite de la virgule décimale pour la valeur de propriété.  | EDM. Decimal  | Oui   | Non                      |
| SRID    | Spécifie l'ID du système de référence système spatial. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> . | EDM. Geography,<br>EDM.<br>GeographyPoint,<br>EDM.<br>GeographyLineString,<br>EDM.<br>GeographyPolygon,<br>EDM.<br>GeographyMultiPoint,<br>EDM.<br>GeographyMultiLineString,<br>EDM.<br>GeographyMultiPolygon,<br>EDM.<br>GeographyCollection,<br>EDM. Geometry,<br>EDM.<br>GeometryPoint,<br>EDM.<br>GeometryLineString,<br>EDM.<br>GeometryPolygon,<br>EDM.<br>GeometryMultiPoint,<br>EDM.<br>GeometryMultiLineString,<br>EDM.<br>GeometryMultiPolygon,<br>EDM.<br>GeometryCollection | Non   | Oui                      |
| Unicode | Indique si la valeur de propriété est stockée au format Unicode.   | EDM. String   | Oui   | Oui                      |

#### NOTE

Lors de la génération d'une base de données à partir d'un modèle conceptuel, l'Assistant génération de base de données reconnaît la valeur de l'attribut **StoreGeneratedPattern** sur un élément de **propriété** s'il se trouve dans l'espace de noms suivant : <https://schemas.microsoft.com/ado/2009/02/edm/annotation>. Les valeurs prises en charge pour l'attribut sont

**Identity** et **computed**. La valeur **Identity** produit une colonne de base de données avec une valeur d'identité générée dans la base de données. Une valeur **calculée** génère une colonne avec une valeur qui est calculée dans la base de données.

#### Exemple

L'exemple suivant illustre l'application de facettes aux propriétés d'un type d'entité :

```
<EntityType Name="Product">
  <Key>
    <PropertyRef Name="ProductId" />
  </Key>
  <Property Type="Int32"
    Name="ProductId" Nullable="false"
    a:StoreGeneratedPattern="Identity"
    xmlns:a="https://schemas.microsoft.com/ado/2009/02/edm/annotation" />
  <Property Type="String"
    Name="ProductName"
    Nullable="false"
    MaxLength="50" />
  <Property Type="String"
    Name="Location"
    Nullable="true"
    MaxLength="25" />
</EntityType>
```

# Spécification MSL

23/11/2019 • 73 minutes to read

Le langage MSL (Mapping Specification Language) est un langage basé sur XML qui décrit le mappage entre le modèle conceptuel et le modèle de stockage d'une application Entity Framework.

Dans une application Entity Framework, les métadonnées de mappage sont chargées à partir d'un fichier. MSL (écrit en MSL) au moment de la génération. Entity Framework utilise les métadonnées de mappage au moment de l'exécution pour traduire les requêtes sur le modèle conceptuel en commandes spécifiques au stockage.

Le Entity Framework Designer (concepteur EF) stocke les informations de mappage dans un fichier. edmx au moment de la conception. Au moment de la génération, le Entity Designer utilise les informations d'un fichier. edmx pour créer le fichier. MSL requis par Entity Framework au moment de l'exécution.

Les noms de tous les types de modèle conceptuel et de stockage référencés en MSL doivent être qualifiés par le nom de leur espace de noms respectif. Pour plus d'informations sur le nom de l'espace de noms du modèle conceptuel, consultez [spécification CSDL](#). Pour plus d'informations sur le nom de l'espace de noms du modèle de stockage, consultez la [spécification SSDL](#).

Les versions de MSL sont différencierées par les espaces de noms XML.

| VERSION MSL | ESPACE DE NOMS XML  |
|-------------|---|
| MSL v1      | urn : schemas-microsoft-com : Windows : Storage : Mapping : CS  |
| MSL v2      | <a href="https://schemas.microsoft.com/ado/2008/09/mapping/cs">https://schemas.microsoft.com/ado/2008/09/mapping/cs</a> |
| MSL v3      | <a href="https://schemas.microsoft.com/ado/2009/11/mapping/cs">https://schemas.microsoft.com/ado/2009/11/mapping/cs</a> |

## Élément Alias (MSL)

L'élément d' **alias** en Mapping Specification Language (MSL) est un enfant de l'élément de mappage utilisé pour définir des alias pour les espaces de noms du modèle conceptuel et du modèle de stockage. Les noms de tous les types de modèle conceptuel et de stockage référencés en MSL doivent être qualifiés par le nom de leur espace de noms respectif. Pour plus d'informations sur le nom de l'espace de noms du modèle conceptuel, consultez Schema, élément (CSDL). Pour plus d'informations sur le nom de l'espace de noms du modèle de stockage, consultez Schema, élément (SSDL).

L'élément **alias** ne peut pas avoir d'éléments enfants.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **alias** .

| NOM D'ATTRIBUT | REQUIS | VALEUR   |
|----------------|--------|--|
| <b>Key</b>     | Oui    | Alias de l'espace de noms spécifié par l'attribut <b>value</b> .           |
| <b>Valeur</b>  | Oui    | Espace de noms pour lequel la valeur de l'élément <b>Key</b> est un alias. |

## Exemple

L'exemple suivant montre un élément d' **alias** qui définit un alias, `c`, pour les types définis dans le modèle conceptuel.

```
<Mapping Space="C-S"
    xmlns="https://schemas.microsoft.com/ado/2009/11/mapping/cs">
    <Alias Key="c" Value="SchoolModel"/>
    <EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
        CdmEntityContainer="SchoolModelEntities">
        <EntityTypeMapping TypeName="c.Course">
            <MappingFragment StoreEntitySet="Course">
                <ScalarProperty Name="CourseID" ColumnName="CourseID" />
                <ScalarProperty Name="Title" ColumnName="Title" />
                <ScalarProperty Name="Credits" ColumnName="Credits" />
                <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
            </MappingFragment>
        </EntityTypeMapping>
    </EntityContainerMapping>
    <EntitySetMapping Name="Courses">
        <EntityTypeMapping TypeName="c.Course">
            <MappingFragment StoreEntitySet="Course">
                <ScalarProperty Name="CourseID" ColumnName="CourseID" />
                <ScalarProperty Name="Title" ColumnName="Title" />
                <ScalarProperty Name="Credits" ColumnName="Credits" />
                <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
            </MappingFragment>
        </EntityTypeMapping>
    </EntitySetMapping>
    <EntitySetMapping Name="Departments">
        <EntityTypeMapping TypeName="c.Department">
            <MappingFragment StoreEntitySet="Department">
                <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
                <ScalarProperty Name="Name" ColumnName="Name" />
                <ScalarProperty Name="Budget" ColumnName="Budget" />
                <ScalarProperty Name="StartDate" ColumnName="StartDate" />
                <ScalarProperty Name="Administrator" ColumnName="Administrator" />
            </MappingFragment>
        </EntityTypeMapping>
    </EntitySetMapping>
    </EntityContainerMapping>
</Mapping>
```

## AssociationEnd, élément (MSL)

L'élément **AssociationEnd** en Mapping Specification Language (MSL) est utilisé lorsque les fonctions de modification d'un type d'entité dans le modèle conceptuel sont mappées aux procédures stockées dans la base de données sous-jacente. Si une procédure stockée de modification accepte un paramètre dont la valeur est conservée dans une propriété Association, l'élément **AssociationEnd** mappe la valeur de la propriété au paramètre. Pour plus d'informations, voir l'exemple ci-dessous.

Pour plus d'informations sur le mappage des fonctions de modification de types d'entité aux procédures stockées, consultez l'élément **ModificationFunctionMapping** (MSL) et procédure pas à pas : mappage d'une entité à des procédures stockées.

L'élément **AssociationEnd** peut avoir les éléments enfants suivants :

- **ScalarProperty**

### Attributs applicables

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **AssociationEnd**.

| NOM D'ATTRIBUT        | REQUIS | VALEUR                       |
|-----------------------|--------|------------------------------|
| <b>AssociationSet</b> | Oui    | Nom de l'association mappée. |

| NOM D'ATTRIBUT | REQUIS | VALEUR  |
|----------------|--------|---|
| <b>From</b>    | Oui    | Valeur de l'attribut <b>FromRole</b> de la propriété de navigation qui correspond à l'Association qui est mappée. Pour plus d'informations, consultez NavigationProperty, élément (CSDL). |
| <b>Pour</b>    | Oui    | Valeur de l'attribut <b>ToRole</b> de la propriété de navigation qui correspond à l'Association qui est mappée. Pour plus d'informations, consultez NavigationProperty, élément (CSDL).   |

## Exemple

Considérons le type d'entité de modèle conceptuel suivant :

```
<EntityType Name="Course">
  <Key>
    <PropertyRef Name="CourseID" />
  </Key>
  <Property Type="Int32" Name="CourseID" Nullable="false" />
  <Property Type="String" Name="Title" Nullable="false" MaxLength="100"
            FixedLength="false" Unicode="true" />
  <Property Type="Int32" Name="Credits" Nullable="false" />
  <NavigationProperty Name="Department"
    Relationship="SchoolModel.FK_Course_Department"
    FromRole="Course" ToRole="Department" />
</EntityType>
```

Considérons également la procédure stockée suivante :

```
CREATE PROCEDURE [dbo].[UpdateCourse]
  @CourseID int,
  @Title nvarchar(50),
  @Credits int,
  @DepartmentID int
  AS
    UPDATE Course SET Title=@Title,
                     Credits=@Credits,
                     DepartmentID=@DepartmentID
    WHERE CourseID=@CourseID;
```

Pour mapper la fonction de mise à jour de l'entité `Course` à cette procédure stockée, vous devez fournir une valeur au paramètre `DepartmentID`. La valeur pour `DepartmentID` ne correspond pas à une propriété sur le type d'entité ; elle est contenue dans une association indépendante dont le mappage est indiqué ici :

```
<AssociationSetMapping Name="FK_Course_Department"
  TypeName="SchoolModel.FK_Course_Department"
  StoreEntitySet="Course">
  <EndProperty Name="Course">
    <ScalarProperty Name="CourseID" ColumnName="CourseID" />
  </EndProperty>
  <EndProperty Name="Department">
    <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
  </EndProperty>
</AssociationSetMapping>
```

Le code suivant montre l'élément **AssociationEnd** utilisé pour mapper la propriété `DepartmentID` de

l'association **FK\_course\_Department** à la procédure stockée **UpdateCourse** (à laquelle la fonction de mise à jour du type d'entité **course** est mappée) :

```
<EntitySetMapping Name="Courses">
  <EntityTypeMapping TypeName="SchoolModel.Course">
    <MappingFragment StoreEntitySet="Course">
      <ScalarProperty Name="Credits" ColumnName="Credits" />
      <ScalarProperty Name="Title" ColumnName="Title" />
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="SchoolModel.Course">
    <ModificationFunctionMapping>
      <UpdateFunction FunctionName="SchoolModel.Store.UpdateCourse">
        <AssociationEnd AssociationSet="FK_Course_Department"
          From="Course" To="Department">
          <ScalarProperty Name="DepartmentID"
            ParameterName="DepartmentID"
            Version="Current" />
        </AssociationEnd>
        <ScalarProperty Name="Credits" ParameterName="Credits"
          Version="Current" />
        <ScalarProperty Name="Title" ParameterName="Title"
          Version="Current" />
        <ScalarProperty Name="CourseID" ParameterName="CourseID"
          Version="Current" />
      </UpdateFunction>
    </ModificationFunctionMapping>
  </EntityTypeMapping>
</EntitySetMapping>
```

## AssociationSetMapping, élément (MSL)

L'élément **AssociationSetMapping** en Mapping Specification Language (MSL) définit le mappage entre une association dans le modèle conceptuel et les colonnes de table dans la base de données sous-jacente.

Les associations dans le modèle conceptuel sont des types dont les propriétés représentent des colonnes de clé primaire et de clé étrangère dans la base de données sous-jacente. L'élément **AssociationSetMapping** utilise deux éléments EndProperty pour définir les mappages entre les propriétés de type d'association et les colonnes de la base de données. Vous pouvez placer des conditions sur ces mappages avec l'élément Condition. Mappez les fonctions d'insertion, de mise à jour et de suppression pour les associer aux procédures stockées dans la base de données avec l'élément ModificationFunctionMapping. Définir des mappages en lecture seule entre des associations et des colonnes de table à l'aide d'une chaîne de Entity SQL dans un élément QueryView.

### NOTE

Si une contrainte référentielle est définie pour une association dans le modèle conceptuel, l'Association n'a pas besoin d'être mappée avec un élément **AssociationSetMapping**. Si un élément **AssociationSetMapping** est présent pour une association qui a une contrainte référentielle, les mappages définis dans l'élément **AssociationSetMapping** seront ignorés. Pour plus d'informations, consultez ReferentialConstraint, élément (CSDL).

L'élément **AssociationSetMapping** peut avoir les éléments enfants suivants

- QueryView (zéro ou un)
- EndProperty (zéro ou deux éléments)
- Condition (zéro, un ou plusieurs)
- ModificationFunctionMapping (zéro ou un)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **AssociationSetMapping** .

| NOM D'ATTRIBUT        | REQUIS | VALEUR   |
|-----------------------|--------|--|
| <b>Nom</b>            | Oui    | Nom de l'ensemble d'associations du modèle conceptuel mappé.                         |
| <b>TypeName</b>       | Non    | Nom qualifié par un espace de noms du type d'association du modèle conceptuel mappé. |
| <b>StoreEntitySet</b> | Non    | Nom de la table mappée.  |

### Exemple

L'exemple suivant montre un élément **AssociationSetMapping** dans lequel l'Association de **service FK\_de cours** définie dans le modèle conceptuel est mappée à la table **course** de la base de données. Les mappages entre les propriétés de type d'association et les colonnes de table sont spécifiés dans les éléments **EndProperty** enfants.

```
<AssociationSetMapping Name="FK_Course_Department"
    TypeName="SchoolModel.FK_Course_Department"
    StoreEntitySet="Course">
    <EndProperty Name="Department">
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
</AssociationSetMapping>
```

## ComplexProperty, élément (MSL)

Un élément **ComplexProperty** en Mapping Specification Language (MSL) définit le mappage entre une propriété de type complexe sur un type d'entité de modèle conceptuel et des colonnes de table dans la base de données sous-jacente. Les mappages de colonnes de propriété sont spécifiés dans des éléments **ScalarProperty** enfants.

L'élément de propriété **complexType** peut avoir les éléments enfants suivants :

- **ScalarProperty** (zéro, un ou plusieurs)
- **ComplexProperty** (zéro ou plus)
- **ComplexTypeMapping** (zéro ou plus)
- **Condition** (zéro, un ou plusieurs)

### Attributs applicables

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **ComplexProperty** :

| NOM D'ATTRIBUT  | REQUIS | VALEUR   |
|-----------------|--------|--|
| <b>Nom</b>      | Oui    | Nom de la propriété complexe d'un type d'entité dans le modèle conceptuel mappé. |
| <b>TypeName</b> | Non    | Nom qualifié par un espace de noms du type de propriété de modèle conceptuel.    |

### Exemple

L'exemple suivant est basé sur le modèle School. Le type complexe suivant a été ajouté au modèle conceptuel :

```
<ComplexType Name="FullName">
    <Property Type="String" Name="LastName"
        Nullable="false" MaxLength="50"
        FixedLength="false" Unicode="true" />
    <Property Type="String" Name="FirstName"
        Nullable="false" MaxLength="50"
        FixedLength="false" Unicode="true" />
</ComplexType>
```

Les propriétés **LastName** et **FirstName** du type d'entité **Person** ont été remplacées par une propriété complexe, **Name**:

```
<EntityType Name="Person">
    <Key>
        <PropertyRef Name="PersonID" />
    </Key>
    <Property Name="PersonID" Type="Int32" Nullable="false"
        annotation:StoreGeneratedPattern="Identity" />
    <Property Name="HireDate" Type="DateTime" />
    <Property Name="EnrollmentDate" Type="DateTime" />
    <Property Name="Name" Type="SchoolModel.FullName" Nullable="false" />
</EntityType>
```

Le MSL suivant montre l'élément **ComplexProperty** utilisé pour mapper la propriété **Name** aux colonnes de la base de données sous-jacente :

```
<EntitySetMapping Name="People">
    <EntityTypeMapping TypeName="SchoolModel.Person">
        <MappingFragment StoreEntitySet="Person">
            <ScalarProperty Name="PersonID" ColumnName="PersonID" />
            <ScalarProperty Name="HireDate" ColumnName="HireDate" />
            <ScalarProperty Name="EnrollmentDate" ColumnName="EnrollmentDate" />
            <ComplexProperty Name="Name" TypeName="SchoolModel.FullName">
                <ScalarProperty Name="FirstName" ColumnName="FirstName" />
                <ScalarProperty Name="LastName" ColumnName="LastName" />
            </ComplexProperty>
        </MappingFragment>
    </EntityTypeMapping>
</EntitySetMapping>
```

## ComplexTypeMapping, élément (MSL)

L'élément **ComplexTypeMapping**, en Mapping Specification Language (MSL) est un enfant de l'élément **ResultMapping** et définit le mappage entre une importation de fonction dans le modèle conceptuel et une procédure stockée dans la base de données sous-jacente lorsque les conditions suivantes sont remplies :

- L'importation de fonction retourne un type complexe conceptuel.
- Les noms de colonne retournés par la procédure stockée ne correspondent pas exactement aux noms de propriété du type complexe.

Par défaut, le mappage entre les colonnes retournées par une procédure stockée et un type complexe est basé sur les noms de colonne et de propriété. Si les noms de colonne ne correspondent pas exactement aux noms de propriété, vous devez utiliser l'élément **ComplexTypeMapping**, pour définir le mappage. Pour obtenir un exemple du mappage par défaut, consultez élément **FunctionImportMapping** (MSL).

L'élément **ComplexTypeMapping**, peut avoir les éléments enfants suivants :

- ScalarProperty (zéro, un ou plusieurs)

## Attributs applicables

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **ComplexTypeMapping**.

| NOM D'ATTRIBUT  | REQUIS | VALEUR   |
|-----------------|--------|--|
| <b>TypeName</b> | Oui    | Nom qualifié par un espace de noms du type complexe mappé. |

## Exemple

Considérons la procédure stockée suivante :

```
CREATE PROCEDURE [dbo].[GetGrades]
    @student_Id int
    AS
        SELECT EnrollmentID as enroll_id,
               Grade as grade,
               CourseID as course_id,
               StudentID as student_id
        FROM dbo.StudentGrade
        WHERE StudentID = @student_Id
```

De même, considérons le type complexe de modèle conceptuel suivant :

```
<ComplexType Name="GradeInfo">
    <Property Type="Int32" Name="EnrollmentID" Nullable="false" />
    <Property Type="Decimal" Name="Grade" Nullable="true"
              Precision="3" Scale="2" />
    <Property Type="Int32" Name="CourseID" Nullable="false" />
    <Property Type="Int32" Name="StudentID" Nullable="false" />
</ComplexType>
```

Afin de créer une importation de fonction qui retourne des instances du type complexe précédent, le mappage entre les colonnes rentrées par la procédure stockée et le type d'entité doit être défini dans un élément

## ComplexTypeMapping :

```
<FunctionImportMapping FunctionImportName="GetGrades"
                           FunctionName="SchoolModel.Store.GetGrades" >
    <ResultMapping>
        <ComplexTypeMapping TypeName="SchoolModel.GradeInfo">
            <ScalarProperty Name="EnrollmentID" ColumnName="enroll_id"/>
            <ScalarProperty Name="CourseID" ColumnName="course_id"/>
            <ScalarProperty Name="StudentID" ColumnName="student_id"/>
            <ScalarProperty Name="Grade" ColumnName="grade"/>
        </ComplexTypeMapping>
    </ResultMapping>
</FunctionImportMapping>
```

## Condition, élément (MSL)

L'élément **condition** en Mapping Specification Language (MSL) place des conditions sur les mappages entre le modèle conceptuel et la base de données sous-jacente. Le mappage défini dans un nœud XML est valide si toutes les conditions, comme spécifié dans les éléments de **condition** enfants, sont remplies. À défaut, le mappage n'est pas valide. Par exemple, si un élément **MappingFragment** contient un ou plusieurs éléments enfants **condition**, le mappage défini dans le nœud **MappingFragment** n'est valide que si toutes les conditions des éléments de **condition** enfants sont remplies.

Chaque condition peut s'appliquer à un **nom** (le nom d'une propriété d'entité de modèle conceptuel, spécifié par l'attribut **Name**) ou à un **ColumnName** (le nom d'une colonne dans la base de données, spécifié par l'attribut **ColumnName**). Lorsque l'attribut **Name** est défini, la condition est vérifiée par rapport à une valeur de propriété d'entité. Lorsque l'attribut **ColumnName** est défini, la condition est vérifiée par rapport à une valeur de colonne. Seul l'un des attributs **Name** ou **ColumnName** peut être spécifié dans un élément **condition**.

#### NOTE

Lorsque l'élément **condition** est utilisé dans un élément **FunctionImportMapping**, seul l'attribut **Name** n'est pas applicable.

L'élément **condition** peut être un enfant des éléments suivants :

- **AssociationSetMapping**
- **ComplexProperty**
- **EntityTypeMapping**
- **MappingFragment**
- **EntityTypeMapping**

L'élément **condition** ne peut pas avoir d'éléments enfants.

#### Attributs applicables

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **condition** :

| NOM D'ATTRIBUT    | REQUIS | VALEUR  |
|-------------------|--------|---|
| <b>ColumnName</b> | Non    | Nom de la colonne de table dont la valeur est utilisée pour évaluer la condition.   |
| <b>IsNull</b>     | Non    | <b>True</b> ou <b>false</b> . Si la valeur est <b>true</b> et que la valeur de la colonne est <b>null</b> , ou si la valeur est <b>false</b> et que la valeur de la colonne n'est pas <b>null</b> , la condition est true. Sinon, la condition n'est pas vérifiée (False).<br>Les attributs <b>IsNull</b> et <b>value</b> ne peuvent pas être utilisés en même temps. |
| <b>Valeur</b>     | Non    | Valeur à laquelle la valeur de colonne est comparée. Si les valeurs sont identiques, la condition est vérifiée (True). Sinon, la condition n'est pas vérifiée (False).<br>Les attributs <b>IsNull</b> et <b>value</b> ne peuvent pas être utilisés en même temps.   |
| <b>Nom</b>        | Non    | Nom de la propriété d'entité de modèle conceptuel dont la valeur est utilisée pour évaluer la condition.<br>Cet attribut n'est pas applicable si l'élément <b>condition</b> est utilisé dans un élément <b>FunctionImportMapping</b> .  |

#### Exemple

L'exemple suivant montre des éléments de **condition** en tant qu'enfants d'éléments **MappingFragment**.

Lorsque **HireDate** n'a pas la valeur null et que **EnrollmentDate** a la valeur null, les données sont mappées entre

le type **SchoolModel. Instructor** et les colonnes **PersonID** et **HireDate** de la table **Person** . Lorsque **EnrollmentDate** n'a pas la valeur null et **HireDate** a la valeur null, les données sont mappées entre le type **SchoolModel. Student** et les colonnes **PersonID** et **inscription** de la table **Person** .

```
<EntitySetMapping Name="People">
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Person)">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="LastName" ColumnName="LastName" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Instructor)">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="HireDate" ColumnName="HireDate" />
      <Condition ColumnName="HireDate" IsNull="false" />
      <Condition ColumnName="EnrollmentDate" IsNull="true" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Student)">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="EnrollmentDate"
        ColumnName="EnrollmentDate" />
      <Condition ColumnName="EnrollmentDate" IsNull="false" />
      <Condition ColumnName="HireDate" IsNull="true" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

## DeleteFunction, élément (MSL)

L'élément **DeleteFunction** en Mapping Specification Language (MSL) mappe la fonction Delete d'un type d'entité ou d'une association dans le modèle conceptuel à une procédure stockée dans la base de données sous-jacente. Les procédures stockées auxquelles des fonctions de modification sont mappées doivent être déclarées dans le modèle de stockage. Pour plus d'informations, consultez Function, élément (SSDL).

### NOTE

Si vous ne mappez pas les trois opérations d'insertion, de mise à jour ou de suppression d'un type d'entité à des procédures stockées, les opérations non mappées échouent si elles sont exécutées au moment de l'exécution et qu'une UpdateException est levée.

### Application de DeleteFunction à EntityTypeMapping

Lorsqu'il est appliqué à l'élément EntityTypeMapping, l'élément **DeleteFunction** mappe la fonction Delete d'un type d'entité dans le modèle conceptuel à une procédure stockée.

L'élément **DeleteFunction** peut avoir les éléments enfants suivants lorsqu'il est appliqué à un élément **EntityTypeMapping** :

- AssociationEnd (zéro, un ou plusieurs)
- ComplexProperty (zéro, un ou plusieurs éléments) ;
- ScalarProperty (zéro ou plus)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **DeleteFunction** lorsqu'il est appliqué à un élément **EntityTypeMapping** .

| NOM D'ATTRIBUT               | REQUIS | VALEUR   |
|------------------------------|--------|--|
| <b>FunctionName</b>          | Oui    | Nom qualifié par un espace de noms de la procédure stockée à laquelle la fonction de suppression est mappée. La procédure stockée doit être déclarée dans le modèle de stockage. |
| <b>RowsAffectedParameter</b> | Non    | Nom du paramètre de sortie qui retourne le nombre de lignes affectées.   |

#### Exemple

L'exemple suivant est basé sur le modèle School et montre l'élément **DeleteFunction** qui mappe la fonction Delete du type d'entité **Person** à la procédure stockée **DeletePerson**. La procédure stockée **DeletePerson** est déclarée dans le modèle de stockage.

```

<EntitySetMapping Name="People">
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="LastName" ColumnName="LastName" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="HireDate" ColumnName="HireDate" />
      <ScalarProperty Name="EnrollmentDate"
                     ColumnName="EnrollmentDate" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <ModificationFunctionMapping>
      <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
        <ScalarProperty Name="EnrollmentDate"
                       ParameterName="EnrollmentDate" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName" />
        <ScalarProperty Name="LastName" ParameterName="LastName" />
        <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
      </InsertFunction>
      <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
        <ScalarProperty Name="EnrollmentDate"
                       ParameterName="EnrollmentDate"
                       Version="Current" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate"
                       Version="Current" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName"
                       Version="Current" />
        <ScalarProperty Name="LastName" ParameterName="LastName"
                       Version="Current" />
        <ScalarProperty Name="PersonID" ParameterName="PersonID"
                       Version="Current" />
      </UpdateFunction>
      <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
        <ScalarProperty Name="PersonID" ParameterName="PersonID" />
      </DeleteFunction>
    </ModificationFunctionMapping>
  </EntityTypeMapping>
</EntitySetMapping>

```

#### Application de **DeleteFunction** à **AssociationSetMapping**

Lorsqu'il est appliqué à l'élément **AssociationSetMapping**, l'élément **DeleteFunction** mappe la fonction Delete d'une association dans le modèle conceptuel à une procédure stockée.

L'élément **DeleteFunction** peut avoir les éléments enfants suivants lorsqu'il est appliqué à l'élément

## **AssociationSetMapping :**

- EndProperty

### **Attributs applicables**

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **DeleteFunction** lorsqu'il est appliqué à l'élément **AssociationSetMapping**.

| NOM D'ATTRIBUT               | REQUIS | VALEUR   |
|------------------------------|--------|--|
| <b>FunctionName</b>          | Oui    | Nom qualifié par un espace de noms de la procédure stockée à laquelle la fonction de suppression est mappée. La procédure stockée doit être déclarée dans le modèle de stockage. |
| <b>RowsAffectedParameter</b> | Non    | Nom du paramètre de sortie qui retourne le nombre de lignes affectées.   |

### **Exemple**

L'exemple suivant est basé sur le modèle School et montre l'élément **DeleteFunction** utilisé pour mapper la fonction Delete de l'Association **CourseInstructor** à la procédure stockée **DeleteCourseInstructor**. La procédure stockée **DeleteCourseInstructor** est déclarée dans le modèle de stockage.

```
<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>
```

## **EndProperty, élément (MSL)**

L'élément **EndProperty** en Mapping Specification Language (MSL) définit le mappage entre une terminaison ou une fonction de modification d'une association de modèle conceptuel et la base de données sous-jacente. Le mappage de colonne de propriété est spécifié dans un élément **ScalarProperty** enfant.

Lorsqu'un élément **EndProperty** est utilisé pour définir le mappage de la fin d'une association de modèle conceptuel, il est un enfant d'un élément **AssociationSetMapping**. Lorsque l'élément **EndProperty** est utilisé pour

définir le mappage d'une fonction de modification d'une association de modèle conceptuel, il est un enfant d'un élément InsertFunction ou d'un élément DeleteFunction.

L'élément **EndProperty** peut avoir les éléments enfants suivants :

- ScalarProperty (zéro, un ou plusieurs)

### Attributs applicables

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **EndProperty** :

| NOM D'ATTRIBUT | REQUIS | VALEUR                                      |
|----------------|--------|---|
| Nom            | Oui    | Nom de la terminaison d'association mappée. |

### Exemple

L'exemple suivant montre un élément **AssociationSetMapping** dans lequel l'association **FK\_cours\_service** dans le modèle conceptuel est mappée à la table **course** de la base de données. Les mappages entre les propriétés de type d'association et les colonnes de table sont spécifiés dans les éléments **EndProperty** enfants.

```
<AssociationSetMapping Name="FK_Course_Department"
    TypeName="SchoolModel.FK_Course_Department"
    StoreEntitySet="Course">
    <EndProperty Name="Department">
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
</AssociationSetMapping>
```

### Exemple

L'exemple suivant montre l'élément **EndProperty** qui mappe les fonctions d'insertion et de suppression d'une association (**CourseInstructor**) à des procédures stockées dans la base de données sous-jacente. Les fonctions mappées sont déclarées dans le modèle de stockage.

```

<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>

```

## EntityContainerMapping, élément (MSL)

L’élément **EntityContainerMapping** en Mapping Specification Language (MSL) mappe le conteneur d’entités du modèle conceptuel au conteneur d’entités dans le modèle de stockage. L’élément **EntityContainerMapping** est un enfant de l’élément Mapping.

L’élément **EntityContainerMapping** peut avoir les éléments enfants suivants (dans l’ordre indiqué) :

- EntitySetMapping (zéro, un ou plusieurs éléments) ;
- AssociationSetMapping (zéro, un ou plusieurs)
- FunctionImportMapping (zéro ou plus)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l’élément **EntityContainerMapping**.

| NOM D'ATTRIBUT               | REQUIS | VALEUR   |
|------------------------------|--------|--|
| <b>StorageModelContainer</b> | Oui    | Nom du conteneur d’entités de modèle de stockage mappé.  |
| <b>CdmEntityContainer</b>    | Oui    | Nom du conteneur d’entités de modèle conceptuel mappé.   |
| <b>GenerateUpdateViews</b>   | Non    | <b>True</b> ou <b>false</b> . Si la <b>valeur est false</b> , aucune vue de mise à jour n'est générée. Cet attribut doit avoir la valeur <b>false</b> lorsque vous avez un mappage en lecture seule qui serait non valide, car les données ne peuvent pas aller-retour. La valeur par défaut est <b>True</b> . |

## Exemple

L'exemple suivant montre un élément **EntityContainerMapping** qui mappe le conteneur **SchoolModelEntities** (le conteneur d'entités de modèle conceptuel) au conteneur **SchoolModelStoreContainer** (conteneur d'entités de modèle de stockage) :

```
<EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
                      CdmEntityContainer="SchoolModelEntities">
  <EntitySetMapping Name="Courses">
    <EntityTypeMapping TypeName="c.Course">
      <MappingFragment StoreEntitySet="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
        <ScalarProperty Name="Title" ColumnName="Title" />
        <ScalarProperty Name="Credits" ColumnName="Credits" />
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
  <EntitySetMapping Name="Departments">
    <EntityTypeMapping TypeName="c.Department">
      <MappingFragment StoreEntitySet="Department">
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
        <ScalarProperty Name="Name" ColumnName="Name" />
        <ScalarProperty Name="Budget" ColumnName="Budget" />
        <ScalarProperty Name="StartDate" ColumnName="StartDate" />
        <ScalarProperty Name="Administrator" ColumnName="Administrator" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
</EntityContainerMapping>
```

## EntitySetMapping, élément (MSL)

L'élément **EntitySetMapping** en Mapping Specification Language (MSL) mappe tous les types dans un jeu d'entités de modèle conceptuel aux jeux d'entités dans le modèle de stockage. Un jeu d'entités dans le modèle conceptuel est un conteneur logique pour instances d'entités de même type (et des types dérivés). Un jeu d'entités dans le modèle de stockage représente une table ou une vue de la base de données sous-jacente. Le jeu d'entités du modèle conceptuel est spécifié par la valeur de l'attribut **Name** de l'élément **EntitySetMapping**. La table ou la vue mappée à est spécifiée par l'attribut **StoreEntitySet** dans chaque élément MappingFragment enfant ou dans l'élément **EntitySetMapping** lui-même.

L'élément **EntitySetMapping** peut avoir les éléments enfants suivants :

- EntityTypeMapping (zéro, un ou plusieurs éléments) ;
- QueryView (zéro ou un)
- MappingFragment (zéro, un ou plusieurs)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **EntitySetMapping**.

| NOM D'ATTRIBUT    | REQUIS | VALEUR   |
|-------------------|--------|--|
| <b>Nom</b>        | Oui    | Nom du jeu d'entités de modèle conceptuel mappé. |
| <b>TypeName 1</b> | Non    | Nom du type d'entité de modèle conceptuel mappé. |

| NOM D'ATTRIBUT             | REQUIS | VALEUR  |
|----------------------------|--------|---|
| <b>StoreEntitySet</b> 1    | Non    | Nom du jeu d'entités de modèle de stockage de destination du mappage.   |
| <b>MakeColumnsDistinct</b> | Non    | <b>True</b> ou <b>false</b> selon que seules des lignes distinctes sont retournées.<br>Si cet attribut est défini sur <b>true</b> , l'attribut <b>GenerateUpdateViews</b> de l'élément EntityContainerMapping doit avoir la valeur <b>false</b> . |

1 les attributs **TypeName** et **StoreEntitySet** peuvent être utilisés à la place des éléments enfants EntityTypeMapping et MappingFragment pour mapper un type d'entité unique à une table unique.

### Exemple

L'exemple suivant montre un élément **EntityTypeMapping** qui mappe trois types (un type de base et deux types dérivés) dans le jeu d'entités **courses** du modèle conceptuel à trois tables différentes dans la base de données sous-jacente. Les tables sont spécifiées par l'attribut **StoreEntitySet** dans chaque élément **MappingFragment**.

```
<EntityTypeMapping TypeName="IsTypeOf(SchoolModel1.Course)">
    <MappingFragment StoreEntitySet="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
        <ScalarProperty Name="Credits" ColumnName="Credits" />
        <ScalarProperty Name="Title" ColumnName="Title" />
    </MappingFragment>
</EntityTypeMapping>
<EntityTypeMapping TypeName="IsTypeOf(SchoolModel1.OnlineCourse)">
    <MappingFragment StoreEntitySet="OnlineCourse">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
        <ScalarProperty Name="URL" ColumnName="URL" />
    </MappingFragment>
</EntityTypeMapping>
<EntityTypeMapping TypeName="IsTypeOf(SchoolModel1.OnsiteCourse)">
    <MappingFragment StoreEntitySet="OnsiteCourse">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
        <ScalarProperty Name="Time" ColumnName="Time" />
        <ScalarProperty Name="Days" ColumnName="Days" />
        <ScalarProperty Name="Location" ColumnName="Location" />
    </MappingFragment>
</EntityTypeMapping>
</EntityTypeMapping>
```

## EntityTypeMapping, élément (MSL)

L'élément **EntityTypeMapping** en Mapping Specification Language (MSL) définit le mappage entre un type d'entité dans le modèle conceptuel et les tables ou vues dans la base de données sous-jacente. Pour plus d'informations sur les types d'entité de modèle conceptuel et les tables ou les vues de base de données sous-jacente, consultez Élément EntityType (CSDL) et Élément EntitySet (SSDL). Le type d'entité de modèle conceptuel qui est mappé est spécifié par l'attribut **TypeName** de l'élément **EntityTypeMapping**. La table ou la vue mappée est spécifiée par l'attribut **StoreEntitySet** de l'élément **MappingFragment** enfant.

L'élément enfant **ModificationFunctionMapping** peut être utilisé pour mapper les fonctions d'insertion, de mise à jour ou de suppression de types d'entités aux procédures stockées de la base de données.

L'élément **EntityTypeMapping** peut avoir les éléments enfants suivants :

- MappingFragment (zéro, un ou plusieurs)
- ModificationFunctionMapping (zéro ou un)
- ScalarProperty
- Condition

#### NOTE

Les éléments **MappingFragment** et **ModificationFunctionMapping** ne peuvent pas être des éléments enfants de l'élément **EntityTypeMapping** en même temps.

#### NOTE

Les éléments **ScalarProperty** et **condition** ne peuvent être que des éléments enfants de l'élément **EntityTypeMapping** lorsqu'il est utilisé dans un élément FunctionImportMapping.

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **EntityTypeMapping**.

| NOM D'ATTRIBUT  | REQUIS | VALEUR   |
|-----------------|--------|--|
| <b>TypeName</b> | Oui    | Nom qualifié par un espace de noms du type d'entité de modèle conceptuel mappé.<br>Si le type correspond à un type abstrait ou dérivé, la valeur doit être<br><code>IsOfType(Namespace-qualified_type_name)</code> |

### Exemple

L'exemple suivant montre un élément EntitySetMapping avec deux éléments **EntityTypeMapping** enfants. Dans le premier élément **EntityTypeMapping**, le type d'entité **SchoolModel. Person** est mappé à la table **Person**. Dans le deuxième élément **EntityTypeMapping**, la fonctionnalité de mise à jour du type **SchoolModel. Person** est mappée à une procédure stockée, **UpdatePerson**, dans la base de données.

```

<EntitySetMapping Name="People">
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="LastName" ColumnName="LastName" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="HireDate" ColumnName="HireDate" />
      <ScalarProperty Name="EnrollmentDate" ColumnName="EnrollmentDate" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <ModificationFunctionMapping>
      <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
        <ScalarProperty Name="EnrollmentDate" ParameterName="EnrollmentDate"
          Version="Current" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate"
          Version="Current" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName"
          Version="Current" />
        <ScalarProperty Name="LastName" ParameterName="LastName"
          Version="Current" />
        <ScalarProperty Name="PersonID" ParameterName="PersonID"
          Version="Current" />
      </UpdateFunction>
    </ModificationFunctionMapping>
  </EntityTypeMapping>
</EntitySetMapping>

```

## Exemple

L'exemple suivant illustre le mappage d'une hiérarchie de types dont le type racine est abstrait. Notez l'utilisation de la syntaxe `IsOfType` pour les attributs **TypeName**.

```

<EntitySetMapping Name="People">
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Person)">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="LastName" ColumnName="LastName" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Instructor)">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="HireDate" ColumnName="HireDate" />
      <Condition ColumnName="HireDate" IsNull="false" />
      <Condition ColumnName="EnrollmentDate" IsNull="true" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.Student)">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="EnrollmentDate"
        ColumnName="EnrollmentDate" />
      <Condition ColumnName="EnrollmentDate" IsNull="false" />
      <Condition ColumnName="HireDate" IsNull="true" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>

```

## FunctionImportMapping, élément (MSL)

L'élément **FunctionImportMapping** en Mapping Specification Language (MSL) définit le mappage entre une

importation de fonction dans le modèle conceptuel et une procédure stockée ou une fonction dans la base de données sous-jacente. Les importations de fonction doivent être déclarées dans le modèle conceptuel et les procédures stockées dans le modèle de stockage. Pour plus d'informations, consultez **FunctionImport**, élément (CSDL) et **Function**, élément (SSDL).

#### NOTE

Par défaut, si une importation de fonction retourne un type d'entité ou un type complexe de modèle conceptuel, les noms des colonnes renvoyés par la procédure stockée sous-jacente doivent correspondre exactement aux noms des propriétés sur le type de modèle conceptuel. Si les noms de colonne ne correspondent pas exactement aux noms de propriété, le mappage doit être défini dans un élément **ResultMapping**.

L'élément **FunctionImportMapping** peut avoir les éléments enfants suivants :

- **ResultMapping** (zéro ou plus)

#### Attributs applicables

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **FunctionImportMapping** :

| NOM D'ATTRIBUT            | REQUIS | VALEUR  |
|---------------------------|--------|---|
| <b>FunctionImportName</b> | Oui    | Nom de l'importation de fonction dans le modèle conceptuel mappé.                   |
| <b>FunctionName</b>       | Oui    | Nom qualifié par un espace de noms de la fonction dans le modèle de stockage mappé. |

#### Exemple

L'exemple suivant est basé sur le modèle School. Considérez la fonction suivante dans le modèle de stockage :

```
<Function Name="GetStudentGrades" Aggregate="false"
          BuiltIn="false" NiladicFunction="false"
          IsComposable="false" ParameterTypeSemantics="AllowImplicitConversion"
          Schema="dbo">
    <Parameter Name="StudentID" Type="int" Mode="In" />
</Function>
```

Considérez également cette importation de fonction dans le modèle conceptuel :

```
<FunctionImport Name="GetStudentGrades" EntitySet="StudentGrades"
                ReturnType="Collection(SchoolModel.StudentGrade)">
    <Parameter Name="StudentID" Mode="In" Type="Int32" />
</FunctionImport>
```

L'exemple suivant montre un élément **FunctionImportMapping** utilisé pour mapper la fonction et l'importation de fonction ci-dessus :

```
<FunctionImportMapping FunctionImportName="GetStudentGrades"
                           FunctionName="SchoolModel.Store.GetStudentGrades" />
```

## InsertFunction, élément (MSL)

L'élément **InsertFunction** en Mapping Specification Language (MSL) mappe la fonction d'insertion d'un type d'entité ou d'une association dans le modèle conceptuel à une procédure stockée dans la base de données sous-jacente. Les procédures stockées auxquelles des fonctions de modification sont mappées doivent être déclarées dans le modèle de stockage. Pour plus d'informations, consultez Function, élément (SSDL).

#### NOTE

Si vous ne mappez pas les trois opérations d'insertion, de mise à jour ou de suppression d'un type d'entité à des procédures stockées, les opérations non mappées échouent si elles sont exécutées au moment de l'exécution et qu'une UpdateException est levée.

L'élément **InsertFunction** peut être un enfant de l'élément **ModificationFunctionMapping** et être appliqué à l'élément **EntityTypeMapping** ou à l'élément **AssociationSetMapping**.

#### Application d'**InsertFunction** à **EntityTypeMapping**

Lorsqu'il est appliqué à l'élément **EntityTypeMapping**, l'élément **InsertFunction** mappe la fonction d'insertion d'un type d'entité dans le modèle conceptuel à une procédure stockée.

L'élément **InsertFunction** peut avoir les éléments enfants suivants lorsqu'il est appliqué à un élément **EntityTypeMapping** :

- AssociationEnd (zéro, un ou plusieurs)
- ComplexProperty (zéro, un ou plusieurs éléments) ;
- ResultBinding (zéro ou un)
- ScalarProperty (zéro ou plus)

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **InsertFunction** lorsqu'ils sont appliqués à un élément **EntityTypeMapping** .

| NOM D'ATTRIBUT               | REQUIS | VALEUR  |
|------------------------------|--------|---|
| <b>FunctionName</b>          | Oui    | Nom qualifié par un espace de noms de la procédure stockée à laquelle la fonction d'insertion est mappée. La procédure stockée doit être déclarée dans le modèle de stockage. |
| <b>RowsAffectedParameter</b> | Non    | Nom du paramètre de sortie qui retourne le nombre de lignes affectées.  |

#### Exemple

L'exemple suivant est basé sur le modèle School et montre l'élément **InsertFunction** utilisé pour mapper la fonction d'insertion du type d'entité Person à la procédure stockée **InsertPerson** . La procédure stockée **InsertPerson** est déclarée dans le modèle de stockage.

```

<EntityTypeMapping TypeName="SchoolModel.Person">
  <ModificationFunctionMapping>
    <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName" />
      <ScalarProperty Name="LastName" ParameterName="LastName" />
      <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
    </InsertFunction>
    <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate"
                      Version="Current" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate"
                      Version="Current" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName"
                      Version="Current" />
      <ScalarProperty Name="LastName" ParameterName="LastName"
                      Version="Current" />
      <ScalarProperty Name="PersonID" ParameterName="PersonID"
                      Version="Current" />
    </UpdateFunction>
    <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
      <ScalarProperty Name="PersonID" ParameterName="PersonID" />
    </DeleteFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>

```

## Application d'InsertFunction à AssociationSetMapping

Lorsqu'il est appliqué à l'élément AssociationSetMapping, l'élément **InsertFunction** mappe la fonction d'insertion d'une association dans le modèle conceptuel à une procédure stockée.

L'élément **InsertFunction** peut avoir les éléments enfants suivants lorsqu'il est appliqué à l'élément **AssociationSetMapping** :

- EndProperty

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **InsertFunction** lorsqu'il est appliqué à l'élément **AssociationSetMapping** .

| NOM D'ATTRIBUT               | REQUIS | VALEUR  |
|------------------------------|--------|---|
| <b>FunctionName</b>          | Oui    | Nom qualifié par un espace de noms de la procédure stockée à laquelle la fonction d'insertion est mappée. La procédure stockée doit être déclarée dans le modèle de stockage. |
| <b>RowsAffectedParameter</b> | Non    | Nom du paramètre de sortie qui retourne le nombre de lignes affectées.  |

### Exemple

L'exemple suivant est basé sur le modèle School et montre l'élément **InsertFunction** utilisé pour mapper la fonction d'insertion de l'Association **CourseInstructor** à la procédure stockée **InsertCourseInstructor** . La procédure stockée **InsertCourseInstructor** est déclarée dans le modèle de stockage.

```

<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>

```

## Élément de mappage (MSL)

L'élément **Mapping** en Mapping Specification Language (MSL) contient des informations pour le mappage d'objets définis dans un modèle conceptuel à une base de données (comme décrit dans un modèle de stockage). Pour plus d'informations, consultez Spécification CSDL et spécification SSDL.

L'élément **Mapping** est l'élément racine d'une spécification de mappage. L'espace de noms XML pour les spécifications de mappage est <https://schemas.microsoft.com/ado/2009/11/mapping/cs>.

L'élément de mappage peut avoir les éléments enfants suivants (dans l'ordre répertorié) :

- Alias (zéro, un ou plusieurs)
- EntityContainerMapping (exactement un)

Les noms de types de modèle conceptuel et de stockage référencés en MSL doivent être qualifiés par le nom de leur espace de noms respectif. Pour plus d'informations sur le nom de l'espace de noms du modèle conceptuel, consultez Schema, élément (CSDL). Pour plus d'informations sur le nom de l'espace de noms du modèle de stockage, consultez Schema, élément (SSDL). Les alias d'espace de noms utilisés en MSL peuvent être définis avec l'élément Alias.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **Mapping** .

| NOM D'ATTRIBUT | REQUIS | VALEUR   |
|----------------|--------|--|
| <b>Espace</b>  | Oui    | <b>C-S.</b> Il s'agit d'une valeur fixe qui ne peut pas être modifiée. |

### Exemple

L'exemple suivant illustre un élément de **mappage** basé sur une partie du modèle School. Pour plus

d'informations sur le modèle School, consultez démarrage rapide (Entity Framework) :

```
<Mapping Space="C-S"
      xmlns="https://schemas.microsoft.com/ado/2009/11/mapping/cs">
  <Alias Key="c" Value="SchoolModel"/>
  <EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
    CdmEntityContainer="SchoolModelEntities">
    <EntityTypeMapping TypeName="c.Course">
      <MappingFragment StoreEntitySet="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
        <ScalarProperty Name="Title" ColumnName="Title" />
        <ScalarProperty Name="Credits" ColumnName="Credits" />
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
  <EntitySetMapping Name="Departments">
    <EntityTypeMapping TypeName="c.Department">
      <MappingFragment StoreEntitySet="Department">
        <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
        <ScalarProperty Name="Name" ColumnName="Name" />
        <ScalarProperty Name="Budget" ColumnName="Budget" />
        <ScalarProperty Name="StartDate" ColumnName="StartDate" />
        <ScalarProperty Name="Administrator" ColumnName="Administrator" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntitySetMapping>
</EntityContainerMapping>
</Mapping>
```

## MappingFragment, élément (MSL)

L'élément **MappingFragment** en Mapping Specification Language (MSL) définit le mappage entre les propriétés d'un type d'entité de modèle conceptuel et une table ou une vue dans la base de données. Pour plus d'informations sur les types d'entité de modèle conceptuel et les tables ou les vues de base de données sous-jacente, consultez Élément EntityType (CSDL) et Élément EntitySet (SSDL). L'élément **MappingFragment** peut être un élément enfant de l'élément EntityTypeMapping ou de l'élément EntitySetMapping.

L'élément **MappingFragment** peut avoir les éléments enfants suivants :

- ComplexType (zéro, un ou plusieurs)
- ScalarProperty (zéro, un ou plusieurs)
- Condition (zéro, un ou plusieurs)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **MappingFragment**.

| NOM D'ATTRIBUT             | REQUIS | VALEUR  |
|----------------------------|--------|---|
| <b>StoreEntitySet</b>      | Oui    | Nom de la table ou de la vue mappée.  |
| <b>MakeColumnsDistinct</b> | Non    | <b>True</b> ou <b>false</b> selon que seules des lignes distinctes sont retournées.<br>Si cet attribut est défini sur <b>true</b> , l'attribut <b>GenerateUpdateViews</b> de l'élément EntityContainerMapping doit avoir la valeur <b>false</b> . |

### Exemple

L'exemple suivant montre un élément **MappingFragment** en tant qu'enfant d'un élément **EntityTypeMapping**.

Dans cet exemple, les propriétés du type de **cours** dans le modèle conceptuel sont mappées aux colonnes de la table **course** dans la base de données.

```
<EntitySetMapping Name="Courses">
  <EntityTypeMapping TypeName="SchoolModel.Course">
    <MappingFragment StoreEntitySet="Course">
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
      <ScalarProperty Name="Title" ColumnName="Title" />
      <ScalarProperty Name="Credits" ColumnName="Credits" />
      <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

## Exemple

L'exemple suivant montre un élément **MappingFragment** en tant qu'enfant d'un élément **EntitySetMapping**.

Comme dans l'exemple ci-dessus, les propriétés du type de **cours** dans le modèle conceptuel sont mappées aux colonnes de la table **course** dans la base de données.

```
<EntitySetMapping Name="Courses" TypeName="SchoolModel.Course">
  <MappingFragment StoreEntitySet="Course">
    <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    <ScalarProperty Name="Title" ColumnName="Title" />
    <ScalarProperty Name="Credits" ColumnName="Credits" />
    <ScalarProperty Name="DepartmentID" ColumnName="DepartmentID" />
  </MappingFragment>
</EntitySetMapping>
```

## ModificationFunctionMapping, élément (MSL)

L'élément **ModificationFunctionMapping** en Mapping Specification Language (MSL) mappe les fonctions d'insertion, de mise à jour et de suppression d'un type d'entité de modèle conceptuel aux procédures stockées dans la base de données sous-jacente. L'élément **ModificationFunctionMapping** peut également mapper les fonctions d'insertion et de suppression pour les associations plusieurs-à-plusieurs dans le modèle conceptuel aux procédures stockées dans la base de données sous-jacente. Les procédures stockées auxquelles des fonctions de modification sont mappées doivent être déclarées dans le modèle de stockage. Pour plus d'informations, consultez Function, élément (SSDL).

### NOTE

Si vous ne mappez pas les trois opérations d'insertion, de mise à jour ou de suppression d'un type d'entité à des procédures stockées, les opérations non mappées échouent si elles sont exécutées au moment de l'exécution et qu'une UpdateException est levée.

### NOTE

Si les fonctions de modification pour une entité dans une hiérarchie d'héritage sont mappées aux procédures stockées, les fonctions de modification de tous les types dans la hiérarchie doivent être mappées aux procédures stockées.

L'élément **ModificationFunctionMapping** peut être un enfant de l'élément **EntityTypeMapping** ou de l'élément **AssociationSetMapping**.

L'élément **ModificationFunctionMapping** peut avoir les éléments enfants suivants :

- DeleteFunction (zéro ou un)
- InsertFunction (zéro ou un)
- UpdateFunction (zéro ou un)

Aucun attribut n'est applicable à l'élément **ModificationFunctionMapping**.

### Exemple

L'exemple suivant montre le mappage de jeu d'entités pour le jeu d'entités **People** dans le modèle School. En plus du mappage de colonnes pour le type d'entité **Person**, le mappage des fonctions d'insertion, de mise à jour et de suppression du type **Person** est affiché. Les fonctions mappées sont déclarées dans le modèle de stockage.

```
<EntitySetMapping Name="People">
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="LastName" ColumnName="LastName" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="HireDate" ColumnName="HireDate" />
      <ScalarProperty Name="EnrollmentDate"
        ColumnName="EnrollmentDate" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <ModificationFunctionMapping>
      <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
        <ScalarProperty Name="EnrollmentDate"
          ParameterName="EnrollmentDate" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName" />
        <ScalarProperty Name="LastName" ParameterName="LastName" />
        <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
      </InsertFunction>
      <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
        <ScalarProperty Name="EnrollmentDate"
          ParameterName="EnrollmentDate"
          Version="Current" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate"
          Version="Current" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName"
          Version="Current" />
        <ScalarProperty Name="LastName" ParameterName="LastName"
          Version="Current" />
        <ScalarProperty Name="PersonID" ParameterName="PersonID"
          Version="Current" />
      </UpdateFunction>
      <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
        <ScalarProperty Name="PersonID" ParameterName="PersonID" />
      </DeleteFunction>
    </ModificationFunctionMapping>
  </EntityTypeMapping>
</EntitySetMapping>
```

### Exemple

L'exemple suivant montre le mappage de l'ensemble d'associations pour l'ensemble d'associations **CourseInstructor** dans le modèle School. En plus du mappage de colonnes pour l'Association **CourseInstructor**, le mappage des fonctions d'insertion et de suppression de l'Association **CourseInstructor** est affiché. Les fonctions mappées sont déclarées dans le modèle de stockage.

```

<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>

```

## QueryView, élément (MSL)

L'élément **QueryView** en Mapping Specification Language (MSL) définit un mappage en lecture seule entre un type d'entité ou une association dans le modèle conceptuel et une table dans la base de données sous-jacente. Le mappage est défini avec une requête Entity SQL qui est évaluée par rapport au modèle de stockage, et vous exprimez le jeu de résultats en termes d'entité ou d'association dans le modèle conceptuel. Les affichages des requêtes étant en lecture seule, les types qu'ils définissent ne peuvent pas être mis à jour au moyen des commandes de mise à jour standard. Les mises à jour de ces types peuvent être effectuées au moyen de fonctions de modification. Pour plus d'informations, consultez Comment : mapper des fonctions de modification à des procédures stockées.

### NOTE

Dans l'élément **QueryView**, Entity SQL expressions qui contiennent des propriétés **GroupBy**, Group ou de navigation ne sont pas prises en charge.

L'élément **QueryView** peut être un enfant de l'élément **EntityTypeMapping** ou de l'élément **AssociationSetMapping**. Dans le cas précédent, l'affichage des requêtes définit un mappage en lecture seule pour une entité dans le modèle conceptuel. Dans le cas précédent, l'affichage des requêtes définit un mappage en lecture seule pour une association dans le modèle conceptuel.

#### NOTE

Si l'élément **AssociationSetMapping** est destiné à une association avec une contrainte référentielle, l'élément **AssociationSetMapping** est ignoré. Pour plus d'informations, consultez **ReferentialConstraint**, élément (CSDL).

L'élément **QueryView** ne peut pas avoir d'éléments enfants.

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **QueryView**.

| NOM D'ATTRIBUT  | REQUIS | VALEUR   |
|-----------------|--------|--|
| <b>TypeName</b> | Non    | Nom du type de modèle conceptuel mappé par l'affichage des requêtes. |

#### Exemple

L'exemple suivant montre l'élément **QueryView** en tant qu'enfant de l'élément **EntitySetMapping** et définit un mappage d'affichage des requêtes pour le type d'entité **Department** dans le modèle School.

```
<EntitySetMapping Name="Departments" >
  <QueryView>
    SELECT VALUE SchoolModel.Department(d.DepartmentID,
      d.Name,
      d.Budget,
      d.StartDate)
    FROM SchoolModelStoreContainer.Department AS d
    WHERE d.Budget > 150000
  </QueryView>
</EntitySetMapping>
```

Étant donné que la requête retourne uniquement un sous-ensemble des membres du type de **service** dans le modèle de stockage, le type de **service** du modèle School a été modifié en fonction de ce mappage comme suit :

```
<EntityType Name="Department">
  <Key>
    <PropertyRef Name="DepartmentID" />
  </Key>
  <Property Type="Int32" Name="DepartmentID" Nullable="false" />
  <Property Type="String" Name="Name" Nullable="false"
    MaxLength="50" FixedLength="false" Unicode="true" />
  <Property Type="Decimal" Name="Budget" Nullable="false"
    Precision="19" Scale="4" />
  <Property Type="DateTime" Name="StartDate" Nullable="false" />
  <NavigationProperty Name="Courses"
    Relationship="SchoolModel.FK_Course_Department"
    FromRole="Department" ToRole="Course" />
</EntityType>
```

#### Exemple

L'exemple suivant montre l'élément **QueryView** en tant qu'enfant d'un élément **AssociationSetMapping** et définit un mappage en lecture seule pour l'Association **FK\_Course\_Department** dans le modèle School.

```

<EntityContainerMapping StorageEntityContainer="SchoolModelStoreContainer"
    CdmEntityContainer="SchoolEntities">
    <EntitySetMapping Name="Courses" >
        <QueryView>
            SELECT VALUE SchoolModel.Course(c.CourseID,
                c.Title,
                c.Credits)
            FROM SchoolModelStoreContainer.Course AS c
        </QueryView>
    </EntitySetMapping>
    <EntitySetMapping Name="Departments" >
        <QueryView>
            SELECT VALUE SchoolModel.Department(d.DepartmentID,
                d.Name,
                d.Budget,
                d.StartDate)
            FROM SchoolModelStoreContainer.Department AS d
            WHERE d.Budget > 150000
        </QueryView>
    </EntitySetMapping>
    <AssociationSetMapping Name="FK_Course_Department" >
        <QueryView>
            SELECT VALUE SchoolModel.FK_Course_Department(
                CREATEREF(SchoolEntities.Departments, row(c.DepartmentID), SchoolModel.Department),
                CREATEREF(SchoolEntities.Courses, row(c.CourseID)) )
            FROM SchoolModelStoreContainer.Course AS c
        </QueryView>
    </AssociationSetMapping>
</EntityContainerMapping>

```

## Commentaires

Vous pouvez définir des affichages des requêtes pour activer les scénarios suivants :

- Définir une entité du modèle conceptuel qui n'inclut pas toutes les propriétés de l'entité dans le modèle de stockage. Cela comprend les propriétés qui n'ont pas de valeurs par défaut et qui ne prennent pas en charge les valeurs **null**.
- Mapper des colonnes calculées du modèle de stockage aux propriétés de types d'entités du modèle conceptuel.
- Définir un mappage dans lequel les conditions utilisées pour partitionner des entités du modèle conceptuel ne sont pas basées sur l'égalité. Lorsque vous spécifiez un mappage conditionnel à l'aide de l'élément **condition**, la condition fournie doit être égale à la valeur spécifiée. Pour plus d'informations, consultez élément **condition** (MSL).
- Mapper la même colonne du modèle de stockage à plusieurs types du modèle conceptuel.
- Mapper plusieurs types à la même table.
- Définir des associations dans le modèle conceptuel qui ne sont pas basées sur des clés étrangères du schéma relationnel.
- Utiliser une logique métier personnalisée pour définir la valeur de propriétés du modèle conceptuel. Par exemple, vous pouvez mapper la valeur de chaîne « T » dans la source de données à la valeur **true**, une valeur booléenne, dans le modèle conceptuel.
- Définir des filtres conditionnels pour les résultats de la requête.
- Appliquer moins de restrictions sur les données dans le modèle conceptuel que dans le modèle de stockage. Par exemple, vous pouvez rendre une propriété dans le modèle conceptuel Nullable même si la colonne à laquelle elle est mappée ne prend pas en charge les valeurs **null**.

Vous devez tenir compte des points suivants lorsque vous définissez des affichages des requêtes pour les entités :

- Les affichages des requêtes sont en lecture seule. Les mises à jour des entités ne peuvent être effectuées qu'au

moyen de fonctions de modification.

- Lorsque vous définissez un type d'entité par un affichage des requêtes, vous devez également définir toutes les entités associées par les affichages des requêtes.
- Lorsque vous mappez une association plusieurs-à-plusieurs à une entité dans le modèle de stockage qui représente une table de liens dans le schéma relationnel, vous devez définir un élément **QueryView** dans l'élément **AssociationSetMapping** pour cette table de liens.
- Les affichages des requêtes doivent être définis pour tous les types d'une hiérarchie des types. Pour ce faire, vous pouvez procéder de différentes façons :
  - Avec un seul élément **QueryView** qui spécifie une seule requête Entity SQL qui retourne une Union de tous les types d'entité dans la hiérarchie.
  - Avec un seul élément **QueryView** qui spécifie une requête Entity SQL unique qui utilise l'opérateur case pour retourner un type d'entité spécifique dans la hiérarchie en fonction d'une condition spécifique.
  - Avec un élément **QueryView** supplémentaire pour un type spécifique dans la hiérarchie. Dans ce cas, utilisez l'attribut **TypeName** de l'élément **QueryView** pour spécifier le type d'entité pour chaque vue.
- Quand une vue de requête est définie, vous ne pouvez pas spécifier l'attribut **StorageSetName** sur l'élément **EntitySetMapping**.
- Quand une vue de requête est définie, l'élément **EntitySetMapping** ne peut pas contenir également des mappages de **Propriétés**.

## Élément ResultBinding (MSL)

L'élément **ResultBinding** en Mapping Specification Language (MSL) mappe les valeurs de colonne retournées par les procédures stockées aux propriétés d'entité dans le modèle conceptuel lorsque les fonctions de modification de type d'entité sont mappées aux procédures stockées dans la base de données sous-jacente. Par exemple, lorsque la valeur d'une colonne d'identité est retournée par une procédure stockée Insert, l'élément **ResultBinding** mappe la valeur retournée à une propriété de type d'entité dans le modèle conceptuel.

L'élément **ResultBinding** peut être un enfant de l'élément **InsertFunction** ou de l'élément **UpdateFunction**.

L'élément **ResultBinding** ne peut pas avoir d'éléments enfants.

### Attributs applicables

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **ResultBinding** :

| NOM D'ATTRIBUT    | REQUIS | VALEUR  |
|-------------------|--------|---|
| <b>Nom</b>        | Oui    | Nom de la propriété d'entité dans le modèle conceptuel mappé. |
| <b>ColumnName</b> | Oui    | Nom de la colonne mappée.                                     |

### Exemple

L'exemple suivant est basé sur le modèle School et montre un élément **InsertFunction** utilisé pour mapper la fonction insert du type d'entité **Person** à la procédure stockée **InsertPerson**. (La procédure stockée **InsertPerson** est indiquée ci-dessous et est déclarée dans le modèle de stockage.) Un élément **ResultBinding** est utilisé pour mapper une valeur de colonne retournée par la procédure stockée (**NewPersonID**) à une propriété de type d'entité (**PersonID**).

```

<EntityTypeMapping TypeName="SchoolModel.Person">
  <ModificationFunctionMapping>
    <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName" />
      <ScalarProperty Name="LastName" ParameterName="LastName" />
      <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
    </InsertFunction>
    <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate"
                      Version="Current" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate"
                      Version="Current" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName"
                      Version="Current" />
      <ScalarProperty Name="LastName" ParameterName="LastName"
                      Version="Current" />
      <ScalarProperty Name="PersonID" ParameterName="PersonID"
                      Version="Current" />
    </UpdateFunction>
    <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
      <ScalarProperty Name="PersonID" ParameterName="PersonID" />
    </DeleteFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>

```

Le code Transact-SQL suivant décrit la procédure stockée **InsertPerson** :

```

CREATE PROCEDURE [dbo].[InsertPerson]
    @LastName nvarchar(50),
    @FirstName nvarchar(50),
    @HireDate datetime,
    @EnrollmentDate datetime
    AS
    INSERT INTO dbo.Person (LastName,
                           FirstName,
                           HireDate,
                           EnrollmentDate)
    VALUES (@LastName,
            @FirstName,
            @HireDate,
            @EnrollmentDate);
    SELECT SCOPE_IDENTITY() as NewPersonID;

```

## Élément ResultMapping (MSL)

L'élément **ResultMapping** en Mapping Specification Language (MSL) définit le mappage entre une importation de fonction dans le modèle conceptuel et une procédure stockée dans la base de données sous-jacente lorsque les conditions suivantes sont remplies :

- L'importation de fonction retourne un type d'entité de modèle conceptuel ou le type complexe.
- Les noms des colonnes renvoyées par la procédure stockée ne correspondent pas exactement aux noms des propriétés sur le type d'entité ou le type complexe.

Par défaut, le mappage entre les colonnes renvoyées par une procédure stockée et un type d'entité ou un type complexe est basé sur les noms de colonne et de propriété. Si les noms de colonne ne correspondent pas exactement aux noms de propriété, vous devez utiliser l'élément **ResultMapping** pour définir le mappage. Pour obtenir un exemple du mappage par défaut, consultez élément **FunctionImportMapping** (MSL).

L'élément **ResultMapping** est un élément enfant de l'élément **FunctionImportMapping**.

L'élément **ResultMapping** peut avoir les éléments enfants suivants :

- **EntityTypeMapping** (zéro, un ou plusieurs éléments) ;
- **ComplexTypeMapping**

Aucun attribut n'est applicable à l'élément **ResultMapping**.

### Exemple

Considérons la procédure stockée suivante :

```
CREATE PROCEDURE [dbo].[GetGrades]
    @student_Id int
    AS
        SELECT      EnrollmentID as enroll_id,
                    Grade as grade,
                    CourseID as course_id,
                    StudentID as student_id
        FROM dbo.StudentGrade
        WHERE StudentID = @student_Id
```

De même, considérons le type d'entité de modèle conceptuel suivant :

```
<EntityType Name="StudentGrade">
    <Key>
        <PropertyRef Name="EnrollmentID" />
    </Key>
    <Property Name="EnrollmentID" Type="Int32" Nullable="false"
              annotation:StoreGeneratedPattern="Identity" />
    <Property Name="CourseID" Type="Int32" Nullable="false" />
    <Property Name="StudentID" Type="Int32" Nullable="false" />
    <Property Name="Grade" Type="Decimal" Precision="3" Scale="2" />
</EntityType>
```

Afin de créer une importation de fonction qui retourne des instances du type d'entité précédent, le mappage entre les colonnes rentrées par la procédure stockée et le type d'entité doit être défini dans un élément

### ResultMapping

```
<FunctionImportMapping FunctionImportName="GetGrades"
                           FunctionName="SchoolModel.Store.GetGrades" >
    <ResultMapping>
        <EntityTypeMapping TypeName="SchoolModel.StudentGrade">
            <ScalarProperty Name="EnrollmentID" ColumnName="enroll_id"/>
            <ScalarProperty Name="CourseID" ColumnName="course_id"/>
            <ScalarProperty Name="StudentID" ColumnName="student_id"/>
            <ScalarProperty Name="Grade" ColumnName="grade"/>
        </EntityTypeMapping>
    </ResultMapping>
</FunctionImportMapping>
```

## ScalarProperty, élément (MSL)

L'élément **ScalarProperty** en Mapping Specification Language (MSL) mappe une propriété sur un type d'entité de modèle conceptuel, un type complexe ou une association à une colonne de table ou un paramètre de procédure stockée dans la base de données sous-jacente.

#### NOTE

Les procédures stockées auxquelles des fonctions de modification sont mappées doivent être déclarées dans le modèle de stockage. Pour plus d'informations, consultez Function, élément (SSDL).

L'élément **ScalarProperty** peut être un enfant des éléments suivants :

- MappingFragment
- InsertFunction
- UpdateFunction
- DeleteFunction
- EndProperty
- ComplexProperty
- ResultMapping

En tant qu'enfant de l'élément **MappingFragment**, **ComplexProperty** ou **EndProperty**, l'élément **ScalarProperty** mappe une propriété du modèle conceptuel à une colonne de la base de données. En tant qu'enfant de l'élément **InsertFunction**, **UpdateFunction** ou **DeleteFunction**, l'élément **ScalarProperty** mappe une propriété du modèle conceptuel à un paramètre de procédure stockée.

L'élément **ScalarProperty** ne peut pas avoir d'éléments enfants.

#### Attributs applicables

Les attributs qui s'appliquent à l'élément **ScalarProperty** varient en fonction du rôle de l'élément.

Le tableau suivant décrit les attributs qui s'appliquent lorsque l'élément **ScalarProperty** est utilisé pour mapper une propriété de modèle conceptuel à une colonne de la base de données :

| NOM D'ATTRIBUT    | REQUIS | VALEUR   |
|-------------------|--------|--|
| <b>Nom</b>        | Oui    | Nom de la propriété de modèle conceptuel mappée. |
| <b>ColumnName</b> | Oui    | Nom de la colonne de table mappée.               |

Le tableau suivant décrit les attributs qui s'appliquent à l'élément **ScalarProperty** lorsqu'il est utilisé pour mapper une propriété de modèle conceptuel à un paramètre de procédure stockée :

| NOM D'ATTRIBUT       | REQUIS | VALEUR  |
|----------------------|--------|---|
| <b>Nom</b>           | Oui    | Nom de la propriété de modèle conceptuel mappée.  |
| <b>ParameterName</b> | Oui    | Nom du paramètre mappé.   |
| <b>Version</b>       | Non    | <b>Actuel</b> ou <b>original</b> selon que la valeur actuelle ou la valeur d'origine de la propriété doit être utilisée pour les contrôles d'accès concurrentiel. |

#### Exemple

L'exemple suivant montre l'élément **ScalarProperty** utilisé de deux manières :

- Pour mapper les propriétés du type d'entité **Person** aux colonnes de la table **Person**.

- Pour mapper les propriétés du type d'entité **Person** aux paramètres de la procédure stockée **UpdatePerson** . Les procédures stockées sont déclarées dans le modèle de stockage.

```

<EntitySetMapping Name="People">
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <MappingFragment StoreEntitySet="Person">
      <ScalarProperty Name="PersonID" ColumnName="PersonID" />
      <ScalarProperty Name="LastName" ColumnName="LastName" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="HireDate" ColumnName="HireDate" />
      <ScalarProperty Name="EnrollmentDate"
                     ColumnName="EnrollmentDate" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="SchoolModel.Person">
    <ModificationFunctionMapping>
      <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
        <ScalarProperty Name="EnrollmentDate"
                       ParameterName="EnrollmentDate" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName" />
        <ScalarProperty Name="LastName" ParameterName="LastName" />
        <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
      </InsertFunction>
      <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
        <ScalarProperty Name="EnrollmentDate"
                       ParameterName="EnrollmentDate"
                       Version="Current" />
        <ScalarProperty Name="HireDate" ParameterName="HireDate"
                       Version="Current" />
        <ScalarProperty Name="FirstName" ParameterName="FirstName"
                       Version="Current" />
        <ScalarProperty Name="LastName" ParameterName="LastName"
                       Version="Current" />
        <ScalarProperty Name="PersonID" ParameterName="PersonID"
                       Version="Current" />
      </UpdateFunction>
      <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
        <ScalarProperty Name="PersonID" ParameterName="PersonID" />
      </DeleteFunction>
    </ModificationFunctionMapping>
  </EntityTypeMapping>
</EntitySetMapping>

```

## Exemple

L'exemple suivant montre l'élément **ScalarProperty** utilisé pour mapper les fonctions d'insertion et de suppression d'une association de modèle conceptuel à des procédures stockées dans la base de données. Les procédures stockées sont déclarées dans le modèle de stockage.

```

<AssociationSetMapping Name="CourseInstructor"
    TypeName="SchoolModel.CourseInstructor"
    StoreEntitySet="CourseInstructor">
    <EndProperty Name="Person">
        <ScalarProperty Name="PersonID" ColumnName="PersonID" />
    </EndProperty>
    <EndProperty Name="Course">
        <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </EndProperty>
    <ModificationFunctionMapping>
        <InsertFunction FunctionName="SchoolModel.Store.InsertCourseInstructor" >
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </InsertFunction>
        <DeleteFunction FunctionName="SchoolModel.Store.DeleteCourseInstructor">
            <EndProperty Name="Course">
                <ScalarProperty Name="CourseID" ParameterName="courseId"/>
            </EndProperty>
            <EndProperty Name="Person">
                <ScalarProperty Name="PersonID" ParameterName="instructorId"/>
            </EndProperty>
        </DeleteFunction>
    </ModificationFunctionMapping>
</AssociationSetMapping>

```

## UpdateFunction, élément (MSL)

L’élément **UpdateFunction** en Mapping Specification Language (MSL) mappe la fonction de mise à jour d’un type d’entité dans le modèle conceptuel à une procédure stockée dans la base de données sous-jacente. Les procédures stockées auxquelles des fonctions de modification sont mappées doivent être déclarées dans le modèle de stockage. Pour plus d’informations, consultez Function, élément (SSDL).

### NOTE

Si vous ne mappez pas les trois opérations d’insertion, de mise à jour ou de suppression d’un type d’entité à des procédures stockées, les opérations non mappées échouent si elles sont exécutées au moment de l’exécution et qu’une UpdateException est levée.

L’élément **UpdateFunction** peut être un enfant de l’élément ModificationFunctionMapping et être appliqué à l’élément EntityTypeMapping.

L’élément **UpdateFunction** peut avoir les éléments enfants suivants :

- AssociationEnd (zéro, un ou plusieurs)
- ComplexProperty (zéro, un ou plusieurs éléments) ;
- ResultBinding (zéro ou un)
- ScalarProperty (zéro ou plus)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l’élément **UpdateFunction**.

| NOM D'ATTRIBUT | REQUIS | VALEUR |
|----------------|--------|--------|
|----------------|--------|--------|

| NOM D'ATTRIBUT               | REQUIS | VALEUR   |
|------------------------------|--------|--|
| <b>FunctionName</b>          | Oui    | Nom qualifié par un espace de noms de la procédure stockée à laquelle la fonction de mise à jour est mappée. La procédure stockée doit être déclarée dans le modèle de stockage. |
| <b>RowsAffectedParameter</b> | Non    | Nom du paramètre de sortie qui retourne le nombre de lignes affectées.   |

## Exemple

L'exemple suivant est basé sur le modèle School et montre l'élément **UpdateFunction** utilisé pour mapper la fonction de mise à jour du type d'entité **Person** à la procédure stockée **UpdatePerson**. La procédure stockée **UpdatePerson** est déclarée dans le modèle de stockage.

```
<EntityTypeMapping TypeName="SchoolModel.Person">
  <ModificationFunctionMapping>
    <InsertFunction FunctionName="SchoolModel.Store.InsertPerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName" />
      <ScalarProperty Name="LastName" ParameterName="LastName" />
      <ResultBinding Name="PersonID" ColumnName="NewPersonID" />
    </InsertFunction>
    <UpdateFunction FunctionName="SchoolModel.Store.UpdatePerson">
      <ScalarProperty Name="EnrollmentDate"
                      ParameterName="EnrollmentDate"
                      Version="Current" />
      <ScalarProperty Name="HireDate" ParameterName="HireDate"
                      Version="Current" />
      <ScalarProperty Name="FirstName" ParameterName="FirstName"
                      Version="Current" />
      <ScalarProperty Name="LastName" ParameterName="LastName"
                      Version="Current" />
      <ScalarProperty Name="PersonID" ParameterName="PersonID"
                      Version="Current" />
    </UpdateFunction>
    <DeleteFunction FunctionName="SchoolModel.Store.DeletePerson">
      <ScalarProperty Name="PersonID" ParameterName="PersonID" />
    </DeleteFunction>
  </ModificationFunctionMapping>
</EntityTypeMapping>
```

# Spécification SSDL

23/11/2019 • 61 minutes to read

SSDL (Store Schema Definition Language) est un langage basé sur XML qui décrit le modèle de stockage d'une application Entity Framework.

Dans une application Entity Framework, les métadonnées du modèle de stockage sont chargées à partir d'un fichier. SSDL (écrit en SSDL) dans une instance de System. Data. Metadata. Edm. StoreItemCollection et sont accessibles à l'aide des méthodes de la Classe System. Data. Metadata. Edm. MetadataWorkspace. Entity Framework utilise les métadonnées du modèle de stockage pour traduire les requêtes sur le modèle conceptuel en commandes spécifiques au stockage.

Le Entity Framework Designer (concepteur EF) stocke les informations de modèle de stockage dans un fichier. edmx au moment de la conception. Au moment de la génération, le Entity Designer utilise les informations d'un fichier. edmx pour créer le fichier. ssdl requis par Entity Framework au moment de l'exécution.

Les versions de SSDL sont différenciées par les espaces de noms XML.

| VERSION SSDL | ESPACE DE NOMS XML  |
|--------------|---|
| SSDL v1      | <a href="https://schemas.microsoft.com/ado/2006/04/edm/ssdl">https://schemas.microsoft.com/ado/2006/04/edm/ssdl</a> |
| SSDL v2      | <a href="https://schemas.microsoft.com/ado/2009/02/edm/ssdl">https://schemas.microsoft.com/ado/2009/02/edm/ssdl</a> |
| SSDL v3      | <a href="https://schemas.microsoft.com/ado/2009/11/edm/ssdl">https://schemas.microsoft.com/ado/2009/11/edm/ssdl</a> |

## Association, élément (SSDL)

Un élément **Association** en Store Schema Definition Language (SSDL) spécifie des colonnes de table qui participent à une contrainte de clé étrangère dans la base de données sous-jacente. Deux éléments End enfants obligatoires spécifient des tables aux terminaisons de l'association et la multiplicité à chaque terminaison. Un élément ReferentialConstraint facultatif spécifie les terminaisons principales et dépendantes de l'association ainsi que les colonnes participantes. Si aucun élément **ReferentialConstraint** n'est présent, un élément AssociationSetMapping doit être utilisé pour spécifier les mappages de colonnes pour l'Association.

L'élément **Association** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un)
- End (exactement deux)
- ReferentialConstraint (zéro ou un)
- éléments d'annotation (zéro, un ou plusieurs).

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **Association**.

| NOM D'ATTRIBUT | REQUIS | VALEUR   |
|----------------|--------|--|
| <b>Nom</b>     | Oui    | Nom de la contrainte de clé étrangère correspondante dans la base de données sous-jacente. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Association**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **Association** qui utilise un élément **ReferentialConstraint** pour spécifier les colonnes qui participent à la contrainte de clé étrangère **FK\_CustomerOrders** :

```
<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
      <OnDelete Action="Cascade" />
    </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

## AssociationSet, élément (SSDL)

L'élément **AssociationSet** en Store Schema Definition Language (SSDL) représente une contrainte de clé étrangère entre deux tables dans la base de données sous-jacente. Les colonnes de la table qui participent à la contrainte de clé étrangère sont spécifiées dans un élément **Association**. L'élément **Association** qui correspond à un élément **AssociationSet** donné est spécifié dans l'attribut **Association** de l'élément **AssociationSet**.

Les ensembles d'associations SSDL sont mappés aux ensembles d'associations CSDL par un élément **AssociationSetMapping**. Toutefois, si l'**Association** CSDL pour un ensemble d'associations CSDL donné est définie à l'aide d'un élément **ReferentialConstraint**, aucun élément **AssociationSetMapping** correspondant n'est nécessaire. Dans ce cas, si un élément **AssociationSetMapping** est présent, les mappages qu'il définit seront remplacés par l'élément **ReferentialConstraint**.

L'élément **AssociationSet** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un)
- End (zéro ou deux éléments) ;
- éléments d'annotation (zéro, un ou plusieurs).

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **AssociationSet**.

| NOM D'ATTRIBUT | REQUIS | VALEUR  |
|----------------|--------|---|
| <b>Nom</b>     | Oui    | Nom de la contrainte de clé étrangère que l'ensemble d'associations représente. |

| NOM D'ATTRIBUT     | REQUIS | VALEUR  |
|--------------------|--------|---|
| <b>Association</b> | Oui    | Nom de l'association qui définit les colonnes qui participent à la contrainte de clé étrangère. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **AssociationSet**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **AssociationSet** qui représente la contrainte de clé étrangère `FK_CustomerOrders` dans la base de données sous-jacente :

```
<AssociationSet Name="FK_CustomerOrders"
    Association="ExampleModel.Store.FK_CustomerOrders">
    <End Role="Customers" EntitySet="Customers" />
    <End Role="Orders" EntitySet="Orders" />
</AssociationSet>
```

## Élément CollectionType (SSDL)

L'élément **CollectionType** en Store Schema Definition Language (SSDL) spécifie que le type de retour d'une fonction est une collection. L'élément **CollectionType** est un enfant de l'élément **ReturnType**. Le type de collection est spécifié à l'aide de l'élément enfant **RowType** :

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **CollectionType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre une fonction qui utilise un élément **CollectionType** pour spécifier que la fonction retourne une collection de lignes.

```
<Function Name="GetProducts" IsComposable="true" Schema="dbo">
    <ReturnType>
        <CollectionType>
            <RowType>
                <Property Name="ProductID" Type="int" Nullable="false" />
                <Property Name="CategoryID" Type="bigint" Nullable="false" />
                <Property Name="ProductName" Type="nvarchar" MaxLength="40" Nullable="false" />
                <Property Name="UnitPrice" Type="money" />
                <Property Name="Discontinued" Type="bit" />
            </RowType>
        </CollectionType>
    </ReturnType>
</Function>
```

## CommandText, élément (SSDL)

L'élément **CommandText** en Store Schema Definition Language (SSDL) est un enfant de l'élément Function qui vous permet de définir une instruction SQL qui est exécutée au niveau de la base de données. L'élément **CommandText** vous permet d'ajouter des fonctionnalités qui sont similaires à une procédure stockée dans la base de données, mais vous définissez l'élément **CommandText** dans le modèle de stockage.

L'élément **CommandText** ne peut pas avoir d'éléments enfants. Le corps de l'élément **CommandText** doit être une instruction SQL valide pour la base de données sous-jacente.

Aucun attribut n'est applicable à l'élément **CommandText**.

### Exemple

L'exemple suivant montre un élément **Function** avec un élément **CommandText** enfant. Exposez la fonction **UpdateProductInOrder** en tant que méthode sur ObjectContext en l'important dans le modèle conceptuel.

```
<Function Name="UpdateProductInOrder" IsComposable="false">
    <CommandText>
        UPDATE Orders
        SET ProductId = @productId
        WHERE OrderId = @orderId;
    </CommandText>
    <Parameter Name="productId"
        Mode="In"
        Type="int"/>
    <Parameter Name="orderId"
        Mode="In"
        Type="int"/>
</Function>
```

## DefiningQuery, élément (SSDL)

L'élément **DefiningQuery** en Store Schema Definition Language (SSDL) vous permet d'exécuter une instruction SQL directement dans la base de données sous-jacente. L'élément **DefiningQuery** est couramment utilisé comme une vue de base de données, mais la vue est définie dans le modèle de stockage au lieu de la base de données. La vue définie dans un élément **DefiningQuery** peut être mappée à un type d'entité dans le modèle conceptuel via un élément EntitySetMapping. Ces mappages sont en lecture seule.

La syntaxe SSDL suivante illustre la déclaration d'un **EntitySet** suivi de l'élément **DefiningQuery** qui contient une requête utilisée pour récupérer la vue.

```
<Schema>
    <EntitySet Name="Tables" EntityType="Self.STable">
        <DefiningQuery>
            SELECT TABLE_CATALOG,
                'test' as TABLE_SCHEMA,
                TABLE_NAME
            FROM INFORMATION_SCHEMA.TABLES
        </DefiningQuery>
    </EntitySet>
</Schema>
```

Vous pouvez utiliser des procédures stockées dans le Entity Framework pour activer des scénarios de lecture-écriture sur des vues. Vous pouvez utiliser une vue de source de données ou une vue de Entity SQL comme table de base pour récupérer des données et pour le traitement des modifications par des procédures stockées.

Vous pouvez utiliser l'élément **DefiningQuery** pour cibler Microsoft SQL Server Compact 3,5. Bien que SQL Server Compact 3,5 ne prenne pas en charge les procédures stockées, vous pouvez implémenter des fonctionnalités similaires avec l'élément **DefiningQuery**. Cet élément peut s'avérer également utile pour créer des procédures stockées afin de surmonter une incompatibilité entre les types de données utilisés dans le langage

de programmation et ceux de la source de données. Vous pouvez écrire un **DefiningQuery** qui accepte un certain ensemble de paramètres, puis appelle une procédure stockée avec un jeu de paramètres différent, par exemple, une procédure stockée qui supprime des données.

## Élément Dependent (SSDL)

L'élément **dépendant** en Store Schema Definition Language (SSDL) est un élément enfant de l'élément **ReferentialConstraint** qui définit la terminaison dépendante d'une contrainte de clé étrangère (également appelée contrainte référentielle). L'élément **dépendant** spécifie la ou les colonnes d'une table qui font référence à une ou plusieurs colonnes de clé primaire. Les éléments **PropertyRef** spécifient les colonnes qui sont référencées. L'élément principal spécifie les colonnes de clés primaires référencées par les colonnes spécifiées dans l'élément **dépendant**.

L'élément **dépendant** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- **PropertyRef** (un ou plusieurs) ;
- éléments d'annotation (zéro, un ou plusieurs).

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **dépendant**.

| NOM D'ATTRIBUT | REQUIS | VALEUR  |
|----------------|--------|---|
| Rôle           | Oui    | La même valeur que l'attribut de <b>rôle</b> (s'il est utilisé) de l'élément de fin correspondant ; dans le cas contraire, il s'agit du nom de la table qui contient la colonne de référence. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **dépendant**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **Association** qui utilise un élément **ReferentialConstraint** pour spécifier les colonnes qui participent à la contrainte de clé étrangère **FK\_**. L'élément **dépendant** spécifie la colonne **CustomerID** de la table **Order** comme terminaison dépendante de la contrainte.

```
<Association Name="FK_CustomerOrders">
    <End Role="Customers"
        Type="ExampleModel.Store.Customers" Multiplicity="1">
        <OnDelete Action="Cascade" />
    </End>
    <End Role="Orders"
        Type="ExampleModel.Store.Orders" Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="Customers">
            <PropertyRef Name="CustomerId" />
        </Principal>
        <Dependent Role="Orders">
            <PropertyRef Name="CustomerId" />
        </Dependent>
    </ReferentialConstraint>
</Association>
```

## Documentation, élément (SSDL)

L'élément **documentation** en Store Schema Definition Language (SSDL) peut être utilisé pour fournir des informations sur un objet défini dans un élément parent.

L'élément **documentation** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- **Summary**: brève description de l'élément parent. (zéro ou un élément).
- **LongDescription**: description complète de l'élément parent. (zéro ou un élément).

### Attributs applicables

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **documentation**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre l'élément de **documentation** en tant qu'élément enfant d'un élément EntityType.

```
<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>
```

## End, élément (SSDL)

L'élément **end** en Store Schema Definition Language (SSDL) spécifie la table et le nombre de lignes à une terminaison d'une contrainte FOREIGN KEY dans la base de données sous-jacente. L'élément **end** peut être un enfant de l'élément Association ou de l'élément AssociationSet. Dans chaque cas, les éléments enfants et les attributs applicables possibles sont différents.

### Élément End comme enfant de l'élément Association

Un élément **end** (en tant qu'enfant de l'élément **Association**) spécifie la table et le nombre de lignes à la fin d'une contrainte de clé étrangère, respectivement avec les attributs **type** et **Multiplicity**. Les terminaisons d'une contrainte de clé étrangère sont définies dans le cadre d'un ensemble d'associations SSDL ; un ensemble d'associations SSDL doit avoir exactement deux terminaisons.

Un élément **end** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- OnDelete (zéro ou un élément)
- Éléments d'annotation (zéro, un ou plusieurs éléments)

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **final** lorsqu'il est l'enfant d'un élément **Association**.

| NOM D'ATTRIBUT      | REQUIS | VALEUR  |
|---------------------|--------|---|
| <b>Type</b>         | Oui    | Nom complet du jeu d'entités SSDL qui est à la terminaison de la contrainte de clé étrangère.   |
| <b>Rôle</b>         | Non    | Valeur de l'attribut <b>role</b> dans l'élément principal ou dépendant de l'élément ReferentialConstraint correspondant (s'il est utilisé).   |
| <b>Multiplicité</b> | Oui    | <p><b>1, 0.. 1</b> ou * selon le nombre de lignes qui peuvent être à la fin de la contrainte de clé étrangère.<br/> <b>1</b> indique qu'une seule ligne existe à la fin de la contrainte de clé étrangère.<br/> <b>0.. 1</b> indique qu'il existe zéro ou une ligne à la fin de la contrainte de clé étrangère.<br/> * indique qu'il existe zéro, une ou plusieurs lignes à la fin de la contrainte de clé étrangère.</p> |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément de **fin**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **Association** qui définit la contrainte de clé étrangère **FK\_**. Les valeurs de **multiplicité** spécifiées sur chaque élément de **fin** indiquent que de nombreuses lignes de la table **Orders** peuvent être associées à une ligne de la table **Customers**, mais qu'une seule ligne de la table **Customers** peut être associée à une ligne dans la table **Orders**. En outre, l'élément **OnDelete** indique que toutes les lignes de la table **Orders** qui font référence à une ligne particulière de la table **Customers** seront supprimées si la ligne de la table **Customers** est supprimée.

```
<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
      <OnDelete Action="Cascade" />
    </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

#### Élément **End** comme enfant de l'élément **AssociationSet**

L'élément **end** (en tant qu'enfant de l'élément **AssociationSet**) spécifie une table à une terminaison d'une contrainte de clé étrangère dans la base de données sous-jacente.

Un élément **end** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un)
- éléments d'annotation (zéro, un ou plusieurs).

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **end** lorsqu'il est l'enfant d'un élément **AssociationSet**.

| NOM D'ATTRIBUT   | REQUIS | VALEUR   |
|------------------|--------|--|
| <b>EntitySet</b> | Oui    | Nom du jeu d'entités SSDL qui est à la terminaison de la contrainte de clé étrangère.                                      |
| <b>Rôle</b>      | Non    | Valeur de l'un des attributs de <b>rôle</b> spécifiés sur un élément de <b>fin</b> de l'élément Association correspondant. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément de **fin**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **EntityContainer** avec un élément **AssociationSet** avec deux éléments **end** :

```
<EntityContainer Name="ExampleModelStoreContainer">
    <EntitySet Name="Customers"
        EntityType="ExampleModel.Store.Customers"
        Schema="dbo" />
    <EntitySet Name="Orders"
        EntityType="ExampleModel.Store.Orders"
        Schema="dbo" />
    <AssociationSet Name="FK_CustomerOrders"
        Association="ExampleModel.Store.FK_CustomerOrders">
        <End Role="Customers" EntitySet="Customers" />
        <End Role="Orders" EntitySet="Orders" />
    </AssociationSet>
</EntityContainer>
```

## EntityContainer, élément (SSDL)

Un élément **EntityContainer** en Store Schema Definition Language (SSDL) décrit la structure de la source de données sous-jacente dans une application Entity Framework : les jeux d'entités SSDL (définis dans les éléments **EntitySet**) représentent les tables d'une base de données, les types d'entités SSDL (définis dans les éléments **EntityType**) représentent les lignes d'une table, et les ensembles d'associations (définis dans les éléments **AssociationSet**) représentent des contraintes. Un conteneur d'entités de modèle de stockage est mappé à un conteneur d'entités de modèle conceptuel via l'élément **EntityContainerMapping**.

Un élément **EntityContainer** peut avoir zéro ou un élément documentation. Si un élément de **documentation** est présent, il doit précéder tous les autres éléments enfants.

Un élément **EntityContainer** peut avoir zéro ou plusieurs des éléments enfants suivants (dans l'ordre indiqué) :

- EntitySet ;
- AssociationSet ;
- éléments d'annotation.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **EntityContainer**.

| NOM D'ATTRIBUT | REQUIS | VALEUR  |
|----------------|--------|---|
| <b>Nom</b>     | Oui    | Nom du conteneur d'entités. Ce nom ne peut pas contenir de point (.). |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **EntityContainer**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **EntityContainer** qui définit deux jeux d'entités et un ensemble d'associations. Notez que les noms de type d'entité et de type d'association sont qualifiés par le nom de l'espace de noms du modèle conceptuel.

```
<EntityContainer Name="ExampleModelStoreContainer">
  <EntityType Name="Customers"
    EntityType="ExampleModel.Store.Customers"
    Schema="dbo" />
  <EntityType Name="Orders"
    EntityType="ExampleModel.Store.Orders"
    Schema="dbo" />
  <AssociationSet Name="FK_CustomerOrders"
    Association="ExampleModel.Store.FK_CustomerOrders">
    <End Role="Customers" EntitySet="Customers" />
    <End Role="Orders" EntitySet="Orders" />
  </AssociationSet>
</EntityContainer>
```

## EntityType, élément (SSDL)

Un élément **EntityType** en Store Schema Definition Language (SSDL) représente une table ou une vue dans la base de données sous-jacente. Un élément EntityType en SSDL représente une ligne dans la table ou la vue. L'attribut **EntityType** d'un élément **EntityType** spécifie le type d'entité SSDL particulier qui représente les lignes dans un jeu d'entités SSDL. Le mappage entre un jeu d'entités CSDL et un jeu d'entités SSDL est spécifié dans un élément EntitySetMapping.

L'élément **EntityType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- DefiningQuery (zéro ou un élément)
- éléments d'annotation.

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **EntityType**.

#### NOTE

Certains attributs (non répertoriés ici) peuvent être qualifiés avec l'alias du **magasin**. Ces attributs sont utilisés par l'Assistant Mise à jour du modèle lors de la mise à jour d'un modèle.

| NOM D'ATTRIBUT    | REQUIS | VALEUR   |
|-------------------|--------|--|
| <b>Nom</b>        | Oui    | Nom du jeu d'entités.  |
| <b>EntityType</b> | Oui    | Nom qualifié complet du type d'entité pour lequel le jeu d'entités contient des instances. |
| <b>Schéma</b>     | Non    | Schéma de base de données.   |
| <b>Table</b>      | Non    | Table de base de données.  |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **EntitySet**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **EntityContainer** qui a deux éléments **EntitySet** et un élément **AssociationSet**:

```
<EntityContainer Name="ExampleModelStoreContainer">
    <EntitySet Name="Customers"
        EntityType="ExampleModel.Store.Customers"
        Schema="dbo" />
    <EntitySet Name="Orders"
        EntityType="ExampleModel.Store.Orders"
        Schema="dbo" />
    <AssociationSet Name="FK_CustomerOrders"
        Association="ExampleModel.Store.FK_CustomerOrders">
        <End Role="Customers" EntitySet="Customers" />
        <End Role="Orders" EntitySet="Orders" />
    </AssociationSet>
</EntityContainer>
```

## EntityType, élément (SSDL)

Un élément **EntityType** en Store Schema Definition Language (SSDL) représente une ligne dans une table ou une vue de la base de données sous-jacente. Un élément EntitySet en SSDL représente la table ou la vue dans laquelle les lignes résultent. L'attribut **EntityType** d'un élément **EntitySet** spécifie le type d'entité SSDL particulier qui représente les lignes dans un jeu d'entités SSDL. Le mappage entre un type d'entité SSDL et un type d'entité CSDL est spécifié dans un élément EntityTypeMapping.

L'élément **EntityType** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un élément)
- Key (zéro ou un élément) ;
- éléments d'annotation.

## Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **EntityType**.

| NOM D'ATTRIBUT | REQUIS | VALEUR  |
|----------------|--------|---|
| <b>Nom</b>     | Oui    | Nom du type d'entité. Cette valeur est habituellement la même que le nom de la table dans laquelle le type d'entité représente une ligne. Cette valeur ne peut pas contenir de point (.). |

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **EntityType**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

## Exemple

L'exemple suivant montre un élément **EntityType** avec deux propriétés :

```
<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>
```

## Function, élément (SSDL)

L'élément **Function** de Store Schema Definition Language (SSDL) spécifie une procédure stockée qui existe dans la base de données sous-jacente.

L'élément **Function** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un)
- Paramètre (zéro, un ou plusieurs)
- CommandText (zéro ou un)
- ReturnType (zéro, un ou plusieurs)
- éléments d'annotation (zéro, un ou plusieurs).

Un type de retour pour une fonction doit être spécifié avec l'élément **ReturnType** ou l'attribut **ReturnType** (voir ci-dessous), mais pas les deux.

Les procédures stockées spécifiées dans le modèle de stockage peuvent être importées dans le modèle conceptuel d'une application. Pour plus d'informations, consultez [interrogation avec des procédures stockées](#). L'élément **Function** peut également être utilisé pour définir des fonctions personnalisées dans le modèle de stockage.

## Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **Function**.

#### NOTE

Certains attributs (non répertoriés ici) peuvent être qualifiés avec l'alias du **magasin**. Ces attributs sont utilisés par l'Assistant Mise à jour du modèle lors de la mise à jour d'un modèle.

| NOM D'ATTRIBUT                | REQUIS | VALEUR  |
|-------------------------------|--------|---|
| <b>Nom</b>                    | Oui    | Nom de la procédure stockée.  |
| <b>ReturnType</b>             | Non    | Type de retour de la procédure stockée.   |
| <b>Aggregate</b>              | Non    | <b>True</b> si la procédure stockée retourne une valeur d'agrégation ; Sinon, <b>false</b> .  |
| <b>Intégrée</b>               | Non    | <b>True</b> si la fonction est une fonction intégrée <sup>1</sup> ; Sinon, <b>false</b> .   |
| <b>StoreFunctionName</b>      | Non    | Nom de la procédure stockée.  |
| <b>NiladicFunction</b>        | Non    | <b>True</b> si la fonction est une fonction niladic <sup>2</sup> ; <b>False</b> dans le cas contraire.  |
| <b>IsComposable</b>           | Non    | <b>True</b> si la fonction est une fonction composable <sup>3</sup> ; <b>False</b> dans le cas contraire.   |
| <b>ParameterTypeSemantics</b> | Non    | Énumération qui définit la sémantique de type utilisée pour résoudre les surcharges de fonction. L'énumération est définie dans le manifeste du fournisseur par définition de fonction. La valeur par défaut est <b>AllowImplicitConversion</b> . |
| <b>Schéma</b>                 | Non    | Nom du schéma dans lequel une procédure stockée est définie.  |

<sup>1</sup> une fonction intégrée est une fonction définie dans la base de données. Pour plus d'informations sur les fonctions définies dans le modèle de stockage, consultez **CommandText**, élément (SSDL).

<sup>2</sup> une fonction niladic est une fonction qui n'accepte aucun paramètre et, lorsqu'elle est appelée, elle ne requiert pas de parenthèses.

<sup>3</sup> deux fonctions sont componables si la sortie d'une fonction peut être l'entrée de l'autre fonction.

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément de **fonction**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **Function** qui correspond à la procédure stockée **UpdateOrderQuantity**. La procédure stockée accepte deux paramètres et ne retourne pas de valeur.

```

<Function Name="UpdateOrderQuantity"
    Aggregate="false"
    BuiltIn="false"
    NiladicFunction="false"
    IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion"
    Schema="dbo">
    <Parameter Name="orderId" Type="int" Mode="In" />
    <Parameter Name="newQuantity" Type="int" Mode="In" />
</Function>

```

## Key, élément (SSDL)

L'élément **clé** en Store Schema Definition Language (SSDL) représente la clé primaire d'une table dans la base de données sous-jacente. **Key** est un élément enfant d'un élément **EntityType**, qui représente une ligne dans une table. La clé primaire est définie dans l'élément **Key** en référençant un ou plusieurs éléments **Property** définis sur l'élément **EntityType**.

L'élément **Key** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- **PropertyRef** (un ou plusieurs) ;
- éléments d'annotation.

Aucun attribut n'est applicable à l'élément **Key**.

### Exemple

L'exemple suivant montre un élément **EntityType** avec une clé qui référence une propriété :

```

<EntityType Name="Customers">
    <Documentation>
        <Summary>Summary here.</Summary>
        <LongDescription>Long description here.</LongDescription>
    </Documentation>
    <Key>
        <PropertyRef Name="CustomerId" />
    </Key>
    <Property Name="CustomerId" Type="int" Nullable="false" />
    <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>

```

## OnDelete, élément (SSDL)

L'élément **OnDelete** en Store Schema Definition Language (SSDL) reflète le comportement de la base de données lorsqu'une ligne qui participe à une contrainte de clé étrangère est supprimée. Si l'action est définie sur **cascade**, les lignes qui font référence à une ligne en cours de suppression seront également supprimées. Si l'action est définie sur **aucun**, les lignes qui font référence à une ligne en cours de suppression ne sont pas supprimées. Un élément **OnDelete** est un élément enfant d'un élément **end**.

Un élément **OnDelete** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un)
- éléments d'annotation (zéro, un ou plusieurs).

### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **OnDelete**.

| NOM D'ATTRIBUT | REQUIS | VALEUR  |
|----------------|--------|---|
| Action         | Oui    | <b>Cascade</b> ou <b>None</b> . (La valeur <b>Restricted</b> est valide mais a le même comportement qu' <b>aucun</b> .) |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **OnDelete**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **Association** qui définit la contrainte de clé étrangère **FK\_**. L'élément **OnDelete** indique que toutes les lignes de la table **Orders** qui font référence à une ligne particulière de la table **Customers** seront supprimées si la ligne de la table **Customers** est supprimée.

```
<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

## Élément Parameter (SSDL)

L'élément **Parameter** en Store Schema Definition Language (SSDL) est un enfant de l'élément **Function** qui spécifie les paramètres d'une procédure stockée dans la base de données.

L'élément **Parameter** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- Documentation (zéro ou un)
- éléments d'annotation (zéro, un ou plusieurs).

#### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **Parameter**.

| NOM D'ATTRIBUT | REQUIS | VALEUR   |
|----------------|--------|--|
| <b>Nom</b>     | Oui    | Nom du paramètre.  |
| <b>Type</b>    | Oui    | Type du paramètre.   |
| <b>Mode</b>    | Non    | <b>In</b> , <b>out</b> ou <b>INOUT</b> selon que le paramètre est un paramètre d'entrée, de sortie ou d'entrée/sortie. |

| NOM D'ATTRIBUT   | REQUIS | VALEUR   |
|------------------|--------|--|
| <b>MaxLength</b> | Non    | Longueur maximale du paramètre.  |
| <b>Précision</b> | Non    | Précision du paramètre.  |
| <b>Échelle</b>   | Non    | Échelle du paramètre.  |
| <b>SRID</b>      | Non    | Identificateur de référence système spatial. Valide uniquement pour les paramètres des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> . |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Parameter**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **Function** qui possède deux éléments **Parameter** qui spécifient des paramètres d'entrée :

```
<Function Name="UpdateOrderQuantity"
          Aggregate="false"
          BuiltIn="false"
          NiladicFunction="false"
          IsComposable="false"
          ParameterTypeSemantics="AllowImplicitConversion"
          Schema="dbo">
    <Parameter Name="orderId" Type="int" Mode="In" />
    <Parameter Name="newQuantity" Type="int" Mode="In" />
</Function>
```

## Élément Principal (SSDL)

L'élément **principal** en Store Schema Definition Language (SSDL) est un élément enfant de l'élément **ReferentialConstraint** qui définit la terminaison principale d'une contrainte de clé étrangère (également appelée contrainte référentielle). L'élément **principal** spécifie la ou les colonnes de clé primaire d'une table qui est référencée par une ou plusieurs colonnes. Les éléments **PropertyRef** spécifient les colonnes qui sont référencées. L'élément **dépendant** spécifie les colonnes qui réfèrent aux colonnes de clés primaires spécifiées dans l'élément **principal**.

L'élément **principal** peut avoir les éléments enfants suivants (dans l'ordre indiqué) :

- **PropertyRef** (un ou plusieurs) ;
- éléments d'annotation (zéro, un ou plusieurs).

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **principal**.

| NOM D'ATTRIBUT | REQUIS | VALEUR  |
|----------------|--------|---|
| Rôle           | Oui    | La même valeur que l'attribut de <b>rôle</b> (s'il est utilisé) de l'élément de fin correspondant ; Sinon, le nom de la table qui contient la colonne référencée. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **principal**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément Association qui utilise un élément **ReferentialConstraint** pour spécifier les colonnes qui participent à la contrainte de clé étrangère **FK\_**. L'élément **principal** spécifie la colonne **CustomerID** de la table **Customer** comme terminaison principale de la contrainte.

```
<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

## Property, élément (SSDL)

L'élément de **propriété** en Store Schema Definition Language (SSDL) représente une colonne dans une table de la base de données sous-jacente. Les éléments de **propriété** sont des enfants d'éléments **EntityType**, qui représentent des lignes dans une table. Chaque élément de **propriété** défini sur un élément **EntityType** représente une colonne.

Un élément de **propriété** ne peut pas avoir d'éléments enfants.

#### Attributs applicables

Le tableau suivant décrit les attributs qui peuvent être appliqués à l'élément **Property**.

| NOM D'ATTRIBUT | REQUIS | VALEUR                             |
|----------------|--------|------------------------------------|
| <b>Nom</b>     | Oui    | Nom de la colonne correspondante.  |
| <b>Type</b>    | Oui    | Type de la colonne correspondante. |

| NOM D'ATTRIBUT               | REQUIS | VALEUR  |
|------------------------------|--------|---|
| <b>Nullable</b>              | Non    | <b>True</b> (valeur par défaut) ou <b>false</b> selon que la colonne correspondante peut avoir une valeur null ou non.  |
| <b>DefaultValue</b>          | Non    | Valeur par défaut de la colonne correspondante.   |
| <b>MaxLength</b>             | Non    | Longueur maximale de la colonne correspondante.   |
| <b>Multiple</b>              | Non    | <b>True</b> ou <b>false</b> selon que la valeur de colonne correspondante sera stockée en tant que chaîne de longueur fixe.   |
| <b>Précision</b>             | Non    | Précision de la colonne correspondante.   |
| <b>Échelle</b>               | Non    | Échelle de la colonne correspondante.   |
| <b>Unicode</b>               | Non    | <b>True</b> ou <b>false</b> selon que la valeur de colonne correspondante sera stockée en tant que chaîne Unicode.  |
| <b>Classement</b>            | Non    | Chaîne qui spécifie l'ordre de tri à utiliser dans la source de données.  |
| <b>SRID</b>                  | Non    | Identificateur de référence système spatial. Valide uniquement pour les propriétés des types spatiaux. Pour plus d'informations, consultez <a href="#">SRID</a> et <a href="#">SRID (SQL Server)</a> .  |
| <b>StoreGeneratedPattern</b> | Non    | <b>None</b> , <b>Identity</b> (si la valeur de colonne correspondante est une identité générée dans la base de données) ou <b>calculée</b> (si la valeur de colonne correspondante est calculée dans la base de données). Non valide pour les propriétés RowType. |

#### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **Property**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

#### Exemple

L'exemple suivant montre un élément **EntityType** avec deux éléments de **propriété** enfants :

```

<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>

```

## Élément PropertyRef (SSDL)

L'élément **PropertyRef** en Store Schema Definition Language (SSDL) fait référence à une propriété définie sur un élément EntityType pour indiquer que la propriété effectuera l'un des rôles suivants :

- Faire partie de la clé primaire de la table représentée par l' **EntityType** . Un ou plusieurs éléments **PropertyRef** peuvent être utilisés pour définir une clé primaire. Pour plus d'informations, consultez Élément Key.
- Faire partie de la terminaison dépendante ou principale d'une contrainte référentielle. Pour plus d'informations, consultez Élément ReferentialConstraint.

L'élément **PropertyRef** ne peut avoir que les éléments enfants suivants :

- Documentation (zéro ou un)
- éléments d'annotation.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **PropertyRef** .

| NOM D'ATTRIBUT | REQUIS | VALEUR                          |
|----------------|--------|---------------------------------|
| <b>Nom</b>     | Oui    | Nom de la propriété référencée. |

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **PropertyRef** . Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage CSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **PropertyRef** utilisé pour définir une clé primaire en référençant une propriété définie sur un élément **EntityType** .

```

<EntityType Name="Customers">
  <Documentation>
    <Summary>Summary here.</Summary>
    <LongDescription>Long description here.</LongDescription>
  </Documentation>
  <Key>
    <PropertyRef Name="CustomerId" />
  </Key>
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>

```

## ReferentialConstraint, élément (SSDL)

L'élément **ReferentialConstraint** en Store Schema Definition Language (SSDL) représente une contrainte de clé étrangère (également appelée contrainte d'intégrité référentielle) dans la base de données sous-jacente. Les terminaisons principale et dépendante de la contrainte sont spécifiées respectivement par les éléments enfants Principal et Dependent. Les colonnes qui participent aux terminaisons principale et dépendante sont référencées avec les éléments PropertyRef.

L'élément **ReferentialConstraint** est un élément enfant facultatif de l'élément Association. Si un élément **ReferentialConstraint** n'est pas utilisé pour mapper la contrainte de clé étrangère spécifiée dans l'élément **Association**, un élément AssociationSetMapping doit être utilisé pour ce faire.

L'élément **ReferentialConstraint** peut avoir les éléments enfants suivants :

- Documentation (zéro ou un)
- Principal (exactement un élément) ;
- Dépendant (exactement un)
- éléments d'annotation (zéro, un ou plusieurs).

### Attributs applicables

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **ReferentialConstraint**. Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL. Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant montre un élément **Association** qui utilise un élément **ReferentialConstraint** pour spécifier les colonnes qui participent à la contrainte de clé étrangère **FK\_CustomerOrders** :

```

<Association Name="FK_CustomerOrders">
  <End Role="Customers"
    Type="ExampleModel.Store.Customers" Multiplicity="1">
    <OnDelete Action="Cascade" />
  </End>
  <End Role="Orders"
    Type="ExampleModel.Store.Orders" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Customers">
      <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
      <PropertyRef Name="CustomerId" />
    </Dependent>
  </ReferentialConstraint>
</Association>

```

## ReturnType, élément (SSDL)

L'élément **ReturnType** en Store Schema Definition Language (SSDL) spécifie le type de retour pour une fonction définie dans un élément **Function**. Un type de retour de fonction peut également être spécifié avec un attribut **ReturnType**.

Le type de retour d'une fonction est spécifié à l'aide de l'attribut **type** ou de l'élément **ReturnType**.

L'élément **ReturnType** peut avoir les éléments enfants suivants :

- CollectionType (un)

### NOTE

Un nombre quelconque d'attributs d'annotation (attributs XML personnalisés) peut être appliqué à l'élément **ReturnType**.

Toutefois, les attributs personnalisés ne peuvent pas appartenir à un espace de noms XML réservé pour le langage SSDL.

Les noms qualifiés complets de deux attributs personnalisés quelconques ne peuvent pas être identiques.

### Exemple

L'exemple suivant utilise une **fonction** qui retourne une collection de lignes.

```
<Function Name="GetProducts" IsComposable="true" Schema="dbo">
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="ProductID" Type="int" Nullable="false" />
        <Property Name="CategoryID" Type="bigint" Nullable="false" />
        <Property Name="ProductName" Type="nvarchar" MaxLength="40" Nullable="false" />
        <Property Name="UnitPrice" Type="money" />
        <Property Name="Discontinued" Type="bit" />
      </RowType>
    </CollectionType>
  </ReturnType>
</Function>
```

## RowType, élément (SSDL)

Un élément **RowType** en Store Schema Definition Language (SSDL) définit une structure sans nom comme type de retour pour une fonction définie dans le magasin.

Un élément **RowType** est l'élément enfant de l'élément **CollectionType** :

Un élément **RowType** peut avoir les éléments enfants suivants :

- Property (un ou plusieurs) ;

### Exemple

L'exemple suivant montre une fonction de magasin qui utilise un élément **CollectionType** pour spécifier que la fonction retourne une collection de lignes (comme spécifié dans l'élément **RowType**).

```

<Function Name="GetProducts" IsComposable="true" Schema="dbo">
  <ReturnType>
    <CollectionType>
      <RowType>
        <Property Name="ProductID" Type="int" Nullable="false" />
        <Property Name="CategoryID" Type="bigint" Nullable="false" />
        <Property Name="ProductName" Type="nvarchar" MaxLength="40" Nullable="false" />
        <Property Name="UnitPrice" Type="money" />
        <Property Name="Discontinued" Type="bit" />
      </RowType>
    </CollectionType>
  </ReturnType>
</Function>

```

## Schema, élément (SSDL)

L'élément **Schema** en Store Schema Definition Language (SSDL) est l'élément racine d'une définition de modèle de stockage. Il contient des définitions pour les objets, les fonctions et les conteneurs qui composent un modèle de stockage.

L'élément **Schema** peut contenir zéro, un ou plusieurs des éléments enfants suivants :

- Association
- EntityType
- EntityContainer
- Fonction

L'élément **Schema** utilise l'attribut **namespace** pour définir l'espace de noms du type d'entité et des objets Association dans un modèle de stockage. Dans un espace de noms, deux objets ne peuvent pas avoir le même nom.

Un espace de noms de modèle de stockage est différent de l'espace de noms XML de l'élément de **schéma**. Un espace de noms de modèle de stockage (tel que défini par l'attribut d' **espace de noms**) est un conteneur logique pour les types d'entités et les types d'association. L'espace de noms XML (indiqué par l'attribut **xmlns**) d'un élément de **schéma** est l'espace de noms par défaut pour les éléments enfants et les attributs de l'élément de **schéma**. Les espaces de noms XML de la forme <https://schemas.microsoft.com/ado/YYYY/MM/edm/ssdl> (où aaaa et MM représentent respectivement une année et un mois) sont réservés au langage SSDL. Des éléments et attributs personnalisés ne peuvent pas être dans des espaces de noms de cette forme.

### Attributs applicables

Le tableau ci-dessous décrit les attributs qui peuvent être appliqués à l'élément **Schema**.

| NOM D'ATTRIBUT | REQUIS | VALEUR |
|----------------|--------|--------|
|----------------|--------|--------|

| NOM D'ATTRIBUT               | REQUIS | VALEUR  |
|------------------------------|--------|---|
| <b>Namespace</b>             | Oui    | Espace de noms du modèle de stockage. La valeur de l'attribut d' <b>espace de noms</b> est utilisée pour former le nom qualifié complet d'un type. Par exemple, si un <b>EntityType</b> nommé <i>Customer</i> se trouve dans l'espace de noms ExampleModel. Store, le nom qualifié complet de l' <b>EntityType</b> est ExampleModel. Store. Customer.<br>Les chaînes suivantes ne peuvent pas être utilisées comme valeur pour l'attribut d' <b>espace de noms</b> : <b>System</b> , <b>transient</b> ou <b>EDM</b> . La valeur de l'attribut d' <b>espace de noms</b> ne peut pas être la même que la valeur de l'attribut d' <b>espace de noms</b> dans l'élément de schéma CSDL. |
| <b>Alias</b>                 | Non    | Identificateur utilisé à la place du nom de l'espace de noms. Par exemple, si un <b>EntityType</b> nommé <i>Customer</i> se trouve dans l'espace de noms ExampleModel. Store et que la valeur de l'attribut <b>alias</b> est <i>StorageModel</i> , vous pouvez utiliser <i>StorageModel. Customer</i> comme nom qualifié complet de l' <b>EntityType</b> .  |
| <b>Fournisseur</b>           | Oui    | Fournisseur de données.   |
| <b>ProviderManifestToken</b> | Oui    | Jeton qui indique au fournisseur quel manifeste de fournisseur retourner. Aucun format n'est défini pour le jeton. Les valeurs du jeton sont définies par le fournisseur. Pour plus d'informations sur les jetons de manifeste du fournisseur SQL Server, consultez SqlClient pour Entity Framework.  |

## Exemple

L'exemple suivant illustre un élément de **schéma** qui contient un élément **EntityContainer**, deux éléments **EntityType** et un élément **Association** .

```
<Schema Namespace="ExampleModel.Store"
        Alias="Self" Provider="System.Data.SqlClient"
        ProviderManifestToken="2008"
        xmlns="https://schemas.microsoft.com/ado/2009/11/edm/ssdl">
  <EntityContainer Name="ExampleModelStoreContainer">
    <EntityType Name="Customers">
      EntityType="ExampleModel.Store.Customers"
      Schema="dbo" />
    <EntityType Name="Orders">
      EntityType="ExampleModel.Store.Orders"
      Schema="dbo" />
    <AssociationSet Name="FK_CustomerOrders">
      Association="ExampleModel.Store.FK_CustomerOrders">
        <End Role="Customers" EntitySet="Customers" />
        <End Role="Orders" EntitySet="Orders" />
    </AssociationSet>
  </EntityContainer>
</Schema>
```

```

</EntityContainer>
<EntityType Name="Customers">
    <Documentation>
        <Summary>Summary here.</Summary>
        <LongDescription>Long description here.</LongDescription>
    </Documentation>
    <Key>
        <PropertyRef Name="CustomerId" />
    </Key>
    <Property Name="CustomerId" Type="int" Nullable="false" />
    <Property Name="Name" Type="nvarchar(max)" Nullable="false" />
</EntityType>
<EntityType Name="Orders" xmlns:c="http://CustomNamespace">
    <Key>
        <PropertyRef Name="OrderId" />
    </Key>
    <Property Name="OrderId" Type="int" Nullable="false"
        c:CustomAttribute="someValue"/>
    <Property Name="ProductId" Type="int" Nullable="false" />
    <Property Name="Quantity" Type="int" Nullable="false" />
    <Property Name="CustomerId" Type="int" Nullable="false" />
    <c:CustomElement>
        Custom data here.
    </c:CustomElement>
</EntityType>
<Association Name="FK_CustomerOrders">
    <End Role="Customers"
        Type="ExampleModel.Store.Customers" Multiplicity="1">
        <OnDelete Action="Cascade" />
    </End>
    <End Role="Orders"
        Type="ExampleModel.Store.Orders" Multiplicity="*" />
<ReferentialConstraint>
    <Principal Role="Customers">
        <PropertyRef Name="CustomerId" />
    </Principal>
    <Dependent Role="Orders">
        <PropertyRef Name="CustomerId" />
    </Dependent>
</ReferentialConstraint>
</Association>
<Function Name="UpdateOrderQuantity">
    Aggregate="false"
    BuiltIn="false"
    NiladicFunction="false"
    IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion"
    Schema="dbo">
    <Parameter Name="orderId" Type="int" Mode="In" />
    <Parameter Name="newQuantity" Type="int" Mode="In" />
</Function>
<Function Name="UpdateProductInOrder" IsComposable="false">
    <CommandText>
        UPDATE Orders
        SET ProductId = @productId
        WHERE OrderId = @orderId;
    </CommandText>
    <Parameter Name="productId"
        Mode="In"
        Type="int"/>
    <Parameter Name="orderId"
        Mode="In"
        Type="int"/>
</Function>
</Schema>

```

## Attributs d'annotation

Les attributs d'annotation en SSDL (Store Schema Definition Language) sont des attributs XML personnalisés dans le modèle de stockage qui fournissent des métadonnées supplémentaires concernant les éléments dans le modèle de stockage. En plus d'avoir une structure XML valide, les contraintes suivantes s'appliquent aux attributs d'annotation :

- Les attributs d'annotation ne doivent pas figurer dans un espace de noms XML réservé pour le langage SSDL.
- Les noms qualifiés complets de deux attributs d'annotation ne doivent pas être identiques.

Plusieurs attributs d'annotation peuvent être appliqués à un élément SSDL donné. Vous pouvez accéder aux métadonnées contenues dans les éléments d'annotation au moment de l'exécution à l'aide des classes de l'espace de noms System. Data. Metadata. Edm.

### Exemple

L'exemple suivant montre un élément EntityType qui a un attribut d'annotation appliqué à la propriété **OrderId**. L'exemple montre également un élément d'annotation ajouté à l'élément **EntityType**.

```
<EntityType Name="Orders" xmlns:c="http://CustomNamespace">
  <Key>
    <PropertyRef Name="OrderId" />
  </Key>
  <Property Name="OrderId" Type="int" Nullable="false"
    c:CustomAttribute="someValue"/>
  <Property Name="ProductId" Type="int" Nullable="false" />
  <Property Name="Quantity" Type="int" Nullable="false" />
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <c:CustomElement>
    Custom data here.
  </c:CustomElement>
</EntityType>
```

## Éléments d'annotation (SSDL)

Les éléments d'annotation en SSDL (Store Schema Definition Language) sont des éléments XML personnalisés dans le modèle de stockage qui fournissent des métadonnées supplémentaires concernant le modèle de stockage. En plus d'avoir une structure XML valide, les contraintes suivantes s'appliquent aux éléments d'annotation :

- Les éléments d'annotation ne doivent pas figurer dans un espace de noms XML réservé au langage SSDL.
- Les noms qualifiés complets de deux éléments d'annotation ne doivent pas être identiques.
- Les éléments d'annotation doivent apparaître après tous les autres éléments enfants d'un élément SSDL donné.

Plusieurs éléments d'annotation peuvent être des enfants d'un élément SSDL donné. À partir de la .NET Framework version 4, les métadonnées contenues dans les éléments d'annotation sont accessibles au moment de l'exécution à l'aide des classes de l'espace de noms System. Data. Metadata. Edm.

### Exemple

L'exemple suivant montre un élément EntityType qui a un élément annotation (**customelement**). L'exemple montre également un attribut d'annotation appliqué à la propriété **OrderId**.

```

<EntityType Name="Orders" xmlns:c="http://CustomNamespace">
  <Key>
    <PropertyRef Name="OrderId" />
  </Key>
  <Property Name="OrderId" Type="int" Nullable="false"
    c:CustomAttribute="someValue"/>
  <Property Name="ProductId" Type="int" Nullable="false" />
  <Property Name="Quantity" Type="int" Nullable="false" />
  <Property Name="CustomerId" Type="int" Nullable="false" />
  <c:CustomElement>
    Custom data here.
  </c:CustomElement>
</EntityType>

```

## Facettes (SSDL)

Les facettes en SSDL (Store Schema Definition Language) représentent des contraintes sur les types de colonne spécifiés dans les éléments Property. Les facettes sont implémentées en tant qu'attributs XML sur les éléments de **propriété**.

Le tableau ci-dessous décrit les facettes prises en charge dans le langage SSDL :

| FACETTE           | DESCRIPTION  |
|-------------------|--|
| <b>Classement</b> | Spécifie la table de classement ou ordre de tri à utiliser lors de l'exécution d'opérations de comparaison et de tri sur des valeurs de la propriété.  |
| <b>Multiple</b>   | Spécifie si la longueur de la valeur de colonne peut varier.   |
| <b>MaxLength</b>  | Spécifie la longueur maximale de la valeur de colonne.   |
| <b>Précision</b>  | Pour les propriétés de type <b>Decimal</b> , spécifie le nombre de chiffres qu'une valeur de propriété peut avoir. Pour les propriétés de type <b>Time</b> , <b>Date Time</b> et <b>Date Time Offset</b> , spécifie le nombre de chiffres pour la partie fractionnaire des secondes de la valeur de colonne. |
| <b>Échelle</b>    | Spécifie le nombre de chiffres à droite de la virgule décimale pour la valeur de colonne.  |
| <b>Unicode</b>    | Indique si la valeur de colonne est stockée au format Unicode.   |

# Définition de requête - Entity Framework Designer

13/09/2018 • 10 minutes to read

Cette procédure pas à pas montre comment ajouter une définition de type de requête et une entité correspondante à un modèle à l'aide du Concepteur EF. Une requête de définition est couramment utilisée pour fournir des fonctionnalités semblables à celles fournies par une vue de base de données, mais la vue est définie dans le modèle, pas la base de données. Une requête de définition vous permet d'exécuter une instruction SQL qui est spécifiée dans le **DefiningQuery** élément d'un fichier .edmx. Pour plus d'informations, consultez [DefiningQuery](#) dans le [spécification SSDL](#).

Lorsque vous utilisez des requêtes de définition, vous devez également définir un type d'entité dans votre modèle. Le type d'entité est utilisé pour exposer les données exposées par la requête de définition. Notez que les données révélées par ce type d'entité sont en lecture seule.

Les requêtes paramétrées ne peuvent pas être exécutées en tant que requêtes de définition. Toutefois, les données peuvent être mises à jour en mappant les fonctions d'insertion, de mise à jour et de suppression du type d'entité qui surface les données aux procédures stockées. Pour plus d'informations, consultez [Insert, Update et Delete avec des procédures stockées](#).

Cette rubrique montre comment effectuer les tâches suivantes.

- Ajouter une requête de définition
- Ajouter un Type d'entité au modèle
- Mapper la requête de définition pour le Type d'entité

## Prérequis

Pour exécuter cette procédure pas à pas, vous avez besoin des éléments suivants :

- Une version récente de Visual Studio.
- Le [base de données School exemple](#).

## Configurer le projet

Cette procédure pas à pas utilise Visual Studio 2012 ou version ultérieure.

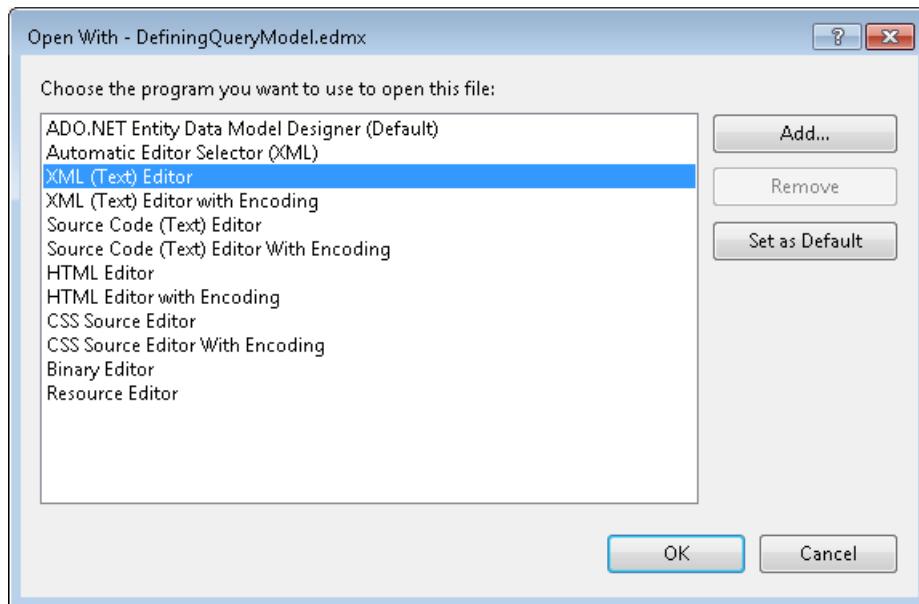
- Ouvrez Visual Studio.
- Dans le menu **Fichier**, pointez sur **Nouveau**, puis cliquez sur **Projet**.
- Dans le volet gauche, cliquez sur **Visual C#**, puis sélectionnez le **Application Console** modèle.
- Entrez **DefiningQuerySample** en tant que le nom du projet et cliquez sur **OK**.

## Créer un modèle basé sur la base de données School

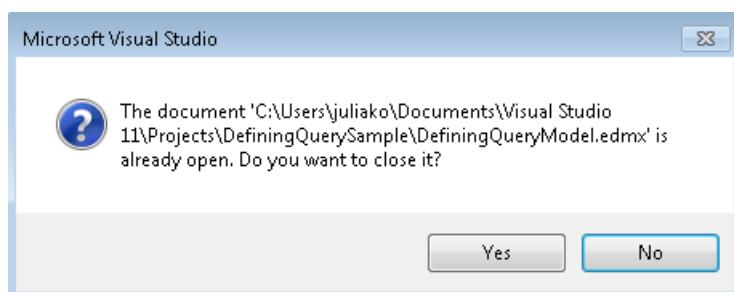
- Cliquez sur le nom de projet dans l'Explorateur de solutions, pointez sur **ajouter**, puis cliquez sur **un nouvel élément**.
- Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet Modèles.
- Entrez **DefiningQueryModel.edmx** pour le nom de fichier, puis cliquez sur **ajouter**.
- Dans la boîte de dialogue Choisir le contenu du modèle, sélectionnez **générer à partir de la base de**

**données**, puis cliquez sur **suivant**.

- Cliquez sur Nouvelle connexion. Dans la boîte de dialogue Propriétés de connexion, entrez le nom du serveur (par exemple, **(localdb)\mssqllocaldb**), sélectionnez la méthode d'authentification, tapez **School** pour le nom de la base de données, puis Cliquez sur **OK**. La boîte de dialogue Choisir votre connexion de données est mis à jour avec le paramètre de votre connexion de base de données.
- Dans la boîte de dialogue Choisir vos objets de base de données, vérifiez le **Tables** noeud. Cette opération ajoute toutes les tables à la **School** modèle.
- Cliquez sur **Terminer**.
- Dans l'Explorateur de solutions, cliquez sur le **DefiningQueryModel.edmx** fichier et sélectionnez **ouvrir avec...**.
- Sélectionnez **éditeur XML (texte)**.



- Cliquez sur **Oui** si vous y êtes invité avec le message suivant :



## Ajouter une requête de définition

Dans cette étape, nous allons utiliser l'éditeur XML pour ajouter une définition de requête et un type d'entité à la section SSDL du fichier .edmx.

- Ajouter un **EntitySet** élément à la section SSDL du fichier .edmx (ligne 5 à 13). Spécifier les éléments suivants :
  - Uniquement les **nom** et **EntityType** les attributs de la **EntitySet** élément sont spécifiées.
  - Le nom qualifié complet du type d'entité est utilisé dans le **EntityType** attribut.
  - L'instruction SQL à exécuter est spécifiée dans le **DefiningQuery** élément.

```

<!-- SSDL content -->
<edmx:StorageModels>
    <Schema Namespace="SchoolModel.Store" Alias="Self" Provider="System.Data.SqlClient"
ProviderManifestToken="2008"
xmlns:store="http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm/ssdl">
        <EntityContainer Name="SchoolModelStoreContainer">
            <EntitySet Name="GradeReport" EntityType="SchoolModel.Store.GradeReport">
                <DefiningQuery>
                    SELECT CourseID, Grade, FirstName, LastName
                    FROM StudentGrade
                    JOIN
                    (SELECT * FROM Person WHERE EnrollmentDate IS NOT NULL) AS p
                    ON StudentID = p.PersonID
                </DefiningQuery>
            </EntitySet>
            <EntitySet Name="Course" EntityType="SchoolModel.Store.Course" store:Type="Tables" Schema="dbo" />

```

- Ajouter le **EntityType** élément à la section SSDL du fichier.fichier comme indiqué ci-dessous. Notez les points suivants :
  - La valeur de la **nom** attribut correspond à la valeur de la **EntityType** d'attribut dans le **EntitySet** élément ci-dessus, bien que le nom qualifié complet de la type d'entité est utilisé dans le **EntityType** attribut.
  - Les noms des propriétés correspondent aux noms de colonnes retournés par l'instruction SQL dans le **DefiningQuery** élément (ci-dessus).
  - Dans cet exemple, la clé d'entité est composée de trois propriétés pour garantir le caractère unique de la valeur de la clé.

```

<EntityType Name="GradeReport">
    <Key>
        <PropertyRef Name="CourseID" />
        <PropertyRef Name="FirstName" />
        <PropertyRef Name="LastName" />
    </Key>
    <Property Name="CourseID"
        Type="int"
        Nullable="false" />
    <Property Name="Grade"
        Type="decimal"
        Precision="3"
        Scale="2" />
    <Property Name="FirstName"
        Type="nvarchar"
        Nullable="false"
        MaxLength="50" />
    <Property Name="LastName"
        Type="nvarchar"
        Nullable="false"
        MaxLength="50" />
</EntityType>

```

#### NOTE

Si plus tard vous exécutez le **Assistant modèle de mise à jour** boîte de dialogue, toutes les modifications apportées au modèle de stockage, y compris les requêtes de définition, sera remplacé.

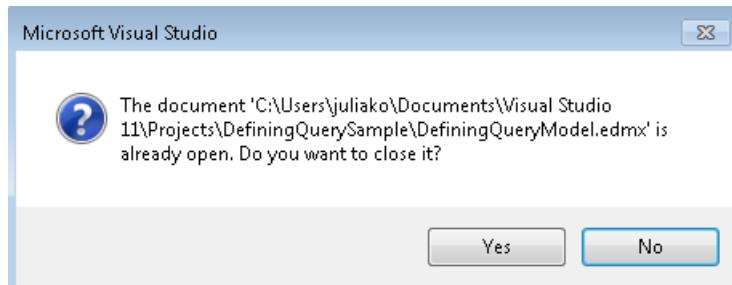
## Ajouter un Type d'entité au modèle

Dans cette étape, nous allons ajouter le type d'entité au modèle conceptuel à l'aide du Concepteur EF. Notez les points suivants :

- Le **nom** de l'entité correspond à la valeur de la **EntityType** d'attribut dans le **EntitySet** élément ci-dessus.
- Les noms des propriétés correspondent aux noms de colonnes retournés par l'instruction SQL dans le **DefiningQuery** élément ci-dessus.
- Dans cet exemple, la clé d'entité est composée de trois propriétés pour garantir le caractère unique de la valeur de la clé.

Ouvrez le modèle dans le Concepteur EF.

- Double-cliquez sur le DefiningQueryModel.edmx.
- Par exemple **Oui** au message suivant :



Le Concepteur d'entités, qui fournit une aire de conception pour la modification de votre modèle, s'affiche.

- Cliquez sur l'aire du concepteur puis sélectionnez **Ajouter nouveau->entité...**.
- Spécifiez **GradeReport** pour le nom de l'entité et **CourseID** pour le **propriété Key**.
- Avec le bouton droit le **GradeReport** entité, puis sélectionnez **Ajouter nouveau - > propriété scalaire**.
- Modifier le nom par défaut de la propriété à **FirstName**.
- Ajoutez une autre propriété scalaire et spécifiez **LastName** pour le nom.
- Ajoutez une autre propriété scalaire et spécifiez **Grade** pour le nom.
- Dans le **propriétés** fenêtre, modifier le **Grade de Type** propriété **décimal**.
- Sélectionnez le **FirstName** et **LastName** propriétés.
- Dans le **propriétés** fenêtre, de modifier le **EntityKey** valeur de propriété à **True**.

Par conséquent, les éléments suivants ont été ajoutés à la **CSDL** section du fichier .edmx.

```
<EntitySet Name="GradeReport" EntityType="SchoolModel.GradeReport" />

<EntityType Name="GradeReport">
  ...
</EntityType>
```

## Mapper la requête de définition pour le Type d'entité

Dans cette étape, nous allons utiliser la fenêtre Détails de mappage pour mapper les concepts et les types d'entités de stockage.

- Cliquez sur le **GradeReport** entité sur l'aire de conception et sélectionnez **mappage de Table**.  
Le **détails de Mapping** fenêtre s'affiche.
- Sélectionnez **GradeReport** à partir de la **<ajouter une Table ou vue>** liste déroulante (situé sous **Tables**).  
Par défaut des mappages entre les concepts et de stockage **GradeReport** type d'entité s'affichent.

| Parameter / Column               | Operator | Property                  | Use Original...                     | Rows Affected Parameter |
|----------------------------------|----------|---------------------------|-------------------------------------|-------------------------|
| <b>Functions</b>                 |          |                           |                                     |                         |
| <b>Insert Using InsertPerson</b> |          |                           |                                     |                         |
| <b>Parameters</b>                |          |                           |                                     |                         |
| @ LastName : nvarchar            | ←        | >LastName : String        | <input type="checkbox"/>            |                         |
| @ FirstName : nvarchar           | ←        | FirstName : String        | <input type="checkbox"/>            |                         |
| @ HireDate : datetime            | ←        | HireDate : DateTime       | <input type="checkbox"/>            |                         |
| @ EnrollmentDate : datetime      | ←        | EnrollmentDate : DateTime | <input type="checkbox"/>            |                         |
| @ Discriminator : nvarchar       | ←        | Discriminator : String    | <input type="checkbox"/>            |                         |
| <b>Result Column Bindings</b>    |          |                           |                                     |                         |
| NewPersonID                      | →        | PersonID : Int32          | <input checked="" type="checkbox"/> |                         |

Par conséquent, le **EntitySetMapping** élément est ajouté à la section mapping du fichier .edmx.

```
<EntitySetMapping Name="GradeReports">
  <EntityTypeMapping TypeName="IsTypeOf(SchoolModel.GradeReport)">
    <MappingFragment StoreEntitySet="GradeReport">
      <ScalarProperty Name="LastName" ColumnName="LastName" />
      <ScalarProperty Name="FirstName" ColumnName="FirstName" />
      <ScalarProperty Name="Grade" ColumnName="Grade" />
      <ScalarProperty Name="CourseID" ColumnName="CourseID" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

- Compilez l'application.

## Appeler la requête de définition dans votre Code

Vous pouvez maintenant exécuter la requête de définition à l'aide de la **GradeReport** type d'entité.

```
using (var context = new SchoolEntities())
{
  var report = context.GradeReports.FirstOrDefault();
  Console.WriteLine("{0} {1} got {2}",
    report.FirstName, report.LastName, report.Grade);
}
```

# Procédures stockées avec plusieurs jeux de résultats

13/09/2018 • 10 minutes to read

Parfois, lorsque vous utilisez stockées procédures, vous devez retourner plusieurs résultats définissent. Ce scénario est couramment utilisé pour réduire le nombre de base de données allers-retours requis pour composer un seul écran. Avant d'EF5, Entity Framework permettrait la procédure stockée à appeler, mais ne renvoie que le premier jeu de résultats au code appelant.

Cet article vous montrera deux méthodes que vous pouvez utiliser pour accéder à plus d'un jeu de résultats à partir d'une procédure stockée dans Entity Framework. Une qui utilise seulement le code et fonctionne avec les deux du Code tout d'abord et le Concepteur EF et une qui ne fonctionne qu'avec le Concepteur EF. Les outils et la prise en charge de l'API pour cela doivent améliorer dans les futures versions d'Entity Framework.

## Modèle

Les exemples de cet article utilisent un Blog de base et modèle de billets où un blog a un billet et nombreuses publications appartient à un blog unique. Nous allons utiliser une procédure stockée dans la base de données qui retourne tous les blogs et des publications, quelque chose comme suit :

```
CREATE PROCEDURE [dbo].[GetAllBlogsAndPosts]
AS
    SELECT * FROM dbo.Blogs
    SELECT * FROM dbo.Posts
```

## L'accès aux résultats multiples définit avec le Code

Nous pouvons exécuter du code utilisé pour émettre une commande SQL brutes pour exécuter notre procédure stockée. L'avantage de cette approche est qu'elle fonctionne avec les deux Code tout d'abord et le Concepteur EF.

Afin d'obtenir des résultats multiples définit le travail que nous devons supprimer à l'API ObjectContext à l'aide de l'interface IObjectContextAdapter.

Une fois que nous avons un ObjectContext nous pouvons utiliser la méthode Translate pour traduire les résultats de notre procédure stockée dans les entités qui peuvent être suivies et utilisées dans EF comme d'habitude.

L'exemple de code suivant illustre cela en action.

```

using (var db = new BloggingContext())
{
    // If using Code First we need to make sure the model is built before we open the connection
    // This isn't required for models created with the EF Designer
    db.Database.Initialize(force: false);

    // Create a SQL command to execute the sproc
    var cmd = db.Database.Connection.CreateCommand();
    cmd.CommandText = "[dbo].[GetAllBlogsAndPosts]";

    try
    {

        db.Database.Connection.Open();
        // Run the sproc
        var reader = cmd.ExecuteReader();

        // Read Blogs from the first result set
        var blogs = ((IObjectContextAdapter)db)
            .ObjectContext
            .Translate<Blog>(reader, "Blogs", MergeOption.AppendOnly);

        foreach (var item in blogs)
        {
            Console.WriteLine(item.Name);
        }

        // Move to second result set and read Posts
        reader.NextResult();
        var posts = ((IObjectContextAdapter)db)
            .ObjectContext
            .Translate<Post>(reader, "Posts", MergeOption.AppendOnly);

        foreach (var item in posts)
        {
            Console.WriteLine(item.Title);
        }
    }
    finally
    {
        db.Database.Connection.Close();
    }
}

```

La méthode Translate accepte le lecteur que nous avons reçu lorsque nous avons exécuté la procédure, un nom de l'EntitySet et une MergeOption. Le nom EntitySet sera identique à la propriété DbSet sur votre contexte dérivé. L'énumération MergeOption contrôle la gestion des résultats si la même entité existe déjà dans la mémoire.

Ici, nous itérer dans la collection de blogs avant l'appel NextResult, il est important étant donné le code ci-dessus, car le premier jeu de résultats doit être utilisé avant de passer au jeu de résultats suivant.

Une fois traduire les deux méthodes sont appelées les entités de Blog et Post sont suivies par Entity Framework de la même façon que toute autre entité puis donc peut être modifié ou supprimé et enregistré comme d'habitude.

#### **NOTE**

Entity Framework ne prend pas en compte un mappage de lorsqu'il crée des entités à l'aide de la méthode Translate. Elle correspondra simplement des noms de colonnes dans le jeu de résultats avec des noms de propriété de vos classes.

#### NOTE

Que si vous avez activé, le chargement différé en accédant à la propriété de billets sur l'une des entités blog puis EF se connectera à la base de données charger en différé toutes ses publications, même si nous avons déjà chargé tous les. Il s'agit, car Entity Framework ne peut pas savoir si vous avez chargé tous les billets ou s'il en existe plus dans la base de données. Si vous souhaitez éviter ce problème vous devrez désactiver le chargement différé.

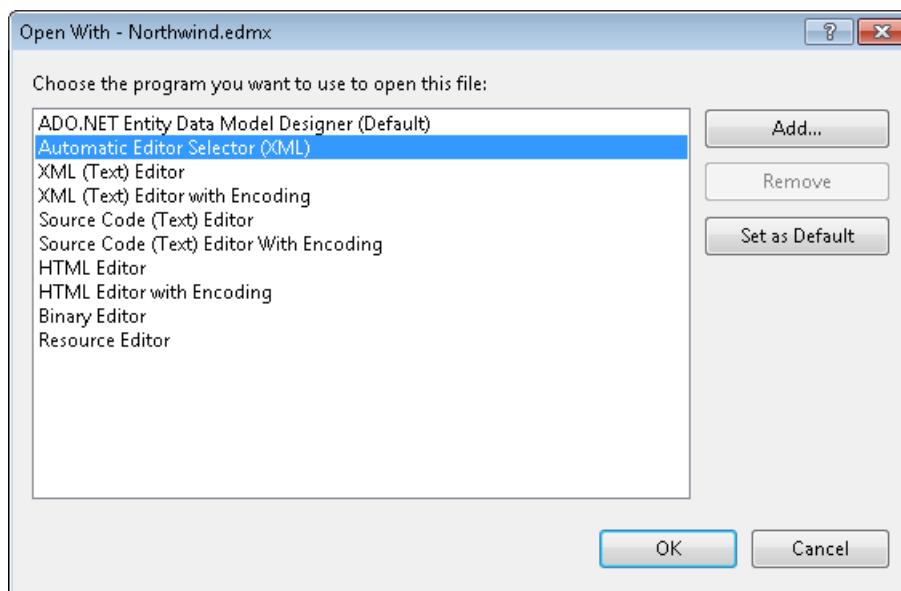
## Plusieurs jeux de résultats avec EDMX configuré dans

#### NOTE

Vous devez cibler .NET Framework 4.5 pour être en mesure de configurer plusieurs jeux de résultats dans EDMX. Si vous ciblez le .NET 4.0, vous pouvez utiliser la méthode basée sur le code indiquée dans la section précédente.

Si vous utilisez le Concepteur EF, vous pouvez également modifier votre modèle afin qu'il sache sur les jeux de résultats qui seront renvoyées. Une chose à savoir avant de la main est que les outils ne sont pas plusieurs résultats défini prenant en charge, vous devez modifier manuellement le fichier edmx. Modification du fichier edmx que cela fonctionne, mais elle entraîne également l'arrêt la validation du modèle dans Visual Studio. Par conséquent, si vous validez votre modèle vous obtiendrez toujours des erreurs.

- Pour ce faire, vous devez ajouter la procédure stockée à votre modèle comme vous le feriez pour une requête de jeu de résultats unique.
- Une fois que vous avez obtenu cela, vous devez cliquez avec le bouton droit sur votre modèle et sélectionnez **ouvrir avec...** puis **Xml**



Une fois que le modèle ouvert en tant que XML, vous devez procédez comme suit :

- Rechercher l'importation de fonction et de type complexe dans votre modèle :

```

<!-- CSDL content -->
<edmx:ConceptualModels>

...

<FunctionImport Name="GetAllBlogsAndPosts" ReturnType="Collection(BlogModel.GetAllBlogsAndPosts_Result)"
/>

...

<ComplexType Name="GetAllBlogsAndPosts_Result">
    <Property Type="Int32" Name="BlogId" Nullable="false" />
    <Property Type="String" Name="Name" Nullable="false" MaxLength="255" />
    <Property Type="String" Name="Description" Nullable="true" />
</ComplexType>

...

</edmx:ConceptualModels>

```

- Supprimer le type complexe
- Mettre à jour de l'importation de fonction afin qu'elle correspond à vos entités, dans notre cas, qu'il doit ressembler à ce qui suit :

```

<FunctionImport Name="GetAllBlogsAndPosts">
    <ReturnType EntitySet="Blogs" Type="Collection(BlogModel.Blog)" />
    <ReturnType EntitySet="Posts" Type="Collection(BlogModel.Post)" />
</FunctionImport>

```

Cela indique le modèle que la procédure stockée retourne deux collections, une des entrées de blog et celui de la validation d'entrées.

- Recherchez l'élément de mappage de fonction :

```

<!-- C-S mapping content -->
<edmx:Mappings>

...

<FunctionImportMapping FunctionImportName="GetAllBlogsAndPosts"
FunctionName="BlogModel.Store.GetAllBlogsAndPosts">
    <ResultMapping>
        <ComplexTypeMapping TypeName="BlogModel.GetAllBlogsAndPosts_Result">
            <ScalarProperty Name="BlogId" ColumnName="BlogId" />
            <ScalarProperty Name="Name" ColumnName="Name" />
            <ScalarProperty Name="Description" ColumnName="Description" />
        </ComplexTypeMapping>
    </ResultMapping>
</FunctionImportMapping>

...

</edmx:Mappings>

```

- Remplacez le mappage de résultat avec l'un pour chaque entité renvoyée, tels que les éléments suivants :

```

<ResultMapping>
  <EntityTypeMapping TypeName = "BlogModel.Blog">
    <ScalarProperty Name="BlogId" ColumnName="BlogId" />
    <ScalarProperty Name="Name" ColumnName="Name" />
    <ScalarProperty Name="Description" ColumnName="Description" />
  </EntityTypeMapping>
</ResultMapping>
<ResultMapping>
  <EntityTypeMapping TypeName="BlogModel.Post">
    <ScalarProperty Name="BlogId" ColumnName="BlogId" />
    <ScalarProperty Name="PostId" ColumnName="PostId"/>
    <ScalarProperty Name="Title" ColumnName="Title" />
    <ScalarProperty Name="Text" ColumnName="Text" />
  </EntityTypeMapping>
</ResultMapping>

```

Il est également possible de mapper les jeux de résultats à des types complexes, tel que celui créé par défaut. Pour ce faire, vous créez un nouveau type complexe, au lieu de les supprimer et utilisez les types complexes partout que que vous aviez utilisé les noms d'entité dans les exemples ci-dessus.

Une fois que ces mappages ont été modifiées, vous pouvez enregistrer le modèle et exécutez le code suivant pour utiliser la procédure stockée :

```

using (var db = new BlogEntities())
{
  var results = db.GetAllBlogsAndPosts();

  foreach (var result in results)
  {
    Console.WriteLine("Blog: " + result.Name);
  }

  var posts = results.GetNextResult<Post>();

  foreach (var result in posts)
  {
    Console.WriteLine("Post: " + result.Title);
  }

  Console.ReadLine();
}

```

#### NOTE

Si vous modifiez manuellement le fichier edmx pour votre modèle, il sera remplacé si vous régénérez jamais le modèle à partir de la base de données.

## Récapitulatif

Ici, nous avons décrit deux méthodes d'accès aux résultats multiples définit à l'aide d'Entity Framework. Les deux d'entre eux sont valides en fonction de votre situation et préférences et vous devez choisir celui qui semble mieux à votre situation. Il est prévu que la prise en charge pour résultat plusieurs jeux sera améliorée dans les futures versions d'Entity Framework et que les étapes dans ce document ne sera plus nécessaire.

# Fonctions table (TVF)

23/11/2019 • 7 minutes to read

## NOTE

**EF5 uniquement** : les fonctionnalités, les API, etc. présentées dans cette page ont été introduites dans Entity Framework 5. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

La vidéo et la procédure pas à pas montrent comment mapper des fonctions table (TVF) à l'aide de l'Entity Framework Designer. Il montre également comment appeler une fonction TVF à partir d'une requête LINQ.

Les TVF sont actuellement pris en charge uniquement dans le flux de travail Database First.

La prise en charge de TVF a été introduite dans Entity Framework version 5. Notez que pour utiliser les nouvelles fonctionnalités telles que les fonctions table, les énumérations et les types spatiaux, vous devez cibler .NET Framework 4,5. Visual Studio 2012 cible .NET 4,5 par défaut.

Les TVF sont très similaires aux procédures stockées avec une différence clé : le résultat d'une fonction TVF est composable. Cela signifie que les résultats d'une fonction TVF peuvent être utilisés dans une requête LINQ alors que les résultats d'une procédure stockée ne le peuvent pas.

## Regarder la vidéo

**Présenté par:** Julia Kornich

[Wmv](#) | [MP4](#) | [WMV \(zip\)](#)

## Conditions préalables

Pour effectuer cette procédure pas à pas, vous devez :

- Installez la [base de données School](#).
- Disposer d'une version récente de Visual Studio

## Configurer le projet

1. Ouvrez Visual Studio
2. Dans le menu **fichier**, pointez sur **nouveau**, puis cliquez sur **projet** .
3. Dans le volet gauche, cliquez sur **Visual C#** , puis sélectionnez le modèle **console** .
4. Entrez **TVF** comme nom du projet, puis cliquez sur **OK** .

## Ajouter une TVF à la base de données

- Sélectionnez **affichage-> Explorateur d'objets SQL Server**
- Si la base de données locale ne figure pas dans la liste des serveurs : cliquez avec le bouton droit sur **SQL Server** et sélectionnez **ajouter SQL Server** utiliser l'**authentification Windows** par défaut pour la connexion au serveur de base de données locale
- Développez le nœud de base de données locale
- Sous le nœud bases de données, cliquez avec le bouton droit sur le nœud de la base de données School, puis sélectionnez **nouvelle requête...**

- Dans l'éditeur T-SQL, collez la définition de TVF suivante.

```
CREATE FUNCTION [dbo].[GetStudentGradesForCourse]
(@CourseID INT)
RETURNS TABLE
RETURN
SELECT [EnrollmentID],
[CourseID],
[StudentID],
[Grade]
FROM [dbo].[StudentGrade]
WHERE CourseID = @CourseID
```

- Cliquez sur le bouton droit de la souris dans l'éditeur T-SQL, puis sélectionnez **exécuter**.
- La fonction GetStudentGradesForCourse est ajoutée à la base de données School.

## Créer un modèle

1. Cliquez avec le bouton droit sur le nom du projet dans Explorateur de solutions, pointez sur **Ajouter**, puis cliquez sur **nouvel élément**.
2. Sélectionnez **données** dans le menu de gauche, puis sélectionnez **ADO.NET Entity Data Model** dans le volet **modèles**.
3. Entrez **TVFModel.edmx** comme nom de fichier, puis cliquez sur **Ajouter**.
4. Dans la boîte de dialogue choisir le contenu du Model, sélectionnez **générer à partir de la base de données**, puis cliquez sur **suivant**.
5. Cliquez sur **nouvelle connexion** entrée (base de données locale) \mssqllocaldb dans la zone de texte Nom du serveur, entrez **School** pour le nom de la base de données, cliquez sur **OK**.
6. Dans la boîte de dialogue choisir vos objets de base de données, sous le nœud **Tables**, sélectionnez les tables **Person, StudentGradeet course**
7. Sélectionnez la fonction **GetStudentGradesForCourse** située sous les **procédures stockées et les fonctions** nœud note, qui à partir de Visual Studio 2012, la Entity designer vous permet d'importer par lot vos procédures stockées et fonctions.
8. Cliquez sur **Terminer**
9. Le Entity Designer, qui fournit une aire de conception pour la modification de votre modèle, est affiché. Tous les objets que vous avez sélectionnés dans la boîte **de dialogue choisir vos objets de base de données** sont ajoutés au modèle.
10. Par défaut, la forme de résultat de chaque fonction ou procédure stockée importée devient automatiquement un nouveau type complexe dans votre modèle d'entité. Mais nous voulons mapper les résultats de la fonction GetStudentGradesForCourse à l'entité StudentGrade : cliquez avec le bouton droit sur l'aire de conception, sélectionnez **Explorateur de modèles** dans l'Explorateur de modèles, sélectionnez **importations de fonction**, puis double-cliquez sur la fonction **GetStudentGradesForCourse** dans la boîte de dialogue Modifier l'importation de fonction, sélectionnez **entités** et choisissez **StudentGrade**

## Conserver et récupérer des données

Ouvrez le fichier dans lequel la méthode main est définie. Ajoutez le code suivant à la fonction main.

Le code suivant montre comment générer une requête qui utilise une fonction table. La requête projette les résultats dans un type anonyme qui contient le titre du cours associé et les étudiants associés avec un niveau

supérieur ou égal à 3,5.

```
using (var context = new SchoolEntities())
{
    var CourseID = 4022;
    var Grade = 3.5M;

    // Return all the best students in the Microeconomics class.
    var students = from s in context.GetStudentGradesForCourse(CourseID)
                   where s.Grade >= Grade
                   select new
                   {
                       s.Person,
                       s.Course.Title
                   };

    foreach (var result in students)
    {
        Console.WriteLine(
            "Couse: {0}, Student: {1} {2}",
            result.Title,
            result.Person.FirstName,
            result.Person.LastName);
    }
}
```

Compilez et exéutez l'application. Le programme génère la sortie suivante :

```
Couse: Microeconomics, Student: Arturo Anand
Couse: Microeconomics, Student: Carson Bryant
```

## Résumé

Dans cette procédure pas à pas, nous avons vu comment mapper des fonctions table (TVF) à l'aide de l'Entity Framework Designer. Elle a également démontré comment appeler une TVF à partir d'une requête LINQ.

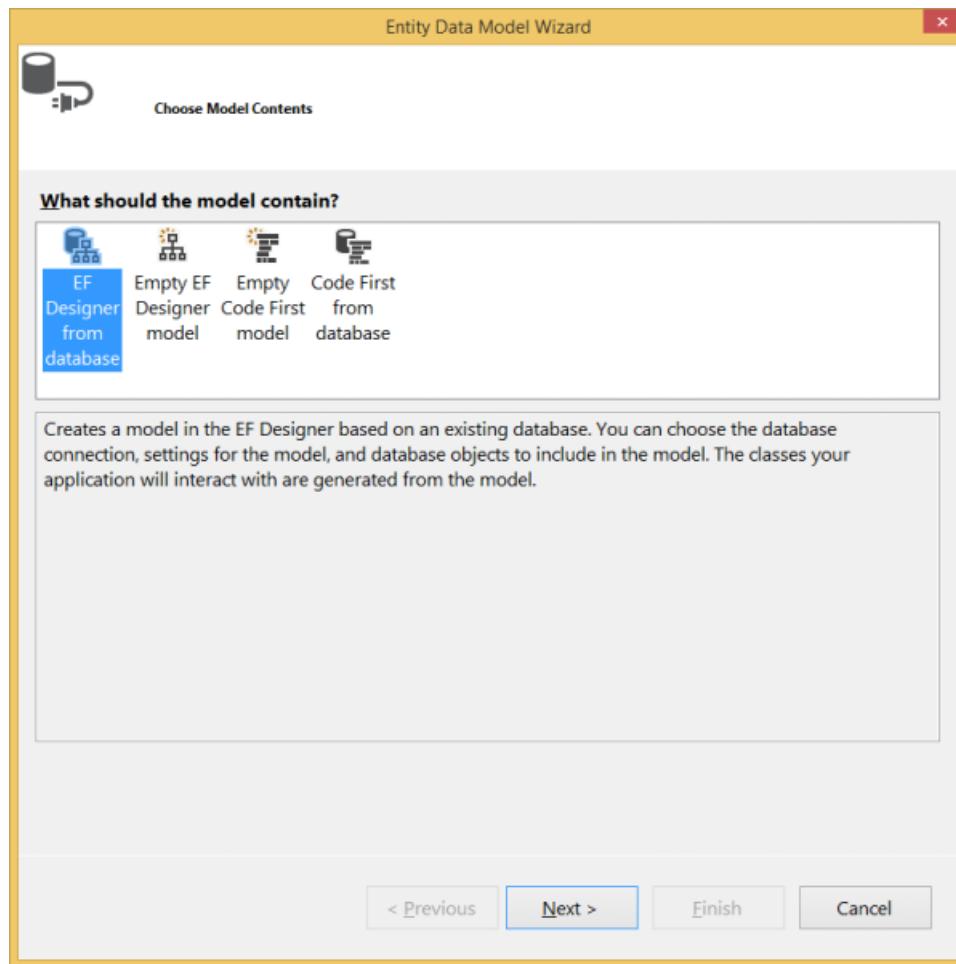
# Raccourcis de clavier du concepteur Entity Framework

13/09/2018 • 14 minutes to read

Cette page fournit une liste de raccourcis clavier qui sont disponibles dans les différents écrans de l'Entity Framework Tools pour Visual Studio.

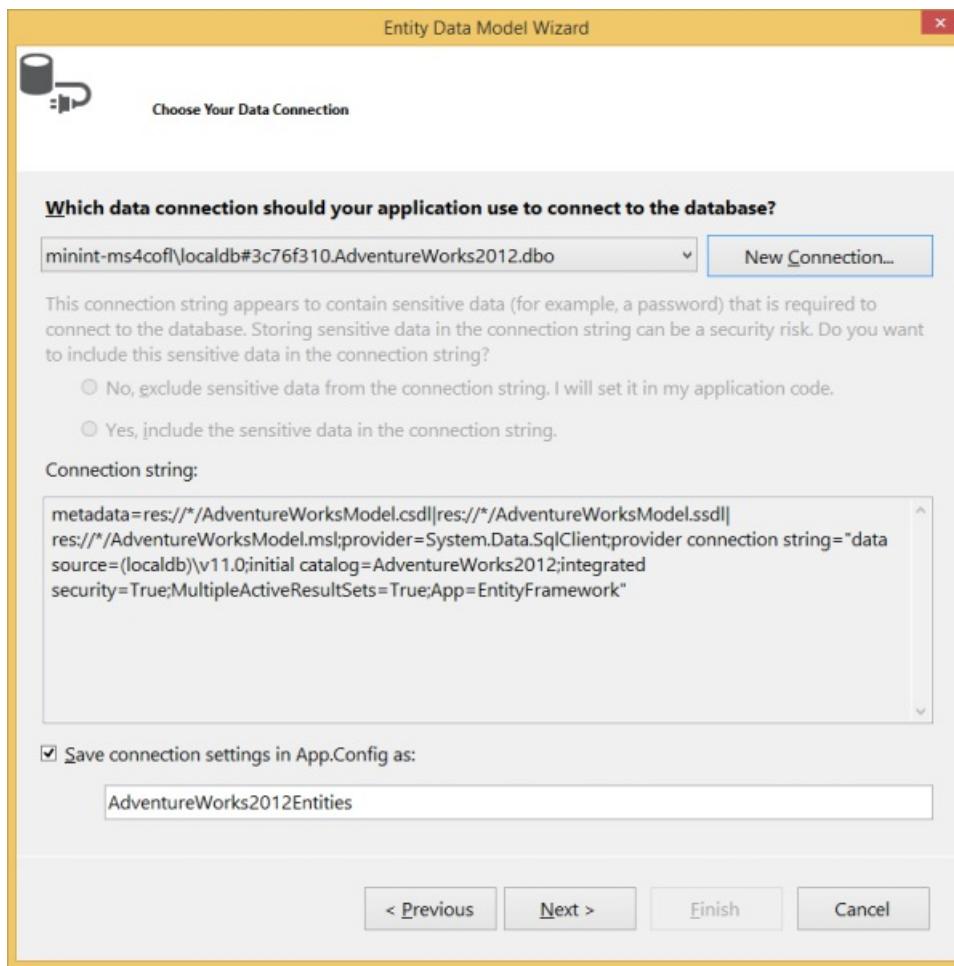
## Assistant ADO.NET Entity Data Model

### Étape 1 : Choisissez le contenu du modèle



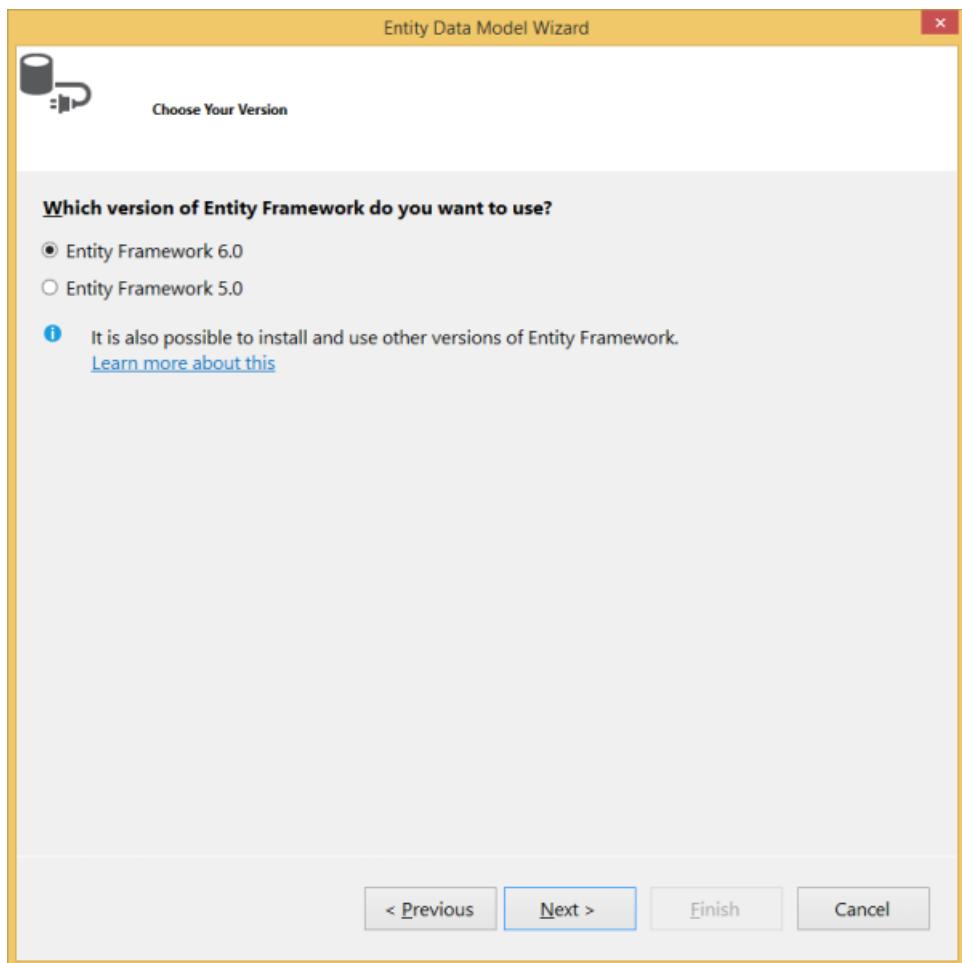
| RACCOURCI      | ACTION   | NOTES   |
|----------------|--|---|
| <b>ALT + n</b> | Déplacer vers l'écran suivant  | Non disponible pour toutes les sélections de contenu du modèle. |
| <b>Alt + f</b> | Terminer l'Assistant   | Non disponible pour toutes les sélections de contenu du modèle. |
| <b>ALT + w</b> | Déplacer le focus vers le « ce que le modèle devrait contenir ? » volet. |   |

### Étape 2 : Choisir votre connexion



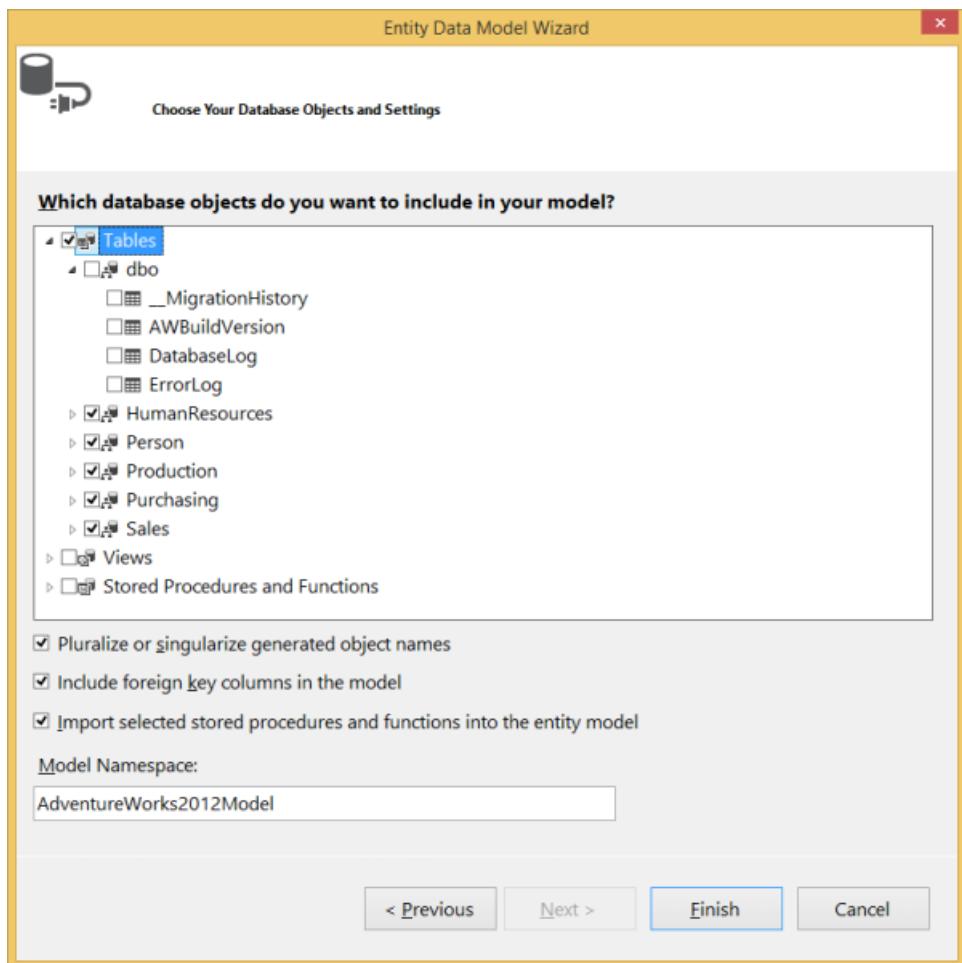
| RACCOURCI      | ACTION  | NOTES   |
|----------------|---|---|
| <b>ALT + n</b> | Déplacer vers l'écran suivant   |   |
| <b>ALT + p</b> | Déplacer vers l'écran précédent   |   |
| <b>ALT + w</b> | Déplacer le focus vers le « ce que le modèle devrait contenir ? » volet.                |   |
| <b>Alt + c</b> | Ouvrez la fenêtre « Propriétés de connexion »   | Permet la définition d'une nouvelle connexion de base de données. |
| <b>Alt + e</b> | Exclure les données sensibles de la chaîne de connexion                                 |   |
| <b>Alt + i</b> | Inclure les données sensibles dans la chaîne de connexion                               |   |
| <b>Alt + s</b> | Activer/désactiver l'option « Enregistrer les paramètres de connexion dans App.Config » |   |

### Étape 3 : Choisir votre Version



| RACCOURCI | ACTION   | NOTES   |
|-----------|--|---|
| ALT + n   | Déplacer vers l'écran suivant                                      |   |
| ALT + p   | Déplacer vers l'écran précédent                                    |   |
| ALT + w   | Déplacer le focus vers la sélection de la version Entity Framework | Permet de spécifier une autre version d'Entity Framework pour une utilisation dans le projet. |

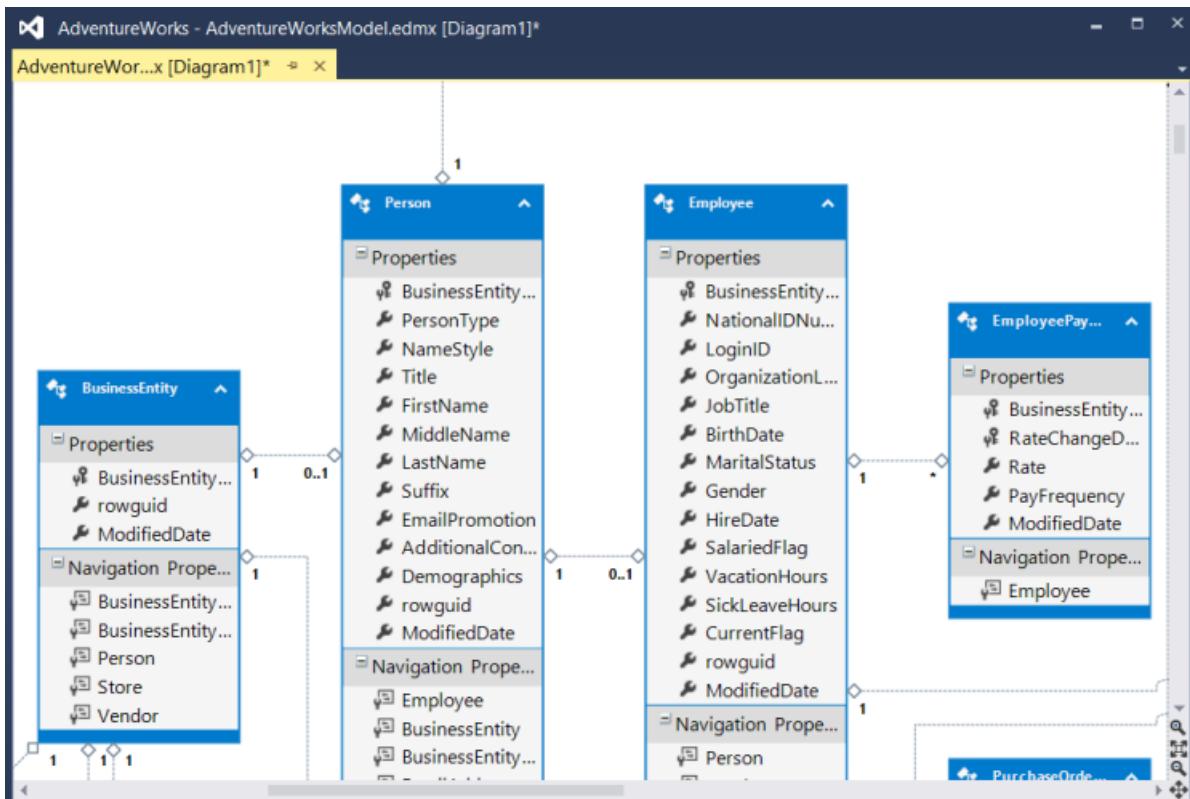
#### Étape 4 : Choisir vos objets de base de données et les paramètres



| RACCOURCI      | ACTION   | NOTES   |
|----------------|--|---|
| <b>Alt + f</b> | Terminer l'Assistant   |   |
| <b>ALT + p</b> | Déplacer vers l'écran précédent  |   |
| <b>ALT + w</b> | Déplacer le focus vers le volet de sélection d'objets de base de données   | Permet de spécification d'objets de base de données pour être inverse conçu.    |
| <b>Alt + s</b> | Activer/désactiver le « mettre au pluriel ou au singulier les noms d'objets générés » option                       |   |
| <b>ALT + k</b> | Activer/désactiver l'option « Inclure les colonnes clés étrangères dans le modèle »                                | Non disponible pour toutes les sélections de contenu du modèle.                 |
| <b>Alt + i</b> | Activer/désactiver l'option « Importer sélectionnés des procédures stockées et fonctions dans le modèle d'entité » | Non disponible pour toutes les sélections de contenu du modèle.                 |
| <b>ALT + m</b> | Place le focus sur le champ de texte « Modèle Namespace »  | Non disponible pour toutes les sélections de contenu du modèle.                 |
| <b>Espace</b>  | Sélection à bascule sur l'élément  | Si l'élément a des enfants, tous les éléments enfants seront également basculer |
| <b>Gauche</b>  | Réduire l'arborescence enfant  |   |

| RACCOURCI          | ACTION  | NOTES |
|--------------------|---|-------|
| <b>Droite</b>      | Développez l'arborescence enfant                  |       |
| <b>À distance</b>  | Accédez à l'élément précédent dans l'arborescence |       |
| <b>Vers le bas</b> | Accédez à l'élément suivant dans l'arborescence   |       |

## Aire du Concepteur EF



| RACCOURCI                 | ACTION                | NOTES  |
|---------------------------|-----------------------|--|
| <b>Entrez/espace</b>      | Basculer la sélection | Active ou désactive la sélection sur l'objet qui a le focus.   |
| <b>Échap</b>              | Annuler la sélection  | Annule la sélection actuelle.  |
| <b>CTRL + A</b>           | Tout Sélectionner     | Sélectionne toutes les formes sur l'aire de conception.  |
| <b>Flèche haut</b>        | Monter                | Déplace les entités d'un incrément de grille sélectionnés.<br>Si, dans une liste, déplace vers le sous-champ frère précédent.      |
| <b>Flèche vers le bas</b> | Descendre             | Déplace l'entité vers le bas un incrément de grille sélectionnée.<br>Si, dans une liste, déplace vers le sous-champ frère suivant. |

| RACCOURCI                     | ACTION                       | NOTES  |
|-------------------------------|------------------------------|--|
| <b>Flèche gauche</b>          | Déplacer à gauche            | Déplace l'entité gauche un incrément de grille sélectionnée.<br>Si, dans une liste, déplace vers le sous-champ frère précédent.  |
| <b>Flèche droite</b>          | Déplacer vers la droite      | Déplace l'incrément de grille droite une entité sélectionnée.<br>Si, dans une liste, déplace vers le sous-champ frère suivant.   |
| <b>Maj + flèche gauche</b>    | Forme de taille de gauche    | Réduit la largeur de l'entité sélectionnée par un incrément de grille.   |
| <b>Maj + flèche droite</b>    | Forme de taille appropriée   | Augmente la largeur de l'entité sélectionnée par un incrément de grille.   |
| <b>Accueil</b>                | Premiers homologues          | Déplace le focus et la sélection au premier objet sur l'aire de conception au même niveau de pair.   |
| <b>Fin</b>                    | Dernière homologue           | Déplace le focus et la sélection vers le dernier objet sur l'aire de conception au même niveau de pair.  |
| <b>CTRL + origine</b>         | Premiers homologues (focus)  | Identique au premier homologue, mais déplace le focus au lieu de déplacer la sélection.  |
| <b>CTRL + fin</b>             | Dernière homologue (focus)   | Identique à la dernière de l'homologue, mais déplace le focus au lieu de déplacer la sélection.  |
| <b>Tab</b>                    | Homologue suivant            | Déplace le focus et la sélection vers l'objet suivant sur l'aire de conception au même niveau de pair.   |
| <b>Maj+Tab</b>                | Homologue précédente         | Déplace le focus et la sélection à l'objet précédent sur l'aire de conception au même niveau de pair.  |
| <b>Ctrl + Alt + Tab</b>       | Homologue suivant (focus)    | Identique à suivant de l'homologue, mais déplace le focus au lieu de déplacer la sélection.  |
| <b>Alt + Ctrl + Maj + Tab</b> | Homologue précédente (focus) | Identique au précédent homologue, mais déplace le focus au lieu de déplacer la sélection.  |
| <                             | Remonter                     | Se déplace vers l'objet suivant sur la surface de conception au niveau supérieur dans la hiérarchie. S'il n'existe aucune forme au-dessus de cette forme dans la hiérarchie (autrement dit, l'objet est placé directement sur l'aire de conception), le diagramme est sélectionné. |

| RACCOURCI  | ACTION                              | NOTES   |
|--|-------------------------------------|---|
| >  | Descendre                           | Se déplace vers l'objet de relation contenant-contenu suivant sur la conception surface un niveau en dessous celle-ci dans la hiérarchie. S'il n'existe aucun objet de relation contenant-contenu, il s'agit d'une absence d'opération. |
| <b>CTRL + &lt;</b>   | Remonter (focus)                    | Identique à remonter la commande, mais déplace le focus sans la sélection.  |
| <b>CTRL + &gt;</b>   | Descendant (focus)                  | Identique à descendant commande, mais déplace le focus sans la sélection.   |
| <b>MAJ + fin</b>   | Suivez à connecté                   | À partir d'une entité, se déplace à une entité qui n'est connectée à cette entité.  |
| <b>Suppr</b>   | Supprimer                           | Supprimer un objet ou un connecteur à partir du diagramme.  |
| <b>Ins</b>   | Insert                              | Ajoute une nouvelle propriété à une entité lorsque l'en-tête de compartiment de « Propriétés scalaires » ou une propriété proprement dite est sélectionnée.   |
| <b>Page préc.</b>  | Diagramme de défilement des         | Fait défiler vers l'aire de conception, incrément égaux à 75 % de la hauteur de l'aire de conception actuellement visible.  |
| <b>Page suiv.</b>  | Diagramme de défilement vers le bas | Fait défiler l'aire de conception.  |
| <b>Maj + Pg vers le bas</b>  | Diagramme de défilement droite      | Fait défiler vers l'aire de conception vers la droite.  |
| <b>MAJ + Page préc.</b>  | Diagramme de défilement gauche      | Fait défiler vers l'aire de conception vers la gauche.  |
| <b>F2</b>  | Entrer en mode édition              | Raccourci clavier standard pour entrer en mode édition pour un contrôle de texte.   |
| <b>MAJ + F10</b>   | Afficher le menu contextuel         | Raccourci clavier standard pour afficher le menu contextuel d'un élément sélectionné.   |
| <b>CTRL + MAJ + souris clic gauche</b><br><b>CTRL + MAJ + molette de souris vers l'avant</b> | Zoom sémantique dans                | Effectue un zoom avant sur la zone de l'affichage des diagrammes sous le pointeur de la souris.   |

| RACCOURCI  | ACTION                               | NOTES   |
|--|--------------------------------------|---|
| <b>CTRL + MAJ + la souris avec le bouton droit cliquez sur CTRL + MAJ + molette de souris vers l'arrière</b> | Zoom sémantique Out                  | Effectue un zoom arrière à partir de la zone de l'affichage des diagrammes sous le pointeur de la souris. Il ré-concentre le diagramme lorsque vous effectuer un zoom arrière trop loin du centre de diagramme actuel.                    |
| <b>CTRL + MAJ + '+' CTRL + molette de souris vers l'avant</b>  | Zoom avant                           | Effectue un zoom avant sur le centre de la vue de diagramme.  |
| <b>CTRL + MAJ + '-' CTRL + molette de souris vers l'arrière</b>  | Zoom arrière                         | Effectue un zoom arrière à partir de la zone sélectionnée de la vue de diagramme. Il ré-concentre le diagramme lorsque vous effectuer un zoom arrière trop loin du centre de diagramme actuel.  |
| <b>CTRL + MAJ + dessiner un rectangle avec le bouton gauche de la souris vers le bas</b>                     | Zone de zoom                         | Effectue un zoom centrée sur la zone que vous avez sélectionné. Lorsque vous maintenez le contrôle + MAJ enfoncées, vous verrez que le curseur se transforme en loupe, ce qui vous permet de définir la zone pour effectuer un zoom dans. |
| <b>Touche du Menu contextuel + suis '</b>  | Ouvrez la fenêtre Détails de mappage | Ouvre la fenêtre Détails de mappage pour modifier les mappages pour l'entité sélectionnée   |

## Fenêtre Détails de mappage

The screenshot shows the 'Mapping Details - Person' window. On the left, there's a tree view under 'Tables' with nodes for 'Maps to Person' (expanded) and 'Column Mappings'. Under 'Maps to Person', there's a node for 'Person' with a plus sign to add conditions. Under 'Column Mappings', there are 15 pairs of columns from the 'Person' table on the left and the 'BusinessEntity' table on the right, each with a double-headed arrow indicating a mapping. The columns listed are: BusinessEntityID, PersonType, NameStyle, Title, FirstName, MiddleName, LastName, Suffix, EmailPromotion, AdditionalContactInfo, Demographics, rowguid, and ModifiedDate. The 'BusinessEntity' side shows types like Int32, String, Boolean, etc.

| RACCOURCI | ACTION | NOTES |
|-----------|--------|-------|
|           |        |       |

| RACCOURCI                       | ACTION               | NOTES  |
|---------------------------------|----------------------|--|
| <b>Tab</b>                      | Basculez le contexte | Bascule entre la zone de fenêtre principale et de la barre d'outils sur la gauche  |
| <b>Touches de direction</b>     | Navigation           | Vous déplacer dans les lignes, ou à droite et gauche sur les colonnes de la zone de fenêtre principale. Déplacer entre les boutons dans la barre d'outils sur la gauche. |
| <b>Entrée Espace</b>            | Sélectionner         | Sélectionne un bouton dans la barre d'outils sur la gauche.  |
| <b>ALT + flèche vers le bas</b> | Ouvrir la liste      | Une liste déroulante si une cellule est sélectionnée qui a une zone de liste déroulante.   |
| <b>Entrée</b>                   | Liste, sélectionnez  | Sélectionne un élément dans une zone de liste déroulante.  |
| <b>Échap</b>                    | Liste de fermeture   | Ferme une zone de liste déroulante.  |

## Navigation de Visual Studio

Entity Framework fournit également un nombre d'actions qui peuvent avoir des raccourcis clavier personnalisés mappés (aucuns raccourcis ne sont mappés par défaut). Pour créer ces raccourcis peuvent être personnalisés, cliquez sur le menu Outils, puis sur Options. Sous l'environnement, choisissez le clavier. Défiler la liste du milieu jusqu'à ce que vous pouvez sélectionner la commande souhaitée, entrez le raccourci dans la zone de texte « Appuyez sur les touches de raccourci » et cliquez sur attribuer. Les raccourcis possibles sont les suivantes :

| RACCOURCI  |
|--|
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.Add.ComplexProperty.ComplexTypes</b> |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddCodeGenerationItem</b>            |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddFunctionImport</b>                |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.AddEnumType</b>               |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.Association</b>               |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.ComplexProperty</b>           |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.ComplexType</b>               |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.Entity</b>                    |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.FunctionImport</b>            |
| <b>OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.Inheritance</b>               |

## RACCOURCI

**OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.NavigationProperty**

**OtherContextMenus.MicrosoftDataEntityDesignContext.AddNew.ScalarProperty**

**OtherContextMenus.MicrosoftDataEntityDesignContext.AddNewDiagram**

**OtherContextMenus.MicrosoftDataEntityDesignContext.AddtoDiagram**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Close**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Collapse**

**OtherContextMenus.MicrosoftDataEntityDesignContext.ConverttoEnum**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.CollapseAll**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.ExpandAll**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.ExportasImage**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Diagram.LayoutDiagram**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Edit**

**OtherContextMenus.MicrosoftDataEntityDesignContext.EntityKey**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Expand**

**OtherContextMenus.MicrosoftDataEntityDesignContext.FunctionImportMapping**

**OtherContextMenus.MicrosoftDataEntityDesignContext.GenerateDatabasefromModel**

**OtherContextMenus.MicrosoftDataEntityDesignContext.GoToDefinition**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Grid>ShowGrid**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Grid.SnaptoGrid**

**OtherContextMenus.MicrosoftDataEntityDesignContext.IncludeRelated**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Mapping Details**

**OtherContextMenus.MicrosoftDataEntityDesignContext.ModelBrowser**

**OtherContextMenus.MicrosoftDataEntityDesignContext.MoveDiagramstoSeparateFile**

**OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Down**

**OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Down5**

## RACCOURCI

**OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.ToBottom**

**OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.ToTop**

**OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Up**

**OtherContextMenus.MicrosoftDataEntityDesignContext.MoveProperties.Up5**

**OtherContextMenus.MicrosoftDataEntityDesignContext.MovetonewDiagram**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Open**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Refactor.MoveToNewComplexType**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Refactor.Rename**

**OtherContextMenus.MicrosoftDataEntityDesignContext.RemovefromDiagram**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Rename**

**OtherContextMenus.MicrosoftDataEntityDesignContext.ScalarPropertyFormat.DisplayName**

**OtherContextMenus.MicrosoftDataEntityDesignContext.ScalarPropertyFormat.DisplayNameandType**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Select.Base Type**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Select.Entity**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Select.Property**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Select.Subtype**

**OtherContextMenus.MicrosoftDataEntityDesignContext.SelectAll**

**OtherContextMenus.MicrosoftDataEntityDesignContext.SelectAssociation**

**OtherContextMenus.MicrosoftDataEntityDesignContext.ShowinDiagram**

**OtherContextMenus.MicrosoftDataEntityDesignContext.ShowinModelBrowser**

**OtherContextMenus.MicrosoftDataEntityDesignContext.StoredProcedureMapping**

**OtherContextMenus.MicrosoftDataEntityDesignContext.TableMapping**

**OtherContextMenus.MicrosoftDataEntityDesignContext.UpdateModelfromDatabase**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Validate**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.10**

RACCOURCI

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.100**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.125**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.150**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.200**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.25**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.300**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.33**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.400**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.50**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.66**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.75**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.Custom**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.ZoomIn**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.ZoomOut**

**OtherContextMenus.MicrosoftDataEntityDesignContext.Zoom.ZoomToFit**

**View.EntityDataModelBrowser**

**View.EntityDataModelMappingDetails**

# Interrogation et recherche d'entités

13/09/2018 • 6 minutes to read

Cette rubrique décrit les différentes méthodes de recherche de données à l'aide d'Entity Framework, y compris LINQ et la méthode Find. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

## Recherche d'entités à l'aide d'une requête

DbSet et IDbSet implémentent IQueryble et peuvent donc servir de point de départ pour écrire une requête LINQ sur la base de données. Nous ne décrirons pas LINQ en détail ici, mais voici quelques exemples simples :

```
using (var context = new BloggingContext())
{
    // Query for all blogs with names starting with B
    var blogs = from b in context.Blogs
                where b.Name.StartsWith("B")
                select b;

    // Query for the Blog named ADO.NET Blog
    var blog = context.Blogs
        .Where(b => b.Name == "ADO.NET Blog")
        .FirstOrDefault();
}
```

Notez que DbSet et IDbSet créent toujours des requêtes sur la base de données et impliquent toujours un aller-retour vers la base de données même si les entités retournées existent déjà dans le contexte. Une requête est exécutée sur la base de données quand :

- Elle est énumérée par une instruction **foreach** (C#) ou **For Each** (Visual Basic).
- Elle est énumérée par une opération de collection comme [ToArray](#), [ToDictionary](#) ou [ToList](#).
- Des opérateurs LINQ, comme [First](#) ou [Any](#) sont spécifiés dans la partie la plus extérieure de la requête.
- Les méthodes suivantes sont appelées : la méthode d'extension [Load](#) sur DbSet, [DbEntityEntry.Reload](#) et [Database.ExecuteSqlCommand](#).

Quand les résultats sont retournés à partir de la base de données, les objets qui n'existent pas dans le contexte sont attachés au contexte. Si un objet est déjà dans le contexte, l'objet existant est retourné (les valeurs actuelles et d'origine des propriétés de l'objet dans l'entrée ne sont **pas** remplacées par les valeurs de la base de données).

Quand vous exécutez une requête, les entités qui ont été ajoutées au contexte, mais n'ont pas encore été enregistrées dans la base de données, ne sont pas retournées dans le jeu de résultats. Pour obtenir les données du contexte, consultez [Données locales](#).

Si une requête ne retourne aucune ligne de la base de données, le résultat est une collection vide au lieu de **null**.

## Recherche d'entités à l'aide de clés primaires

La méthode Find sur DbSet utilise la valeur de clé primaire pour tenter de trouver une entité suivie par le contexte. Si l'entité est introuvable dans le contexte, une requête est envoyée à la base de données pour y rechercher l'entité. Null est retourné si l'entité est introuvable dans le contexte ou dans la base de données.

Find est différent de l'utilisation d'une requête pour deux raisons :

- Un aller-retour vers la base de données est effectué uniquement si l'entité avec la clé spécifiée est introuvable dans le contexte.
- Find retourne les entités qui sont dans l'état Added. Autrement dit, Find retourne les entités qui ont été ajoutées au contexte, mais n'ont pas encore été enregistrées dans la base de données.

## Recherche d'une entité par clé primaire

Le code suivant montre des exemples d'utilisation de Find :

```
using (var context = new BloggingContext())
{
    // Will hit the database
    var blog = context.Blogs.Find(3);

    // Will return the same instance without hitting the database
    var blogAgain = context.Blogs.Find(3);

    context.Blogs.Add(new Blog { Id = -1 });

    // Will find the new blog even though it does not exist in the database
    var newBlog = context.Blogs.Find(-1);

    // Will find a User which has a string primary key
    var user = context.Users.Find("johndoe1987");
}
```

## Recherche d'une entité par clé primaire composite

Entity Framework permet à vos entités d'avoir des clés composites, c'est-à-dire des clés composées de plusieurs propriétés. Par exemple, vous avez une entité BlogSettings qui représente des paramètres utilisateur pour un blog particulier. Comme un utilisateur n'aura jamais une entité BlogSettings pour chaque blog, vous pouvez choisir pour BlogSettings une clé primaire qui est une combinaison de BlogId et Username. Le code suivant tente de rechercher l'entité BlogSettings avec BlogId = 3 et Username = "johndoe1987" :

```
using (var context = new BloggingContext())
{
    var settings = context.BlogSettings.Find(3, "johndoe1987");
}
```

Quand vous avez des clés composites, vous devez utiliser ColumnAttribute ou l'API Fluent pour spécifier l'ordre des propriétés de la clé composite. L'appel de Find doit utiliser cet ordre quand vous spécifiez les valeurs qui forment la clé.

# Méthode Load

26/10/2018 • 2 minutes to read

Il existe plusieurs scénarios où vous souhaitez charger des entités à partir de la base de données dans le contexte sans immédiatement sert à rien avec ces entités. Un bon exemple est chargement d'entités pour la liaison de données comme décrit dans [données locales](#). Une méthode courante pour ce faire consiste à écrire une requête LINQ, puis appelez `ToList` dessus, uniquement pour ignorer la liste créée. La méthode d'extension `Load` fonctionne exactement comme `ToList`, sauf qu'elle évite la création de la liste complètement.

Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

Voici deux exemples d'utilisation de charge. La première est effectuée à partir d'une application Windows Forms de liaison de données où la charge est utilisé pour interroger des entités avant la liaison à la collection locale, comme décrit dans [données locales](#):

```
protected override void OnLoad(EventArgs e)
{
    base.OnLoad(e);

    _context = new ProductContext();

    _context.Categories.Load();
    categoryBindingSource.DataSource = _context.Categories.Local.ToBindingList();
}
```

Le deuxième exemple s'affiche à l'aide de charge pour charger une collection filtrée des entités associées, comme décrit dans [le chargement des entités associées](#):

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Load the posts with the 'entity-framework' tag related to a given blog
    context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Where(p => p.Tags.Contains("entity-framework"))
        .Load();
}
```

# Données locales

11/10/2019 • 17 minutes to read

L'exécution d'une requête LINQ directement sur un DbSet enverra toujours une requête à la base de données, mais vous pouvez accéder aux données qui sont actuellement en mémoire à l'aide de la propriété DbSet. Local. Vous pouvez également accéder aux informations supplémentaires EF en effectuant le suivi de vos entités à l'aide des méthodes DbContext. Entry et DbContext. ChangeTracker. Entries. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

## Utilisation de local pour examiner les données locales

La propriété locale de DbSet fournit un accès simple aux entités du jeu qui font actuellement l'objet d'un suivi par le contexte et qui n'ont pas été marquées comme supprimées. L'accès à la propriété locale n'entraîne jamais l'envoi d'une requête à la base de données. Cela signifie qu'il est généralement utilisé après qu'une requête a déjà été exécutée. La méthode d'extension Load peut être utilisée pour exécuter une requête afin que le contexte effectue le suivi des résultats. Exemple :

```
using (var context = new BloggingContext())
{
    // Load all blogs from the database into the context
    context.Blogs.Load();

    // Add a new blog to the context
    context.Blogs.Add(new Blog { Name = "My New Blog" });

    // Mark one of the existing blogs as Deleted
    context.Blogs.Remove(context.Blogs.Find(1));

    // Loop over the blogs in the context.
    Console.WriteLine("In Local: ");
    foreach (var blog in context.Blogs.Local)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            blog.BlogId,
            blog.Name,
            context.Entry(blog).State);
    }

    // Perform a query against the database.
    Console.WriteLine("\nIn DbSet query: ");
    foreach (var blog in context.Blogs)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            blog.BlogId,
            blog.Name,
            context.Entry(blog).State);
    }
}
```

Si nous avions deux blogs dans la base de données « ADO.NET blog » avec un BlogId de 1 et un « the Visual Studio blog » avec un BlogId de 2, nous pourrions attendre la sortie suivante :

```
In Local:  
Found 0: My New Blog with state Added  
Found 2: The Visual Studio Blog with state Unchanged
```

```
In DbSet query:  
Found 1: ADO.NET Blog with state Deleted  
Found 2: The Visual Studio Blog with state Unchanged
```

Cela illustre trois points :

- Le nouveau blog « mon nouveau blog » est inclus dans la collection locale, même s'il n'a pas encore été enregistré dans la base de données. Ce blog a une clé primaire égale à zéro, car la base de données n'a pas encore généré de clé réelle pour l'entité.
- Le « blog ADO.NET » n'est pas inclus dans la collection locale, bien qu'il soit toujours suivi par le contexte. Cela est dû au fait que nous avons supprimé le DbSet, ce qui le marque comme étant supprimé.
- Lorsque DbSet est utilisé pour exécuter une requête, le blog marqué pour suppression (blog ADO.NET) est inclus dans les résultats et le nouveau blog (mon nouveau blog) qui n'a pas encore été enregistré dans la base de données n'est pas inclus dans les résultats. Cela est dû au fait que DbSet exécute une requête sur la base de données et que les résultats retournés reflètent toujours ce qui se trouve dans la base de données.

## Utilisation de l'environnement local pour ajouter et supprimer des entités dans le contexte

La propriété locale sur DbSet retourne un [ObservableCollection](#) avec des événements raccordés de sorte qu'il reste synchronisé avec le contenu du contexte. Cela signifie que des entités peuvent être ajoutées ou supprimées de la collection locale ou du DbSet. Cela signifie également que les requêtes qui apportent de nouvelles entités dans le contexte entraînent la mise à jour de la collection locale avec ces entités. Exemple :

```

using (var context = new BloggingContext())
{
    // Load some posts from the database into the context
    context.Posts.Where(p => p.Tags.Contains("entity-framework")).Load();

    // Get the local collection and make some changes to it
    var localPosts = context.Posts.Local;
    localPosts.Add(new Post { Name = "What's New in EF" });
    localPosts.Remove(context.Posts.Find(1));

    // Loop over the posts in the context.
    Console.WriteLine("In Local after entity-framework query: ");
    foreach (var post in context.Posts.Local)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            post.Id,
            post.Title,
            context.Entry(post).State);
    }

    var post1 = context.Posts.Find(1);
    Console.WriteLine(
        "State of post 1: {0} is {1}",
        post1.Name,
        context.Entry(post1).State);

    // Query some more posts from the database
    context.Posts.Where(p => p.Tags.Contains("asp.net")).Load();

    // Loop over the posts in the context again.
    Console.WriteLine("\nIn Local after asp.net query: ");
    foreach (var post in context.Posts.Local)
    {
        Console.WriteLine(
            "Found {0}: {1} with state {2}",
            post.Id,
            post.Title,
            context.Entry(post).State);
    }
}

```

En supposant que nous avions quelques publications marquées avec 'Entity-Framework' et 'asp.net', la sortie peut ressembler à ceci :

```

In Local after entity-framework query:
Found 3: EF Designer Basics with state Unchanged
Found 5: EF Code First Basics with state Unchanged
Found 0: What's New in EF with state Added
State of post 1: EF Beginners Guide is Deleted

In Local after asp.net query:
Found 3: EF Designer Basics with state Unchanged
Found 5: EF Code First Basics with state Unchanged
Found 0: What's New in EF with state Added
Found 4: ASP.NET Beginners Guide with state Unchanged

```

Cela illustre trois points :

- La nouvelle publication « nouveautés dans EF » qui a été ajoutée à la collection locale est suivie par le contexte dans l'état ajouté. Elle est donc insérée dans la base de données lorsque SaveChanges est appelé.
- La publication qui a été supprimée de la collection locale (Guide du débutant d'EF) est désormais marquée comme supprimée dans le contexte. Elle sera donc supprimée de la base de données lorsque SaveChanges est

appelé.

- La publication supplémentaire (Guide du débutant ASP.NET) chargée dans le contexte avec la seconde requête est automatiquement ajoutée à la collection locale.

Une dernière chose à noter sur la version locale est qu'il s'agit d'une performance ObservableCollection qui n'est pas idéale pour un grand nombre d'entités. Par conséquent, si vous traitez des milliers d'entités dans votre contexte, il peut être déconseillé d'utiliser local.

## Utilisation de la liaison de données locale pour WPF

La propriété locale sur DbSet peut être utilisée directement pour la liaison de données dans une application WPF, car il s'agit d'une instance de ObservableCollection. Comme décrit dans les sections précédentes, cela signifie qu'elle sera automatiquement synchronisée avec le contenu du contexte et que le contenu du contexte sera automatiquement synchronisé avec lui. Notez que vous devez préremplir la collection locale avec des données pour qu'il n'y ait rien à lier, car local n'entraîne jamais de requête de base de données.

Il ne s'agit pas d'un emplacement approprié pour un exemple de liaison de données WPF complète, mais les éléments clés sont :

- Configurer une source de liaison
- Liez-le à la propriété locale de votre ensemble
- Remplir localement à l'aide d'une requête dans la base de données.

## Liaison WPF avec les propriétés de navigation

Si vous effectuez une liaison de données maître/détail, vous pouvez lier l'affichage détails à une propriété de navigation de l'une de vos entités. Un moyen simple d'effectuer ce travail consiste à utiliser ObservableCollection pour la propriété de navigation. Exemple :

```
public class Blog
{
    private readonly ObservableCollection<Post> _posts =
        new ObservableCollection<Post>();

    public int BlogId { get; set; }
    public string Name { get; set; }

    public virtual ObservableCollection<Post> Posts
    {
        get { return _posts; }
    }
}
```

## Utilisation de local pour nettoyer les entités dans SaveChanges

Dans la plupart des cas, les entités supprimées d'une propriété de navigation ne sont pas automatiquement marquées comme supprimées dans le contexte. Par exemple, si vous supprimez un objet post de la collection blog.publications, cette publication ne sera pas automatiquement supprimée lorsque SaveChanges est appelé. Si vous avez besoin de la supprimer, vous devrez peut-être trouver ces entités en suspens et les marquer comme supprimées avant d'appeler SaveChanges ou dans le cadre d'une SaveChanges substituée. Exemple :

```

public override int SaveChanges()
{
    foreach (var post in this.Posts.Local.ToList())
    {
        if (post.Blog == null)
        {
            this.Posts.Remove(post);
        }
    }

    return base.SaveChanges();
}

```

Le code ci-dessus utilise la collection locale pour rechercher toutes les publications et marque toutes celles qui n'ont pas de référence de blog comme étant supprimées. L'appel de `ToList()` est requis, car sinon, la collection est modifiée par l'appel `Remove` pendant son énumération. Dans la plupart des autres cas, vous pouvez interroger directement la propriété locale sans utiliser `ToList` en premier.

## Utilisation des paramètres local et ToBindingList pour la liaison de données Windows Forms

Windows Forms ne prend pas en charge la liaison de données de fidélité complète avec `ObservableCollection` directement. Toutefois, vous pouvez toujours utiliser la propriété locale `DbSet` pour la liaison de données afin d'obtenir tous les avantages décrits dans les sections précédentes. Pour cela, vous devez utiliser la méthode d'extension `ToBindingList`, qui crée une implémentation `IBindingList` sauvegardée par l'`ObservableCollection` local.

Il ne s'agit pas d'un emplacement approprié pour un exemple de liaison de données Windows Forms complète, mais les éléments clés sont :

- Configurer une source de liaison d'objet
- Liez-le à la propriété locale de votre jeu à l'aide de `local.ToBindingList()`
- Remplir localement à l'aide d'une requête dans la base de données

## Obtention d'informations détaillées sur les entités suivies

La plupart des exemples de cette série utilisent la méthode `Entry` pour retourner une instance `DbEntityEntry` pour une entité. Cet objet d'entrée sert ensuite de point de départ pour la collecte d'informations sur l'entité, par exemple son état actuel, ainsi que pour l'exécution d'opérations sur l'entité, telles que le chargement explicite d'une entité associée.

Les méthodes d'entrée retournent des objets `DbEntityEntry` pour la plupart ou toutes les entités faisant l'objet d'un suivi par le contexte. Cela vous permet de collecter des informations ou d'effectuer des opérations sur de nombreuses entités plutôt que sur une seule entrée. Exemple :

```

using (var context = new BloggingContext())
{
    // Load some entities into the context
    context.Blogs.Load();
    context.Authors.Load();
    context.Readers.Load();

    // Make some changes
    context.Blogs.Find(1).Title = "The New ADO.NET Blog";
    context.Blogs.Remove(context.Blogs.Find(2));
    context.Authors.Add(new Author { Name = "Jane Doe" });
    context.Readers.Find(1).Username = "johndoe1987";

    // Look at the state of all entities in the context
    Console.WriteLine("All tracked entities: ");
    foreach (var entry in context.ChangeTracker.Entries())
    {
        Console.WriteLine(
            "Found entity of type {0} with state {1}",
            ObjectContext.GetObjectType(entry.Entity.GetType()).Name,
            entry.State);
    }

    // Find modified entities of any type
    Console.WriteLine("\nAll modified entities: ");
    foreach (var entry in context.ChangeTracker.Entries()
        .Where(e => e.State == EntityState.Modified))
    {
        Console.WriteLine(
            "Found entity of type {0} with state {1}",
            ObjectContext.GetObjectType(entry.Entity.GetType()).Name,
            entry.State);
    }

    // Get some information about just the tracked blogs
    Console.WriteLine("\nTracked blogs: ");
    foreach (var entry in context.ChangeTracker.Entries<Blog>())
    {
        Console.WriteLine(
            "Found Blog {0}: {1} with original Name {2}",
            entry.Entity.BlogId,
            entry.Entity.Name,
            entry.Property(p => p.Name).OriginalValue);
    }

    // Find all people (author or reader)
    Console.WriteLine("\nPeople: ");
    foreach (var entry in context.ChangeTracker.Entries<IPerson>())
    {
        Console.WriteLine("Found Person {0}", entry.Entity.Name);
    }
}

```

Vous remarquerez que nous introduisons une classe Author et Reader dans l'exemple, ces deux classes implémentent l'interface IPerson.

```

public class Author : IPerson
{
    public int AuthorId { get; set; }
    public string Name { get; set; }
    public string Biography { get; set; }
}

public class Reader : IPerson
{
    public int ReaderId { get; set; }
    public string Name { get; set; }
    public string Username { get; set; }
}

public interface IPerson
{
    string Name { get; }
}

```

Supposons que nous ayons les données suivantes dans la base de données :

Blog avec BlogId = 1 et Name = 'ADO.NET blog'  
 Blog avec BlogId = 2 et Name = 'blog de Visual Studio'  
 Blog avec BlogId = 3 et Name = '.NET Framework blog'  
 Auteur avec réécriture = 1 et nom = « joe bloggs »  
 Lecteur avec ReaderId = 1 et Name = « John Doe »

La sortie de l'exécution du code est la suivante :

```

All tracked entities:
Found entity of type Blog with state Modified
Found entity of type Blog with state Deleted
Found entity of type Blog with state Unchanged
Found entity of type Author with state Unchanged
Found entity of type Author with state Added
Found entity of type Reader with state Modified

All modified entities:
Found entity of type Blog with state Modified
Found entity of type Reader with state Modified

Tracked blogs:
Found Blog 1: The New ADO.NET Blog with original Name ADO.NET Blog
Found Blog 2: The Visual Studio Blog with original Name The Visual Studio Blog
Found Blog 3: .NET Framework Blog with original Name .NET Framework Blog

People:
Found Person John Doe
Found Person Joe Bloggs
Found Person Jane Doe

```

Ces exemples illustrent plusieurs points :

- Les méthodes d'entrée retournent des entrées pour les entités dans tous les États, y compris Deleted. Comparez ce paramètre au paramètre local qui exclut les entités supprimées.
- Les entrées de tous les types d'entités sont retournées lorsque la méthode des entrées non génériques est utilisée. Lorsque la méthode des entrées génériques est utilisée, les entrées sont retournées uniquement pour les entités qui sont des instances du type générique. Il a été utilisé ci-dessus pour obtenir des entrées pour tous les blogs. Elle a également été utilisée pour obtenir des entrées pour toutes les entités qui implémentent IPerson. Cela démontre que le type générique n'a pas besoin d'être un type d'entité réel.
- LINQ to Objects peut être utilisé pour filtrer les résultats retournés. Il a été utilisé ci-dessus pour rechercher

des entités de tout type, à condition qu'elles soient modifiées.

Notez que les instances `DbEntityEntry` contiennent toujours une entité non null. Les entrées de relation et les entrées stub ne sont pas représentées en tant qu'instances `DbEntityEntry`. il n'est donc pas nécessaire de les filtrer.

# sans suivi

13/09/2018 • 2 minutes to read

Vous pouvez être amené à obtenir des entités d'une requête, mais n'a pas ces entités être suivis par le contexte. Cela peut entraîner de meilleures performances lors de l'interrogation pour un grand nombre d'entités dans les scénarios en lecture seule. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

Une nouvelle méthode d'extension AsNoTracking permet à n'importe quelle requête à exécuter de cette façon.  
Exemple :

```
using (var context = new BloggingContext())
{
    // Query for all blogs without tracking them
    var blogs1 = context.Blogs.AsNoTracking();

    // Query for some blogs without tracking them
    var blogs2 = context.Blogs
        .Where(b => b.Name.Contains(".NET"))
        .AsNoTracking()
        .ToList();
}
```

# Requêtes SQL brutes

27/09/2018 • 5 minutes to read

Entity Framework vous permet d'interroger à l'aide de LINQ avec vos classes d'entité. Toutefois, il peut arriver que vous souhaitez exécuter des requêtes à l'aide de requêtes SQL brutes directement par rapport à la base de données. Cela inclut l'appel de procédures stockées, qui peuvent être utiles pour les modèles de Code First qui ne prennent actuellement pas en charge le mappage à des procédures stockées. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

## Écriture de requêtes SQL pour les entités

La méthode `SqlQuery` sur `DbSet` permet à une requête SQL brute à écrire qui retournera des instances d'entité. Les objets rentrés sont suivis par le contexte comme ils le seraient si elles ont été rentrées par une requête LINQ. Exemple :

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.SqlQuery("SELECT * FROM dbo.Blogs").ToList();
```

Notez que, comme pour les requêtes LINQ, la requête n'est pas exécutée jusqu'à ce que les résultats sont énumérés, dans l'exemple ci-dessus, cela est effectué avec l'appel à `ToList`.

Soyez attentif à chaque fois que les requêtes SQL brutes sont écrits pour deux raisons. Tout d'abord, la requête doit être écrite pour vous assurer qu'elle renvoie uniquement les entités qui sont réellement du type demandé. Par exemple, lors de l'utilisation des fonctionnalités telles que l'héritage, il est facile d'écrire une requête qui crée des entités qui ont le type CLR approprié.

En second lieu, certains types de requêtes SQL brutes exposent à des risques de sécurité potentiels, notamment concernant les attaques par injection SQL. Assurez-vous que vous utilisez des paramètres dans votre requête dans la méthode correcte pour protéger contre ces attaques.

## Chargement d'entités à partir de procédures stockées

Vous pouvez utiliser `DbSet.SqlQuery` pour charger des entités à partir des résultats d'une procédure stockée. Par exemple, le code suivant appelle le `dbo`. Procédure `GetBlogs` dans la base de données :

```
using (var context = new BloggingContext())
{
    var blogs = context.Blogs.SqlQuery("dbo.GetBlogs").ToList();
```

Vous pouvez également transmettre des paramètres à une procédure stockée à l'aide de la syntaxe suivante :

```
using (var context = new BloggingContext())
{
    var blogId = 1;

    var blogs = context.Blogs.SqlQuery("dbo.GetBlogById @p0", blogId).Single();
```

## Écriture de requêtes SQL pour les types de non-entité

Une requête SQL renvoie des instances de n'importe quel type, y compris les types primitifs, peut être créée à l'aide de la méthode `SqlQuery` sur la classe de base de données. Exemple :

```
using (var context = new BloggingContext())
{
    var blogNames = context.Database.SqlQuery<string>(
        "SELECT Name FROM dbo.Blogs").ToList();
}
```

Les résultats retournés par `SqlQuery` sur la base de données ne sont jamais suivis par le contexte même si les objets sont des instances d'un type d'entité.

## Envoi de commandes à la base de données brutes

Commandes de non-requête peuvent être envoyés à la base de données à l'aide de la méthode `ExecuteSqlCommand` sur la base de données. Exemple :

```
using (var context = new BloggingContext())
{
    context.Database.ExecuteSqlCommand(
        "UPDATE dbo.Blogs SET Name = 'Another Name' WHERE BlogId = 1");
}
```

Notez que toutes les modifications apportées aux données dans la base de données à l'aide de `ExecuteSqlCommand` sont opaques pour le contexte jusqu'à ce que les entités chargées ou rechargées à partir de la base de données.

### Paramètres de sortie

Si les paramètres de sortie sont utilisés, leurs valeurs ne sera pas disponibles jusqu'à ce que les résultats ont été lues entièrement. Il s'agit en raison du comportement sous-jacent de `DbDataReader`, consultez [extraction de données à l'aide d'un DataReader](#) pour plus d'informations.

# Chargement des entités associées

16/09/2019 • 9 minutes to read

Entity Framework prend en charge trois méthodes de chargement des données associées : le chargement hâtif, le chargement différé et le chargement explicite. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

## Chargement hâtif

Le chargement hâtif est le processus par lequel une requête pour un type d'entité charge également des entités associées dans le cadre de la requête. Le chargement hâtif est obtenu à l'aide de la méthode `Include`. Par exemple, les requêtes ci-dessous chargent les blogs et toutes les publications associées à chaque blog.

```
using (var context = new BloggingContext())
{
    // Load all blogs and related posts.
    var blogs1 = context.Blogs
        .Include(b => b.Posts)
        .ToList();

    // Load one blog and its related posts.
    var blog1 = context.Blogs
        .Where(b => b.Name == "ADO.NET Blog")
        .Include(b => b.Posts)
        .FirstOrDefault();

    // Load all blogs and related posts
    // using a string to specify the relationship.
    var blogs2 = context.Blogs
        .Include("Posts")
        .ToList();

    // Load one blog and its related posts
    // using a string to specify the relationship.
    var blog2 = context.Blogs
        .Where(b => b.Name == "ADO.NET Blog")
        .Include("Posts")
        .FirstOrDefault();
}
```

### NOTE

`Include` est une méthode d'extension dans l'espace de noms `System.Data.Entity`. Assurez-vous que vous utilisez cet espace de noms.

## Chargement à plusieurs niveaux

Il est également possible de charger de façon dynamique plusieurs niveaux d'entités associées. Les requêtes ci-dessous montrent des exemples de la procédure à suivre pour les propriétés de navigation de collection et de référence.

```

using (var context = new BloggingContext())
{
    // Load all blogs, all related posts, and all related comments.
    var blogs1 = context.Blogs
        .Include(b => b.Posts.Select(p => p.Comments))
        .ToList();

    // Load all users, their related profiles, and related avatar.
    var users1 = context.Users
        .Include(u => u.Profile.Avatar)
        .ToList();

    // Load all blogs, all related posts, and all related comments
    // using a string to specify the relationships.
    var blogs2 = context.Blogs
        .Include("Posts.Comments")
        .ToList();

    // Load all users, their related profiles, and related avatar
    // using a string to specify the relationships.
    var users2 = context.Users
        .Include("Profile.Avatar")
        .ToList();
}

```

#### NOTE

Il n'est pas possible actuellement de filtrer les entités associées qui sont chargées. `Include` affichera toujours toutes les entités associées.

## Chargement différé

Le chargement différé est le processus par lequel une entité ou une collection d'entités est chargée automatiquement à partir de la base de données la première fois qu'une propriété faisant référence à l'entité/aux entités est accédée. Lors de l'utilisation des types d'entités POCO, le chargement différé s'effectue en créant des instances de types de proxy dérivés, puis en substituant les propriétés virtuelles pour ajouter le raccordement de chargement. Par exemple, lors de l'utilisation de la classe d'entité blog définie ci-dessous, les publications associées sont chargées la première fois que la propriété de navigation `publications` est accédée :

```

public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public string Tags { get; set; }

    public virtual ICollection<Post> Posts { get; set; }
}

```

### Désactivation du chargement différé pour la sérialisation

Le chargement différé et la sérialisation ne sont pas bien confondus. Si vous n'êtes pas prudent, vous pouvez terminer l'interrogation de votre base de données tout simplement parce que le chargement différé est activé. La plupart des sérialiseurs fonctionnent en accédant à chaque propriété sur une instance d'un type. L'accès aux propriétés déclenche un chargement différé, si bien que davantage d'entités sont sérialisées. Sur ces entités, les propriétés sont accessibles, et d'autres entités sont chargées. Il est recommandé de désactiver le chargement différé avant de sérialiser une entité. Les sections suivantes montrent comment procéder.

### Désactivation du chargement différé pour des propriétés de navigation spécifiques

Le chargement différé de la collection de publications peut être désactivé en rendant la propriété des publications non virtuelle :

```
public class Blog
{
    public int BlogId { get; set; }
    public string Name { get; set; }
    public string Url { get; set; }
    public string Tags { get; set; }

    public ICollection<Post> Posts { get; set; }
}
```

Le chargement de la collection de publications peut toujours être effectué à l'aide du chargement hâtif (consultez *chargement hâtif* ci-dessus) ou de la méthode Load (voir *chargement explicite* ci-dessous).

### Désactiver le chargement différé pour toutes les entités

Le chargement différé peut être désactivé pour toutes les entités du contexte en définissant un indicateur sur la propriété de configuration. Par exemple :

```
public class BloggingContext : DbContext
{
    public BloggingContext()
    {
        this.Configuration.LazyLoadingEnabled = false;
    }
}
```

Le chargement d'entités associées peut toujours être effectué à l'aide du chargement hâtif (consultez *chargement hâtif* ci-dessus) ou de la méthode Load (voir *chargement explicite* ci-dessous).

## Chargement explicite

Même si le chargement différé est désactivé, il est toujours possible de charger tardivement les entités associées, mais cela doit être effectué avec un appel explicite. Pour ce faire, vous utilisez la méthode Load sur l'entrée de l'entité associée. Par exemple :

```
using (var context = new BloggingContext())
{
    var post = context.Posts.Find(2);

    // Load the blog related to a given post.
    context.Entry(post).Reference(p => p.Blog).Load();

    // Load the blog related to a given post using a string.
    context.Entry(post).Reference("Blog").Load();

    var blog = context.Blogs.Find(1);

    // Load the posts related to a given blog.
    context.Entry(blog).Collection(p => p.Posts).Load();

    // Load the posts related to a given blog
    // using a string to specify the relationship.
    context.Entry(blog).Collection("Posts").Load();
}
```

#### NOTE

La méthode de référence doit être utilisée lorsqu'une entité a une propriété de navigation vers une autre entité unique. En revanche, la méthode de collection doit être utilisée lorsqu'une entité a une propriété de navigation vers une collection d'autres entités.

### Application de filtres lors du chargement explicite d'entités associées

La méthode de requête fournit l'accès à la requête sous-jacente que Entity Framework utilisera lors du chargement des entités associées. Vous pouvez ensuite utiliser LINQ pour appliquer des filtres à la requête avant de l'exécuter avec un appel à une méthode d'extension LINQ comme `ToList`, `Load`, etc. La méthode de requête peut être utilisée avec les propriétés de navigation de référence et de collection, mais elle est particulièrement utile pour les collections où elle peut être utilisée pour charger uniquement une partie de la collection. Par exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Load the posts with the 'entity-framework' tag related to a given blog.
    context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Where(p => p.Tags.Contains("entity-framework"))
        .Load();

    // Load the posts with the 'entity-framework' tag related to a given blog
    // using a string to specify the relationship.
    context.Entry(blog)
        .Collection("Posts")
        .Query()
        .Where(p => p.Tags.Contains("entity-framework"))
        .Load();
}
```

Lors de l'utilisation de la méthode de requête, il est généralement préférable de désactiver le chargement différé pour la propriété de navigation. En effet, dans le cas contraire, l'ensemble de la collection peut être chargé automatiquement par le mécanisme de chargement différé avant ou après l'exécution de la requête filtrée.

#### NOTE

Alors que la relation peut être spécifiée en tant que chaîne au lieu d'une expression lambda, le `IQueryable` retourné n'est pas générique lorsqu'une chaîne est utilisée et la méthode de cast est généralement nécessaire avant toute opération utile.

### Utilisation de Query pour compter les entités associées sans les charger

Il est parfois utile de savoir combien d'entités sont liées à une autre entité de la base de données sans réellement avoir à charger toutes ces entités. Pour ce faire, vous pouvez utiliser la méthode `Query` avec la méthode `Count` LINQ. Par exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Count how many posts the blog has.
    var postCount = context.Entry(blog)
        .Collection(b => b.Posts)
        .Query()
        .Count();

}
```

# Enregistrement de données avec Entity Framework 6

13/09/2018 • 2 minutes to read

Cette section vous donne des informations sur les fonctionnalités de suivi de changements d'Entity Framework et sur ce que qui se produit quand vous appelez `SaveChanges` pour conserver les changements des objets dans la base de données.

# Déetecter des modifications automatique

13/09/2018 • 3 minutes to read

Lors de l'utilisation de la plupart des entités POCO la détermination de la manière dont une entité a changé (et par conséquent les mises à jour doivent être envoyées à la base de données) est gérée par l'algorithme de détecter les modifications. Détecter works de modifications en détectant les différences entre les valeurs de propriété actuelles de l'entité et les valeurs de propriété d'origine qui sont stockés dans un instantané lorsque l'entité a été interrogée ou attachée. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

Par défaut, Entity Framework effectue automatiquement les modifications de détecter lorsque les méthodes suivantes sont appelées :

- DbSet.Find
- DbSet.Local
- DbSet.Add
- DbSet.AddRange
- DbSet.Remove
- DbSet.RemoveRange
- DbSet.Attach
- DbContext.SaveChanges
- DbContext.GetValidationErrors
- DbContext.Entry
- DbChangeTracker.Entries

## La désactivation de la détection automatique des modifications

Si vous suivez un grand nombre d'entités dans votre contexte et que vous appelez une de ces méthodes plusieurs fois dans une boucle, vous pouvez obtenir des améliorations significatives des performances en désactivant la détection des modifications apportées pendant la durée de la boucle. Exemple :

```
using (var context = new BloggingContext())
{
    try
    {
        context.Configuration.AutoDetectChangesEnabled = false;

        // Make many calls in a loop
        foreach (var blog in aLotOfBlogs)
        {
            context.Blogs.Add(blog);
        }
    }
    finally
    {
        context.Configuration.AutoDetectChangesEnabled = true;
    }
}
```

N'oubliez pas de réactiver la détection des modifications apportées après la boucle, nous avons utilisé un bloc try/finally pour vous assurer qu'il est toujours réactivé même si le code dans la boucle lève une exception.

Une alternative à la désactivation et réactivation de l'est de laisser la détection automatique des modifications mis hors tension à toutes les heures et un contexte d'appel. ChangeTracker.DetectChanges explicitement ou utilisez l'assidûment les proxys de suivi. Ces deux options avancées et peut facilement introduire des bogues subtils dans votre application donc de les utiliser avec précaution.

Si vous avez besoin ajouter ou supprimer un grand nombre d'objets à partir d'un contexte, envisagez d'utiliser DbSet.AddRange et DbSet.RemoveRange. Cette méthode détecte automatiquement les modifications qu'une seule fois après avoir effectué les opérations d'ajout / suppression.

# Utilisation des États des entités

08/12/2018 • 10 minutes to read

Cette section explique comment ajouter et joindre des entités à un contexte et comment Entity Framework traite ces pendant SaveChanges. Entity Framework effectue le suivi de l'état des entités pendant qu'ils sont connectés à un contexte, mais dans les scénarios déconnectés ou applications multicouches vous pouvez laisser EF savoir quel état vos entités doit être dans. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

## États des entités et SaveChanges

Une entité peut avoir l'un des cinq états tel que défini par l'énumération de l'état d'entité. Ces États sont :

- Ajout : l'entité est suivie par le contexte, mais n'existe pas encore dans la base de données
- Unchanged : l'entité est suivie par le contexte et existe dans la base de données et ses valeurs de propriété n'ont pas changé à partir des valeurs dans la base de données
- Modifié : l'entité est suivie par le contexte et existe dans la base de données, et tout ou partie de ses valeurs de propriété ont été modifiées.
- Supprimé : l'entité est suivie par le contexte et existe dans la base de données, mais a été marquée pour suppression à partir de la base de données lors du prochain qu'appel de SaveChanges
- Détaché : l'entité n'est pas suivie par le contexte

SaveChanges effectue des opérations différentes pour les entités dans différents états :

- Entités inchangées ne sont pas touchées par SaveChanges. Mises à jour ne sont pas envoyées à la base de données pour les entités dans un état Unchanged.
- Ajouté des entités sont insérées dans la base de données et devient inchangées lorsque SaveChanges retourne.
- Entités modifiées sont mises à jour dans la base de données et devenir inchangées lorsque SaveChanges retourne.
- Entités supprimées sont supprimées de la base de données et sont ensuite détachées à partir du contexte.

Les exemples suivants montrent des façons dans lequel l'état d'une entité ou un graphique d'entité peut être modifié.

## Ajout d'une nouvelle entité au contexte

Une nouvelle entité peut être ajoutée au contexte en appelant la méthode Add sur DbSet. Cela place l'entité dans l'état Added, ce qui signifie qu'elle sera insérée dans la base de données la prochaine fois que SaveChanges est appelée. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = new Blog { Name = "ADO.NET Blog" };
    context.Blogs.Add(blog);
    context.SaveChanges();
}
```

Une autre façon d'ajouter une nouvelle entité au contexte consiste à modifier son état Added. Exemple :

```

using (var context = new BloggingContext())
{
    var blog = new Blog { Name = "ADO.NET Blog" };
    context.Entry(blog).State = EntityState.Added;
    context.SaveChanges();
}

```

Enfin, vous pouvez ajouter une nouvelle entité au contexte par elle liant à une autre entité qui est déjà suivie. Il peut s'agir en ajoutant la nouvelle entité à la propriété de navigation de collection d'une autre entité ou en définissant une propriété de navigation de référence d'une autre entité pour pointer vers la nouvelle entité.

Exemple :

```

using (var context = new BloggingContext())
{
    // Add a new User by setting a reference from a tracked Blog
    var blog = context.Blogs.Find(1);
    blog.Owner = new User { UserName = "johndoe1987" };

    // Add a new Post by adding to the collection of a tracked Blog
    blog.Posts.Add(new Post { Name = "How to Add Entities" });

    context.SaveChanges();
}

```

Notez que, pour tous ces exemples, si l'entité en cours d'ajout a des références à d'autres entités qui ne sont pas encore suivies ensuite ces nouvelles entités est également ajouté au contexte et sera insérée dans la base de données la prochaine fois que SaveChanges est appelée.

## Attachement d'une entité existante au contexte

Si vous avez une entité dont vous savez déjà existe dans la base de données mais qui n'est pas actuellement en cours suivi par le contexte vous pouvez indiquer le contexte pour effectuer le suivi de l'entité à l'aide de la méthode d'attachement sur DbSet. L'entité sera à l'état inchangé dans le contexte. Exemple :

```

var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Blogs.Attach(existingBlog);

    // Do some more work...

    context.SaveChanges();
}

```

Notez qu'aucune modification n'est apportées à la base de données si SaveChanges est appelée sans effectuer toute autre manipulation de l'entité attachée. Il s'agit, car l'entité est à l'état inchangé.

Une autre façon d'attacher une entité existante au contexte consiste à modifier son état à « Unchanged ». Exemple :

```

var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Entry(existingBlog).State = EntityState.Unchanged;

    // Do some more work...

    context.SaveChanges();
}

```

Notez que pour ces deux exemples si l'entité qui est attachée a des références à d'autres entités qui ne sont pas encore suivies ensuite ces nouvelles entités seront également attachés au contexte à l'état inchangé.

## Attachement d'un existant mais l'entité modifiée au contexte

Si vous avez une entité dont vous savez déjà existe dans la base de données, mais pour les modifications qui ont été apportées vous pouvez indiquer le contexte pour attacher l'entité et de définir son état sur Modified. Exemple :

```

var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Entry(existingBlog).State = EntityState.Modified;

    // Do some more work...

    context.SaveChanges();
}

```

Lorsque vous modifiez l'état Modified toutes les propriétés de l'entité seront marquées comme modifiée et toutes les valeurs de propriété seront envoyés à la base de données lorsque SaveChanges est appelée.

Notez que si l'entité qui est attachée a des références à d'autres entités qui ne sont pas encore suivies, ces nouvelles entités sera attaché au contexte dans un état Unchanged, celles-ci ne sont pas automatiquement converties Modified. Si vous disposez de plusieurs entités qui doivent être marqués Modified vous devez définir l'état pour chacune de ces entités individuellement.

## Modification de l'état d'une entité de suivi

Vous pouvez modifier l'état d'une entité est déjà suivie en définissant la propriété d'état sur son entrée. Exemple :

```

var existingBlog = new Blog { BlogId = 1, Name = "ADO.NET Blog" };

using (var context = new BloggingContext())
{
    context.Blogs.Attach(existingBlog);
    context.Entry(existingBlog).State = EntityState.Unchanged;

    // Do some more work...

    context.SaveChanges();
}

```

Notez que l'appel Add ou attacher pour une entité qui est déjà suivie peut également servir pour modifier l'état d'entité. Par exemple, attacher l'appel d'une entité qui se trouve actuellement dans l'état Added modifie son état à « Unchanged ».

## Insérer ou mettre à jour de modèle

Un modèle courant pour certaines applications consiste à ajouter une entité en tant que nouveau (ce qui entraîne l'insertion d'une base de données) ou attacher une entité comme existante et la marquer comme modifié (ce qui entraîne une mise à jour de la base de données) en fonction de la valeur de la clé primaire. Par exemple, lors de l'utilisation de clés primaires de base de données générée entier, il est courant de traiter une entité avec une clé en tant que nouvelle de zéro et une entité avec une clé différente de zéro comme existant. Ce modèle peut être obtenu en définissant l'état d'entité selon une vérification de la valeur de clé primaire. Exemple :

```
public void InsertOrUpdate(Blog blog)
{
    using (var context = new BloggingContext())
    {
        context.Entry(blog).State = blog.BlogId == 0 ?
            EntityState.Added :
            EntityState.Modified;

        context.SaveChanges();
    }
}
```

Notez que lorsque vous modifiez l'état Modified toutes les propriétés de l'entité seront marquées comme modifiée et toutes les valeurs de propriété seront envoyées à la base de données lorsque SaveChanges est appelée.

# Utilisation des valeurs de propriété

11/10/2019 • 17 minutes to read

Dans la plupart des Entity Framework effectuent le suivi de l'État, les valeurs d'origine et les valeurs actuelles des propriétés de vos instances d'entité. Toutefois, il peut y avoir certains cas, tels que des scénarios déconnectés, où vous souhaitez afficher ou manipuler les informations d'EF sur les propriétés. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

Entity Framework effectue le suivi de deux valeurs pour chaque propriété d'une entité suivie. La valeur actuelle est, comme le nom l'indique, la valeur actuelle de la propriété dans l'entité. La valeur d'origine est la valeur que la propriété avait lorsque l'entité était interrogée à partir de la base de données ou attachée au contexte.

Il existe deux mécanismes généraux pour l'utilisation des valeurs de propriété :

- La valeur d'une propriété unique peut être obtenue de manière fortement typée à l'aide de la méthode de propriété.
- Les valeurs de toutes les propriétés d'une entité peuvent être lues dans un objet `DbPropertyValues`.  
`DbPropertyValues` agit ensuite comme un objet de type dictionnaire pour permettre la lecture et la définition des valeurs de propriété. Les valeurs d'un objet `DbPropertyValues` peuvent être définies à partir de valeurs dans un autre objet `DbPropertyValues` ou à partir de valeurs d'un autre objet, comme une autre copie de l'entité ou un objet de transfert de données simple (DTO).

Les sections ci-dessous présentent des exemples d'utilisation des deux mécanismes ci-dessus.

## Obtention et définition de la valeur actuelle ou d'origine d'une propriété individuelle

L'exemple ci-dessous montre comment la valeur actuelle d'une propriété peut être lue, puis définie sur une nouvelle valeur :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(3);

    // Read the current value of the Name property
    string currentName1 = context.Entry(blog).Property(u => u.Name).CurrentValue;

    // Set the Name property to a new value
    context.Entry(blog).Property(u => u.Name).CurrentValue = "My Fancy Blog";

    // Read the current value of the Name property using a string for the property name
    object currentName2 = context.Entry(blog).Property("Name").CurrentValue;

    // Set the Name property to a new value using a string for the property name
    context.Entry(blog).Property("Name").CurrentValue = "My Boring Blog";
}
```

Utilisez la propriété `OriginalValue` à la place de la propriété `CurrentValue` pour lire ou définir la valeur d'origine.

Notez que la valeur renournée est de type « Object » lorsqu'une chaîne est utilisée pour spécifier le nom de la propriété. En revanche, la valeur renournée est fortement typée si une expression lambda est utilisée.

La définition de la valeur de propriété comme celle-ci marquera uniquement la propriété comme modifiée si la nouvelle valeur est différente de l'ancienne valeur.

Quand une valeur de propriété est définie de cette façon, la modification est détectée automatiquement, même si AutoDetectChanges est désactivé.

## Obtention et définition de la valeur actuelle d'une propriété non mappée

La valeur actuelle d'une propriété qui n'est pas mappée à la base de données peut également être lue. Un exemple de propriété non mappée peut être une propriété RssLink sur le blog. Cette valeur peut être calculée en fonction du BlogId et ne doit donc pas être stockée dans la base de données. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    // Read the current value of an unmapped property
    var rssLink = context.Entry(blog).Property(p => p.RssLink).CurrentValue;

    // Use a string to specify the property name
    var rssLinkAgain = context.Entry(blog).Property("RssLink").CurrentValue;
}
```

La valeur actuelle peut également être définie si la propriété expose un accesseur Set.

La lecture des valeurs des propriétés non mappées est utile lors de l'exécution de Entity Framework validation des propriétés non mappées. Pour la même raison, les valeurs actuelles peuvent être lues et définies pour les propriétés des entités qui ne sont pas actuellement suivies par le contexte. Exemple :

```
using (var context = new BloggingContext())
{
    // Create an entity that is not being tracked
    var blog = new Blog { Name = "ADO.NET Blog" };

    // Read and set the current value of Name as before
    var currentName1 = context.Entry(blog).Property(u => u.Name).CurrentValue;
    context.Entry(blog).Property(u => u.Name).CurrentValue = "My Fancy Blog";
    var currentName2 = context.Entry(blog).Property("Name").CurrentValue;
    context.Entry(blog).Property("Name").CurrentValue = "My Boring Blog";
}
```

Notez que les valeurs d'origine ne sont pas disponibles pour les propriétés non mappées ou pour les propriétés des entités qui ne font pas l'objet d'un suivi par le contexte.

## Vérification de la présence d'une propriété comme étant modifiée

L'exemple ci-dessous montre comment vérifier si une propriété individuelle est marquée comme modifiée :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    var nameIsModified1 = context.Entry(blog).Property(u => u.Name).IsModified;

    // Use a string for the property name
    var nameIsModified2 = context.Entry(blog).Property("Name").IsModified;
}
```

Les valeurs des propriétés modifiées sont envoyées en tant que mises à jour de la base de données lorsque SaveChanges est appelé.

## Marquage d'une propriété comme modifiée

L'exemple ci-dessous montre comment forcer une propriété individuelle à être marquée comme modifiée :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    context.Entry(blog).Property(u => u.Name).IsModified = true;

    // Use a string for the property name
    context.Entry(blog).Property("Name").IsModified = true;
}
```

Le marquage d'une propriété comme modifiée force l'envoi d'une mise à jour à la base de données pour la propriété lorsque SaveChanges est appelé, même si la valeur actuelle de la propriété est identique à sa valeur d'origine.

Il n'est pas possible actuellement de réinitialiser une propriété individuelle pour qu'elle ne soit pas modifiée une fois qu'elle a été marquée comme modifiée. C'est ce que nous envisageons de prendre en charge dans une prochaine version.

## Lecture des valeurs actuelles, d'origine et de la base de données pour toutes les propriétés d'une entité

L'exemple ci-dessous montre comment lire les valeurs actuelles, les valeurs d'origine et les valeurs réellement dans la base de données pour toutes les propriétés mappées d'une entité.

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    // Make a modification to Name in the tracked entity
    blog.Name = "My Cool Blog";

    // Make a modification to Name in the database
    context.Database.SqlCommand("update dbo.Blogs set Name = 'My Boring Blog' where Id = 1");

    // Print out current, original, and database values
    Console.WriteLine("Current values:");
    PrintValues(context.Entry(blog).CurrentValues);

    Console.WriteLine("\nOriginal values:");
    PrintValues(context.Entry(blog).OriginalValues);

    Console.WriteLine("\nDatabase values:");
    PrintValues(context.Entry(blog).GetDatabaseValues());
}

public static void PrintValues(DbPropertyValues values)
{
    foreach (var propertyName in values.PropertyNames)
    {
        Console.WriteLine("Property {0} has value {1}",
                          propertyName, values[propertyName]);
    }
}
```

Les valeurs actuelles sont les valeurs que les propriétés de l'entité contiennent actuellement. Les valeurs d'origine sont les valeurs qui ont été lues à partir de la base de données lors de l'interrogation de l'entité. Les valeurs de

base de données sont les valeurs telles qu'elles sont actuellement stockées dans la base de données. L'obtention des valeurs de la base de données est utile lorsque les valeurs de la base de données peuvent avoir changé depuis l'interrogation de l'entité, par exemple lorsqu'une modification simultanée de la base de données a été effectuée par un autre utilisateur.

## Définition des valeurs actuelles ou d'origine à partir d'un autre objet

Les valeurs actuelles ou d'origine d'une entité suivie peuvent être mises à jour en copiant des valeurs à partir d'un autre objet. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    var coolBlog = new Blog { Id = 1, Name = "My Cool Blog" };
    var boringBlog = new BlogDto { Id = 1, Name = "My Boring Blog" };

    // Change the current and original values by copying the values from other objects
    var entry = context.Entry(blog);
    entry.CurrentValues.SetValues(coolBlog);
    entry.OriginalValues.SetValues(boringBlog);

    // Print out current and original values
    Console.WriteLine("Current values:");
    PrintValues(entry.CurrentValues);

    Console.WriteLine("\nOriginal values:");
    PrintValues(entry.OriginalValues);
}

public class BlogDto
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

L'exécution du code ci-dessus imprime :

```
Current values:
Property Id has value 1
Property Name has value My Cool Blog

Original values:
Property Id has value 1
Property Name has value My Boring Blog
```

Cette technique est parfois utilisée lors de la mise à jour d'une entité avec des valeurs obtenues à partir d'un appel de service ou d'un client dans une application multiniveau. Notez que l'objet utilisé n'a pas besoin d'être du même type que l'entité, à condition qu'il possède des propriétés dont les noms correspondent à ceux de l'entité. Dans l'exemple ci-dessus, une instance de BlogDTO est utilisée pour mettre à jour les valeurs d'origine.

Notez que seules les propriétés qui sont définies sur des valeurs différentes lorsqu'elles sont copiées à partir de l'autre objet sont marquées comme modifiées.

## Définition des valeurs actuelles ou d'origine à partir d'un dictionnaire

Les valeurs actuelles ou d'origine d'une entité suivie peuvent être mises à jour en copiant des valeurs à partir d'un dictionnaire ou d'une autre structure de données. Exemple :

```

using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    var newValues = new Dictionary<string, object>
    {
        { "Name", "The New ADO.NET Blog" },
        { "Url", "blogs.msdn.com/adonet" },
    };

    var currentValues = context.Entry(blog).CurrentValues;

    foreach (var propertyName in newValues.Keys)
    {
        currentValues[propertyName] = newValues[propertyName];
    }

    PrintValues(currentValues);
}

```

Utilisez la propriété `OriginalValues` au lieu de la propriété `CurrentValues` pour définir les valeurs d'origine.

## Définition des valeurs actuelles ou d'origine à partir d'un dictionnaire à l'aide de la propriété

Une alternative à l'utilisation de `CurrentValues` ou `OriginalValues`, comme indiqué ci-dessus, consiste à utiliser la méthode `Property` pour définir la valeur de chaque propriété. Cela peut être préférable lorsque vous devez définir les valeurs des propriétés complexes. Exemple :

```

using (var context = new BloggingContext())
{
    var user = context.Users.Find("johndoe1987");

    var newValues = new Dictionary<string, object>
    {
        { "Name", "John Doe" },
        { "Location.City", "Redmond" },
        { "Location.State.Name", "Washington" },
        { "Location.State.Code", "WA" },
    };

    var entry = context.Entry(user);

    foreach (var propertyName in newValues.Keys)
    {
        entry.Property(propertyName).CurrentValue = newValues[propertyName];
    }
}

```

Dans l'exemple ci-dessus, les propriétés complexes sont accessibles à l'aide de noms en pointillés. Pour d'autres méthodes d'accès aux propriétés complexes, consultez les deux sections plus loin dans cette rubrique, en particulier sur les propriétés complexes.

## Création d'un objet cloné contenant les valeurs actuelles, d'origine ou de base de données

L'objet `DbPropertyValues` retourné par `CurrentValues`, `OriginalValues` ou `GetDatabaseValues` peut être utilisé pour créer un clone de l'entité. Ce clone contient les valeurs de propriété de l'objet `DbPropertyValues` utilisé pour le créer. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);

    var clonedBlog = context.Entry(blog).GetDatabaseValues().ToObject();
}
```

Notez que l'objet retourné n'est pas l'entité et qu'il n'est pas suivi par le contexte. L'objet retourné n'a pas non plus de relations définies sur d'autres objets.

L'objet cloné peut être utile pour résoudre les problèmes liés aux mises à jour simultanées de la base de données, notamment lorsqu'une interface utilisateur qui implique la liaison de données à des objets d'un certain type est utilisée.

## Obtention et définition des valeurs actuelles ou d'origine des propriétés complexes

La valeur d'un objet complexe entier peut être lue et définie à l'aide de la méthode `Property`, tout comme pour une propriété primitive. En outre, vous pouvez accéder à l'objet complexe et lire ou définir les propriétés de cet objet, ou même un objet imbriqué. Voici quelques exemples :

```

using (var context = new BloggingContext())
{
    var user = context.Users.Find("johndoe1987");

    // Get the Location complex object
    var location = context.Entry(user)
        .Property(u => u.Location)
        .CurrentValue;

    // Get the nested State complex object using chained calls
    var state1 = context.Entry(user)
        .ComplexProperty(u => u.Location)
        .Property(l => l.State)
        .CurrentValue;

    // Get the nested State complex object using a single lambda expression
    var state2 = context.Entry(user)
        .Property(u => u.Location.State)
        .CurrentValue;

    // Get the nested State complex object using a dotted string
    var state3 = context.Entry(user)
        .Property("Location.State")
        .CurrentValue;

    // Get the value of the Name property on the nested State complex object using chained calls
    var name1 = context.Entry(user)
        .ComplexProperty(u => u.Location)
        .ComplexProperty(l => l.State)
        .Property(s => s.Name)
        .CurrentValue;

    // Get the value of the Name property on the nested State complex object using a single lambda expression
    var name2 = context.Entry(user)
        .Property(u => u.Location.State.Name)
        .CurrentValue;

    // Get the value of the Name property on the nested State complex object using a dotted string
    var name3 = context.Entry(user)
        .Property("Location.State.Name")
        .CurrentValue;
}

```

Utilisez la propriété `OriginalValue` à la place de la propriété `CurrentValue` pour récupérer ou définir une valeur d'origine.

Notez que la propriété ou la méthode `ComplexProperty` peut être utilisée pour accéder à une propriété complexe. Toutefois, la méthode `ComplexProperty` doit être utilisée si vous souhaitez accéder à l'objet complexe avec des appels de propriété ou de `ComplexProperty` supplémentaires.

## Utilisation de `DbPropertyValues` pour accéder à des propriétés complexes

Quand vous utilisez `CurrentValues`, `OriginalValues` ou `GetDatabaseValues` pour obtenir toutes les valeurs actuelles, d'origine ou de base de données pour une entité, les valeurs de toutes les propriétés complexes sont retournées en tant qu'objets `DbPropertyValues` imbriqués. Ces objets imbriqués peuvent ensuite être utilisés pour récupérer les valeurs de l'objet complexe. Par exemple, la méthode suivante affiche les valeurs de toutes les propriétés, y compris les valeurs des propriétés complexes et des propriétés complexes imbriquées.

```
public static void WritePropertyValues(string parentPropertyName, DbPropertyValues propertyValues)
{
    foreach (var propertyName in propertyValues.PropertyNames)
    {
        var nestedValues = propertyValues[propertyName] as DbPropertyValues;
        if (nestedValues != null)
        {
            WritePropertyValues(parentPropertyName + propertyName + ".", nestedValues);
        }
        else
        {
            Console.WriteLine("Property {0}{1} has value {2}",
                parentPropertyName, propertyName,
                propertyValues[propertyName]);
        }
    }
}
```

Pour imprimer toutes les valeurs de propriété actuelles, la méthode est appelée comme suit :

```
using (var context = new BloggingContext())
{
    var user = context.Users.Find("johndoe1987");

    WritePropertyValues("", context.Entry(user).CurrentValues);
}
```

# Gestion de conflits d'accès concurrentiel

13/09/2018 • 9 minutes to read

Tentative d'enregistrement de votre entité dans la base de données dans l'espoir que les données n'a pas changé depuis l'entité a été chargée optimiste implique l'accès concurrentiel optimiste. S'il s'avère que les données ont changé, une exception est levée et vous devez résoudre le conflit avant d'enregistrer à nouveau. Cette rubrique explique comment gérer ces exceptions dans Entity Framework. Les techniques présentées dans cette rubrique s'appliquent également aux modèles créés avec Code First et EF Designer.

Cette publication n'est pas l'emplacement approprié pour obtenir une présentation complète de l'accès concurrentiel optimiste. Les sections ci-dessous supposent une connaissance de la résolution d'accès concurrentiel et affichent des modèles pour les tâches courantes.

Bon nombre de ces modèles font utiliser les sujets abordés dans [travaillez avec des valeurs de propriété](#).

Résolution des problèmes d'accès concurrentiel lorsque vous utilisez des associations indépendantes (où la clé étrangère n'est pas mappée à une propriété de votre entité) est beaucoup plus difficile que lorsque vous utilisez des associations de clé étrangère. Par conséquent, si vous vous apprêtez à effectuer la résolution d'accès concurrentiel dans votre application, il est conseillé de toujours correspondre les clés étrangères dans vos entités. Tous les exemples ci-dessous partent du principe que vous utilisez des associations de clé étrangère.

Une `DbUpdateConcurrencyException` est levée par `SaveChanges` lorsqu'une exception d'accès concurrentiel optimiste est détectée lors de la tentative d'enregistrement d'une entité qui utilise les associations de clé étrangère.

## Résolution des exceptions d'accès concurrentiel optimiste avec recharger (wins de la base de données)

La méthode `Reload` peut être utilisée pour remplacer les valeurs actuelles de l'entité avec les valeurs maintenant dans la base de données. L'entité est ensuite généralement envoyée à l'utilisateur dans une certaine forme et ils doivent essayer de la rendre de nouveau leurs modifications et enregistrez à nouveau. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    blog.Name = "The New ADO.NET Blog";

    bool saveFailed;
    do
    {
        saveFailed = false;

        try
        {
            context.SaveChanges();
        }
        catch (DbUpdateConcurrencyException ex)
        {
            saveFailed = true;

            // Update the values of the entity that failed to save from the store
            ex.Entries.Single().Reload();
        }
    } while (saveFailed);
}
```

Un bon moyen de simuler une exception d'accès concurrentiel consiste à définir un point d'arrêt sur l'appel de SaveChanges, puis modifiez une entité qui est enregistrée dans la base de données à l'aide d'un autre outil tel que SQL Management Studio. Vous pouvez également insérer une ligne avant SaveChanges pour mettre à jour de la base de données directement à l'aide de SqlCommand. Exemple :

```
context.Database.SqlCommand(  
    "UPDATE dbo.Blogs SET Name = 'Another Name' WHERE BlogId = 1");
```

La méthode entrées sur DbUpdateConcurrencyException retourne les instances DbEntityEntry pour les entités qui n'a pas pu mettre à jour. (Cette propriété actuellement retourne toujours une valeur unique pour les problèmes d'accès concurrentiel. Elle peut retourner plusieurs valeurs pour les exceptions de mise à jour générale.) Une alternative dans certains cas peut consister à obtenir les entrées pour toutes les entités devant être rechargées à partir de la base de données et appeler Recharge pour chacun d'eux.

## Résolution des exceptions d'accès concurrentiel optimiste en tant que priorité au client

L'exemple ci-dessus qui utilise le rechargement est parfois appelé wins de la base de données ou priorité au magasin, car les valeurs de l'entité sont remplacées par les valeurs de la base de données. Vous pouvez parfois souhaiter l'inverse, remplacer les valeurs dans la base de données avec les valeurs actuellement dans l'entité. Cela est parfois appelé priorité au client et peut être effectuée en obtenant les valeurs actuelles de la base de données et de les définir en tant que les valeurs d'origine pour l'entité. (Consultez [travaillez avec des valeurs de propriété](#) pour plus d'informations sur les valeurs actuelles et d'origine.) Exemple :

```
using (var context = new BloggingContext())  
{  
    var blog = context.Blogs.Find(1);  
    blog.Name = "The New ADO.NET Blog";  
  
    bool saveFailed;  
    do  
    {  
        saveFailed = false;  
        try  
        {  
            context.SaveChanges();  
        }  
        catch (DbUpdateConcurrencyException ex)  
        {  
            saveFailed = true;  
  
            // Update original values from the database  
            var entry = ex.Entries.Single();  
            entry.OriginalValues.SetValues(entry.GetDatabaseValues());  
        }  
  
        } while (saveFailed);  
}
```

## Résolution personnalisée des exceptions d'accès concurrentiel optimiste

Vous pouvez être amené à combiner les valeurs actuellement dans la base de données avec les valeurs actuellement dans l'entité. Cela nécessite généralement une interaction utilisateur ou de la logique personnalisée. Par exemple, vous pouvez présenter un formulaire à l'utilisateur contenant les valeurs actuelles, les valeurs dans la base de données, et une valeur par défaut définie de valeurs résolues. L'utilisateur est ensuite modifier les valeurs

résolues en fonction des besoins, et il serait ces valeurs résolues enregistrées dans la base de données. Cela est possible en utilisant les objets DbPropertyValues retourné par CurrentValues et GetDatabaseValues sur l'entrée de l'entité. Exemple :

```
using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    blog.Name = "The New ADO.NET Blog";

    bool saveFailed;
    do
    {
        saveFailed = false;
        try
        {
            context.SaveChanges();
        }
        catch (DbUpdateConcurrencyException ex)
        {
            saveFailed = true;

            // Get the current entity values and the values in the database
            var entry = ex.Entries.Single();
            var currentValues = entry.CurrentValues;
            var databaseValues = entry.GetDatabaseValues();

            // Choose an initial set of resolved values. In this case we
            // make the default be the values currently in the database.
            var resolvedValues = databaseValues.Clone();

            // Have the user choose what the resolved values should be
            HaveUserResolveConcurrency(currentValues, databaseValues, resolvedValues);

            // Update the original values with the database values and
            // the current values with whatever the user choose.
            entry.OriginalValues.SetValues(databaseValues);
            entry.CurrentValues.SetValues(resolvedValues);
        }
    } while (saveFailed);
}

public void HaveUserResolveConcurrency(DbPropertyValues currentValues,
                                         DbPropertyValues databaseValues,
                                         DbPropertyValues resolvedValues)
{
    // Show the current, database, and resolved values to the user and have
    // them edit the resolved values to get the correct resolution.
}
```

## Résolution personnalisée des exceptions d'accès concurrentiel optimiste à l'aide d'objets

Le code ci-dessus utilise des instances de DbPropertyValues pour passer des cours, base de données et valeurs résolues. Parfois, il peut être plus facile à utiliser des instances de votre type d'entité pour cela. Cela est possible à l'aide de méthodes ToObject et SetValues de DbPropertyValues. Exemple :

```

using (var context = new BloggingContext())
{
    var blog = context.Blogs.Find(1);
    blog.Name = "The New ADO.NET Blog";

    bool saveFailed;
    do
    {
        saveFailed = false;
        try
        {
            context.SaveChanges();
        }
        catch (DbUpdateConcurrencyException ex)
        {
            saveFailed = true;

            // Get the current entity values and the values in the database
            // as instances of the entity type
            var entry = ex.Entries.Single();
            var databaseValues = entry.GetDatabaseValues();
            var databaseValuesAsBlog = (Blog)databaseValues.ToObject();

            // Choose an initial set of resolved values. In this case we
            // make the default be the values currently in the database.
            var resolvedValuesAsBlog = (Blog)databaseValues.ToObject();

            // Have the user choose what the resolved values should be
            HaveUserResolveConcurrency((Blog)entry.Entity,
                databaseValuesAsBlog,
                resolvedValuesAsBlog);

            // Update the original values with the database values and
            // the current values with whatever the user choose.
            entry.OriginalValues.SetValues(databaseValues);
            entry.CurrentValues.SetValues(resolvedValuesAsBlog);
        }
    } while (saveFailed);
}

public void HaveUserResolveConcurrency(Blog entity,
    Blog databaseValues,
    Blog resolvedValues)
{
    // Show the current, database, and resolved values to the user and have
    // them update the resolved values to get the correct resolution.
}

```

# Utilisation des transactions

18/07/2019 • 15 minutes to read

## NOTE

**EF6 et versions ultérieures uniquement** : Les fonctionnalités, les API, etc. décrites dans cette page ont été introduites dans Entity Framework 6. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas.

Ce document décrit l'utilisation des transactions dans EF6, y compris les améliorations que nous avons ajoutées depuis EF5 pour faciliter l'utilisation des transactions.

## Par défaut, EF

Dans toutes les versions de Entity Framework, chaque fois que vous exécutez **SaveChanges ()** pour insérer, mettre à jour ou supprimer sur la base de données, l'infrastructure encapsule cette opération dans une transaction. Cette transaction ne dure que suffisamment longtemps pour exécuter l'opération, puis se termine. Lorsque vous exécutez une autre opération, une nouvelle transaction est démarrée.

À compter **de EF6 Database. ExecuteSqlCommand ()** par défaut, la commande est encapsulée dans une transaction, si celle-ci n'est pas déjà présente. Il existe des surcharges de cette méthode qui vous permettent de substituer ce comportement si vous le souhaitez. En outre, dans EF6, l'exécution de procédures stockées incluses dans le modèle via des API telles que **ObjectContext. ExecuteFunction ()** fait de même (sauf que le comportement par défaut ne peut pas être remplacé).

Dans les deux cas, le niveau d'isolation de la transaction est le niveau d'isolation que le fournisseur de base de données considère comme son paramètre par défaut. Par défaut, par exemple, sur SQL Server il s'agit de READ COMMITTED.

Entity Framework n'encapsule pas les requêtes dans une transaction.

Cette fonctionnalité par défaut convient à un grand nombre d'utilisateurs et, si tel est le cas, il n'est pas nécessaire de faire quelque chose de différent dans EF6. Il vous suffit d'écrire le code comme vous l'avez toujours fait.

Toutefois, certains utilisateurs ont besoin d'un meilleur contrôle sur leurs transactions. Cela est abordé dans les sections suivantes.

## Fonctionnement des API

Avant EF6 Entity Framework insistait sur l'ouverture de la connexion de base de données elle-même (une exception a été levée si une connexion déjà ouverte était passée). Étant donné qu'une transaction ne peut être démarrée que sur une connexion ouverte, cela signifiait que le seul moyen pour un utilisateur d'encapsuler plusieurs opérations dans une seule transaction consistait à utiliser une **TransactionScope** ou à utiliser la propriété **ObjectContext. Connection** et à démarrer appel de **Open ()** et **BeginTransaction ()** directement sur l'objet **EntityConnection** retourné. En outre, les appels d'API qui ont contacté la base de données échouent si vous avez démarré une transaction sur la connexion de base de données sous-jacente.

## NOTE

La limitation de l'acceptation des connexions fermées a été supprimée dans Entity Framework 6. Pour plus d'informations, consultez [gestion des connexions](#).

À compter de EF6, le Framework fournit désormais:

1. **Database.BeginTransaction ()** : Méthode plus simple pour un utilisateur de démarrer et de terminer des transactions dans un DbContext existant, ce qui permet de combiner plusieurs opérations au sein de la même transaction et, par conséquent, toutes validées ou toutes restaurées comme une seule. Il permet également à l'utilisateur de spécifier plus facilement le niveau d'isolation de la transaction.
2. **Database.UseTransaction ()** : qui permet à DbContext d'utiliser une transaction qui a été démarrée en dehors de la Entity Framework.

### Combinaison de plusieurs opérations en une seule transaction dans le même contexte

**Database.BeginTransaction ()** a deux remplacements: un qui accepte un **IsolationLevel** explicite et un qui n'accepte aucun argument et utilise la valeur IsolationLevel par défaut du fournisseur de base de données sous-jacent. Les deux substitutions retournent un objet **DbContextTransaction** qui fournit des méthodes **Commit ()** et **Rollback ()** qui effectuent des validations et des restaurations sur la transaction de magasin sous-jacente.

Le **DbContextTransaction** est destiné à être supprimé une fois qu'il a été validé ou restauré. Un moyen simple d'y parvenir consiste à utiliser (...) {...} syntaxe qui appellera automatiquement **Dispose ()** lorsque le bloc using se termine:

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        static void StartOwnTransactionWithinContext()
        {
            using (var context = new BloggingContext())
            {
                using (var dbContextTransaction = context.Database.BeginTransaction())
                {
                    context.Database.ExecuteSqlCommand(
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'"
                    );

                    var query = context.Posts.Where(p => p.Blog.Rating >= 5);
                    foreach (var post in query)
                    {
                        post.Title += "[Cool Blog]";
                    }

                    context.SaveChanges();

                    dbContextTransaction.Commit();
                }
            }
        }
    }
}
```

#### **NOTE**

Le démarrage d'une transaction nécessite que la connexion à la banque sous-jacente soit ouverte. L'appel de Database.BeginTransaction () entraîne l'ouverture de la connexion si elle n'est pas déjà ouverte. Si DbContextTransaction a ouvert la connexion, il la ferme quand la méthode Dispose () est appelée.

### **Passage d'une transaction existante au contexte**

Parfois, vous souhaitez une transaction qui est encore plus large dans l'étendue et qui comprend des opérations sur la même base de données mais en dehors d'EF. Pour ce faire, vous devez ouvrir la connexion et démarrer la transaction vous-même, puis indiquer à EF a) d'utiliser la connexion de base de données déjà ouverte et b) pour utiliser la transaction existante sur cette connexion.

Pour ce faire, vous devez définir et utiliser un constructeur sur votre classe de contexte qui hérite de l'un des constructeurs DbContext qui prend i) un paramètre de connexion existant et II) la valeur booléenne contextOwnsConnection.

#### **NOTE**

L'indicateur contextOwnsConnection doit avoir la valeur false lorsqu'il est appelé dans ce scénario. Cela est important car il informe Entity Framework qu'il ne doit pas fermer la connexion quand elle l'a fait (par exemple, consultez la ligne 4 ci-dessous):

```
using (var conn = new SqlConnection("..."))
{
    conn.Open();
    using (var context = new BloggingContext(conn, contextOwnsConnection: false))
    {
    }
}
```

En outre, vous devez démarrer la transaction vous-même (y compris la fonction IsolationLevel si vous souhaitez éviter le paramètre par défaut) et laisser Entity Framework savoir qu'une transaction existante a déjà démarré sur la connexion (voir la ligne 33 ci-dessous).

Vous êtes alors libre d'exécuter des opérations de base de données directement sur le SqlConnection lui-même ou sur DbContext. Toutes ces opérations sont exécutées dans une transaction. Vous êtes responsable de la validation ou de la restauration de la transaction et de l'appel de dispose () sur celle-ci, ainsi que pour la fermeture et la suppression de la connexion de base de données. Par exemple :

```

using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        static void UsingExternalTransaction()
        {
            using (var conn = new SqlConnection("..."))
            {
                conn.Open();

                using (var sqlTxn = conn.BeginTransaction(System.Data.IsolationLevel.Snapshot))
                {
                    var sqlCommand = new SqlCommand();
                    sqlCommand.Connection = conn;
                    sqlCommand.Transaction = sqlTxn;
                    sqlCommand.CommandText =
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'";
                    sqlCommand.ExecuteNonQuery();

                    using (var context =
                        new BloggingContext(conn, contextOwnsConnection: false))
                    {
                        context.Database.UseTransaction(sqlTxn);

                        var query = context.Posts.Where(p => p.Blog.Rating >= 5);
                        foreach (var post in query)
                        {
                            post.Title += "[Cool Blog]";
                        }
                        context.SaveChanges();
                    }

                    sqlTxn.Commit();
                }
            }
        }
    }
}

```

## Effacement de la transaction

Vous pouvez passer la valeur null à Database.UseTransaction () pour effacer Entity Framework connaissance de la transaction actuelle. Entity Framework ne valide pas ou ne restaure pas la transaction existante lorsque vous procédez ainsi, utilisez avec précaution et uniquement si vous êtes sûr que c'est ce que vous voulez faire.

## Erreurs dans UseTransaction

Vous verrez une exception de Database.UseTransaction () si vous transmettez une transaction lorsque:

- Entity Framework a déjà une transaction existante
- Entity Framework est déjà en cours d'utilisation dans un TransactionScope
- L'objet de connexion dans la transaction passée a la valeur null. Autrement dit, la transaction n'est pas associée à une connexion. il s'agit généralement d'un signe que la transaction est déjà terminée.
- L'objet de connexion dans la transaction passée ne correspond pas à la connexion de Entity Framework.

# Utilisation de transactions avec d'autres fonctionnalités

Cette section explique en détail comment les transactions ci-dessus interagissent avec:

- Résilience de la connexion
- Méthodes asynchrones
- Transactions TransactionScope

## Résilience des connexions

La nouvelle fonctionnalité de résilience de connexion ne fonctionne pas avec les transactions initiées par l'utilisateur. Pour plus d'informations, consultez [nouvelle tentative de stratégies d'exécution](#).

## Programmation asynchrone

L'approche décrite dans les sections précédentes ne nécessite pas d'options ou de paramètres supplémentaires pour [fonctionner avec la requête asynchrone](#) et les méthodes d'enregistrement. Toutefois, sachez que, en fonction de ce que vous faites dans les méthodes asynchrones, cela peut entraîner des transactions de longue durée, qui peuvent à leur tour entraîner des blocages ou des blocages, ce qui est incorrect pour les performances de l'application globale.

## Transactions TransactionScope

Avant EF6, la méthode recommandée pour fournir des transactions plus étendues consistait à utiliser un objet TransactionScope:

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        static void UsingTransactionScope()
        {
            using (var scope = new TransactionScope(TransactionScopeOption.Required))
            {
                using (var conn = new SqlConnection("..."))
                {
                    conn.Open();

                    var sqlCommand = new SqlCommand();
                    sqlCommand.Connection = conn;
                    sqlCommand.CommandText =
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'";
                    sqlCommand.ExecuteNonQuery();

                    using (var context =
                        new BloggingContext(conn, contextOwnsConnection: false))
                    {
                        var query = context.Posts.Where(p => p.Blog.Rating > 5);
                        foreach (var post in query)
                        {
                            post.Title += "[Cool Blog]";
                        }
                        context.SaveChanges();
                    }
                }

                scope.Complete();
            }
        }
    }
}

```

SqlConnection et Entity Framework utiliseront tous deux la transaction TransactionScope ambiante et seront donc validés ensemble.

À compter de .NET 4.5.1, TransactionScope a été mis à jour pour utiliser également des méthodes asynchrones à l'aide de l'énumération [TransactionScopeAsyncFlowOption](#) :

```

using System.Collections.Generic;
using System.Data.Entity;
using System.Data.SqlClient;
using System.Linq;
using System.Transactions;

namespace TransactionsExamples
{
    class TransactionsExample
    {
        public static void AsyncTransactionScope()
        {
            using (var scope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled))
            {
                using (var conn = new SqlConnection("..."))
                {
                    await conn.OpenAsync();

                    var sqlCommand = new SqlCommand();
                    sqlCommand.Connection = conn;
                    sqlCommand.CommandText =
                        @"UPDATE Blogs SET Rating = 5" +
                        " WHERE Name LIKE '%Entity Framework%'";
                    await sqlCommand.ExecuteNonQueryAsync();

                    using (var context = new BloggingContext(conn, contextOwnsConnection: false))
                    {
                        var query = context.Posts.Where(p => p.Blog.Rating > 5);
                        foreach (var post in query)
                        {
                            post.Title += "[Cool Blog]";
                        }

                        await context.SaveChangesAsync();
                    }
                }
            }
        }
    }
}

```

Il existe toujours des limitations à l'approche TransactionScope:

- Nécessite .NET 4.5.1 ou version ultérieure pour utiliser des méthodes asynchrones.
- Elle ne peut pas être utilisée dans les scénarios de Cloud, sauf si vous êtes sûr de disposer d'une seule connexion (les scénarios Cloud ne prennent pas en charge les transactions distribuées).
- Elle ne peut pas être associée à l'approche Database. UseTransaction () des sections précédentes.
- Elle lèvera des exceptions si vous émettez une DDL et que vous n'avez pas activé les transactions distribuées par le biais du service MSDTC.

Avantages de l'approche TransactionScope:

- Elle met automatiquement à niveau une transaction locale vers une transaction distribuée si vous créez plusieurs connexions à une base de données spécifique ou associez une connexion à une base de données avec une connexion à une base de données différente au sein de la même transaction (Remarque: vous devez avoir le service MSDTC configuré pour autoriser les transactions distribuées pour que cela fonctionne).
- Facilité de codage. Si vous préférez que la transaction soit ambiante et traitée implicitement en arrière-plan plutôt que explicitement sous contrôle, l'approche TransactionScope peut vous convenir mieux.

En résumé, avec les API New Database. BeginTransaction () et Database. UseTransaction () ci-dessus, l'approche TransactionScope n'est plus nécessaire pour la plupart des utilisateurs. Si vous continuez à utiliser TransactionScope, tenez compte des limitations ci-dessus. Nous vous recommandons d'utiliser à la place

l'approche décrite dans les sections précédentes dans la mesure du possible.

# Validation de données

25/10/2019 • 15 minutes to read

## NOTE

**EF 4.1 uniquement** : les fonctionnalités, les API, etc. présentées dans cette page ont été introduites dans Entity Framework 4.1. Si vous utilisez une version antérieure, certaines ou toutes les informations ne s'appliquent pas

Le contenu de cette page est adapté à partir d'un article écrit à l'origine par Julie Lerman (<https://thedatafarm.com>).

Entity Framework fournit une grande variété de fonctionnalités de validation qui peuvent être alimentées par une interface utilisateur pour la validation côté client ou être utilisées pour la validation côté serveur. Lorsque vous utilisez code First, vous pouvez spécifier des validations à l'aide d'une annotation ou de configurations d'API Fluent. Des validations supplémentaires, et plus complexes, peuvent être spécifiées dans le code et fonctionnent si votre modèle se passe d'abord par code First, Model First ou Database.

## Le modèle

Je vais démontrer les validations à l'aide d'une simple paire de classes : blog et billet.

```
public class Blog
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string BloggerName { get; set; }
    public DateTime DateCreated { get; set; }
    public virtual ICollection<Post> Posts { get; set; }
}

public class Post
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime DateCreated { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    public ICollection<Comment> Comments { get; set; }
}
```

## Annotations de données

Code First utilise des annotations de l'assembly `System.ComponentModel.DataAnnotations` comme un moyen de configurer les classes code First. Parmi ces annotations, citons celles qui fournissent des règles telles que les `Required`, les `MaxLength` et les `MinLength`. Plusieurs applications clientes .NET reconnaissent également ces annotations, par exemple, ASP.NET MVC. Vous pouvez réaliser la validation côté client et côté serveur avec ces annotations. Par exemple, vous pouvez forcer la propriété titre du blog à être une propriété obligatoire.

```
[Required]
public string Title { get; set; }
```

Sans aucune modification de code ou de balisage supplémentaire dans l'application, une application MVC existante effectue la validation côté client, même en générant dynamiquement un message à l'aide des noms de propriété et

d'annotation.

## Create

Blog

Title  The Title field is required.

BloggerName  Julie

DateCreated  3/15/2011

Dans la méthode de publication de cette vue Create View, Entity Framework est utilisé pour enregistrer le nouveau blog dans la base de données, mais la validation côté client de MVC est déclenchée avant que l'application n'atteigne ce code.

Toutefois, la validation côté client n'a pas de preuve de puce. Les utilisateurs peuvent avoir un impact sur les fonctionnalités de leur navigateur ou pire encore, un pirate peut utiliser des astuces pour éviter les validations de l'interface utilisateur. Mais Entity Framework reconnaîtront également l'annotation `Required` et la valider.

Un moyen simple de tester cela consiste à désactiver la fonctionnalité de validation côté client de MVC. Vous pouvez le faire dans le fichier Web.config de l'application MVC. La section appSettings a une clé pour ClientValidationEnabled. L'affectation de la valeur false à cette clé empêchera l'interface utilisateur d'effectuer des validations.

```
<appSettings>
    <add key="ClientValidationEnabled" value="false"/>
    ...
</appSettings>
```

Même si la validation côté client est désactivée, vous obtiendrez la même réponse dans votre application. Le message d'erreur « le champ titre est obligatoire » s'affiche comme avant. À ce stade, il s'agit d'un résultat de la validation côté serveur. Entity Framework effectue la validation sur l'annotation de `Required` (avant même que les deux à la fois pour générer une commande `INSERT` à envoyer à la base de données) et retourner l'erreur à MVC qui affichera le message.

## API Fluent

Vous pouvez utiliser l'API Fluent code First au lieu des annotations pour accéder au même côté client & validation côté serveur. Au lieu d'utiliser `Required`, je vous montrerai cela à l'aide d'une validation `MaxLength`.

Les configurations de l'API Fluent sont appliquées comme code First pour générer le modèle à partir des classes. Vous pouvez injecter les configurations en remplaçant la méthode `OnModelCreating` de la classe `DbContext`. Voici une configuration spécifiant que la propriété `BloggerName` ne peut pas comporter plus de 10 caractères.

```

public class BlogContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<Post> Posts { get; set; }
    public DbSet<Comment> Comments { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>().Property(p => p.BloggerName).HasMaxLength(10);
    }
}

```

Les erreurs de validation levées sur la base des configurations de l'API Fluent n'atteignent pas automatiquement l'interface utilisateur, mais vous pouvez la capturer dans le code et y répondre en conséquence.

Voici un code d'erreur de gestion des exceptions dans la classe BlogController de l'application qui capture cette erreur de validation quand Entity Framework tente d'enregistrer un blog avec un BloggerName qui dépasse la valeur maximale de 10 caractères.

```

[HttpPost]
public ActionResult Edit(int id, Blog blog)
{
    try
    {
        db.Entry(blog).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    catch (DbEntityValidationException ex)
    {
        var error = ex.EntityValidationErrors.First().ValidationErrors.First();
        this.ModelState.AddModelError(error.PropertyName, error.ErrorMessage);
        return View();
    }
}

```

La validation n'est pas renvoyée automatiquement dans la vue, c'est pourquoi le code supplémentaire qui utilise `ModelState.AddModelError` est utilisé. Cela permet de s'assurer que les détails de l'erreur sont affichés dans la vue qui utilisera ensuite le `ValidationMessageFor` `HtmlHelper` pour afficher l'erreur.

```
@Html.ValidationMessageFor(model => model.BloggerName)
```

## IValidatableObject

`IValidatableObject` est une interface qui réside dans `System.ComponentModel.DataAnnotations`. Bien qu'il ne fasse pas partie de l'API Entity Framework, vous pouvez toujours l'utiliser pour la validation côté serveur dans vos classes Entity Framework. `IValidatableObject` fournit une méthode `Validate` que Entity Framework appellera au cours de l'opération `SaveChanges` ou vous pouvez appeler vous-même chaque fois que vous souhaitez valider les classes.

Les configurations telles que `Required` et `MaxLength` effectuent la validation sur un champ unique. Dans la méthode `Validate` vous pouvez avoir une logique encore plus complexe, par exemple, en comparant deux champs.

Dans l'exemple suivant, la classe `Blog` a été étendue pour implémenter `IValidatableObject`, puis fournir une règle selon laquelle les `Title` et `BloggerName` ne peuvent pas correspondre.

```

public class Blog : IValidatableObject
{
    public int Id { get; set; }

    [Required]
    public string Title { get; set; }

    public string BloggerName { get; set; }
    public DateTime DateCreated { get; set; }
    public virtual ICollection<Post> Posts { get; set; }

    public IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
    {
        if (Title == BloggerName)
        {
            yield return new ValidationResult(
                "Blog Title cannot match Blogger Name",
                new[] { nameof(Title), nameof(BloggerName) });
        }
    }
}

```

Le constructeur `ValidationResult` prend un `string` qui représente le message d'erreur et un tableau de `string`s qui représentent les noms de membres associés à la validation. Étant donné que cette validation vérifie à la fois les `Title` et les `BloggerName`, les deux noms de propriété sont retournés.

Contrairement à la validation fournie par l'API Fluent, ce résultat de validation est reconnu par la vue et le gestionnaire d'exceptions que j'ai utilisé précédemment pour ajouter l'erreur dans `ModelState` n'est pas nécessaire. Étant donné que je définis les deux noms de propriété dans le `ValidationResult`, les `HtmlHelper` MVC affichent le message d'erreur pour ces deux propriétés.

## Edit

**Blog**

|                                     |  |                                      |
|-------------------------------------|--|--------------------------------------|
| Title                               | <input type="text" value="Julie"/>                 | Blog Title cannot match Blogger Name |
| BloggerName                         | <input type="text" value="Julie"/>                 | Blog Title cannot match Blogger Name |
| DateCreated                         | <input type="text" value="3/11/2011 12:00:00 AM"/> |                                      |
| <input type="button" value="Save"/> |  |                                      |

## DbContext. ValidateEntity

`DbContext` a une méthode substituable appelée `ValidateEntity`. Lorsque vous appelez `SaveChanges`, Entity Framework appelle cette méthode pour chaque entité dans son cache dont l'État n'est pas `Unchanged`. Vous pouvez placer la logique de validation directement ici ou même utiliser cette méthode pour appeler, par exemple, la méthode `Blog.Validate` ajoutée dans la section précédente.

Voici un exemple de `ValidateEntity` remplacement qui valide les nouveaux `Post`s pour s'assurer que le titre de publication n'a pas déjà été utilisé. Il commence par vérifier si l'entité est une publication et que son état est ajouté. Si c'est le cas, il recherche dans la base de données s'il existe déjà une publication avec le même titre. S'il existe déjà une publication existante, une nouvelle `DbEntityValidationResult` est créée.

`DbEntityValidationResult` héberge un `DbEntityEntry` et un `IICollection<DbValidationErrors>` pour une entité

unique. Au début de cette méthode, un `DbEntityValidationResult` est instancié, puis toutes les erreurs découvertes sont ajoutées à sa collection `ValidationErrors`.

```
protected override DbEntityValidationResult ValidateEntity (
    System.Data.Entity.Infrastructure.DbEntityEntry entityEntry,
    IDictionary<object, object> items)
{
    var result = new DbEntityValidationResult(entityEntry, new List<DbValidationError>());

    if (entityEntry.Entity is Post post && entityEntry.State == EntityState.Added)
    {
        // Check for uniqueness of post title
        if (Posts.Where(p => p.Title == post.Title).Any())
        {
            result.ValidationErrors.Add(
                new System.Data.Entity.Validation.DbValidationError(
                    nameof>Title),
                    "Post title must be unique."));
        }
    }

    if (result.ValidationErrors.Count > 0)
    {
        return result;
    }
    else
    {
        return base.ValidateEntity(entityEntry, items);
    }
}
```

## Déclenchement explicite de la validation

Un appel à `SaveChanges` déclenche toutes les validations traitées dans cet article. Mais vous n'avez pas besoin de vous appuyer sur `SaveChanges`. Vous préférerez peut-être valider ailleurs dans votre application.

`DbContext.GetValidationErrors` déclenchera toutes les validations, celles définies par les annotations ou l'API Fluent, la validation créée dans `IValidatableObject` (par exemple, `Blog.Validate`) et les validations effectuées dans la méthode `DbContext.ValidateEntity`.

Le code suivant appellera `GetValidationErrors` sur l'instance actuelle d'une `DbContext`. Les `ValidationErrors` sont regroupés par type d'entité en `DbEntityValidationResult`. Le code itère d'abord au `DbEntityValidationResult`s retourné par la méthode, puis à travers chaque `DbValidationError` à l'intérieur de.

```
foreach (var validationResult in db.GetValidationErrors())
{
    foreach (var error in validationResult.ValidationErrors)
    {
        Debug.WriteLine(
            "Entity Property: {0}, Error {1}",
            error.PropertyName,
            error.ErrorMessage);
    }
}
```

## Autres considérations relatives à l'utilisation de la validation

Voici quelques autres points à prendre en compte lors de l'utilisation de la validation Entity Framework :

- Le chargement différé est désactivé au cours de la validation

- EF valide les annotations de données sur les propriétés non mappées (propriétés qui ne sont pas mappées à une colonne de la base de données)
- La validation est effectuée après la détection des modifications au cours de `SaveChanges`. Si vous apportez des modifications au cours de la validation, il vous incombe de notifier le dispositif de suivi des modifications
- `DbUnexpectedValidationException` est levée si des erreurs se produisent pendant la validation
- Les facettes qui Entity Framework incluses dans le modèle (longueur maximale, obligatoire, etc.) entraînent la validation, même s'il n'y a pas d'annotations de données dans vos classes et/ou si vous avez utilisé le concepteur EF pour créer votre modèle
- Règles de précédence :
  - Les appels de l'API Fluent remplacent les annotations de données correspondantes
- Ordre d'exécution :
  - La validation de la propriété se produit avant la validation du type
  - La validation de type se produit uniquement si la validation de la propriété a échoué
- Si une propriété est complexe, sa validation inclut également les éléments suivants :
  - Validation au niveau de la propriété sur les propriétés de type complexe
  - Validation au niveau du type sur le type complexe, y compris `IValidatableObject` validation sur le type complexe

## Récapitulatif

L'API de validation de Entity Framework s'exécute très bien avec la validation côté client dans MVC, mais vous n'avez pas à vous appuyer sur la validation côté client. Entity Framework s'occupe de la validation côté serveur pour DataAnnotations ou les configurations que vous avez appliquées avec l'API Fluent code First.

Vous avez également vu un certain nombre de points d'extensibilité pour personnaliser le comportement, que vous utilisez l'interface `IValidatableObject` ou que vous appuyez sur la méthode `DbContext.ValidateEntity`. Et ces deux derniers moyens de validation sont disponibles via le `DbContext`, que vous utilisez le flux de travail Code First, Model First ou Database First pour décrire votre modèle conceptuel.

# Blogs Entity Framework

23/11/2019 • 2 minutes to read

Outre la documentation du produit, ces blogs peuvent être une source d'informations utiles sur Entity Framework :

## Blogs de l'équipe EF

- [Balise de blog .NET : Entity Framework](#)
- [Blog ADO.NET \(n'est plus utilisé\)](#)
- [Blog de conception EF \(n'est plus utilisé\)](#)

## Blogueurs d'équipe EF actuels et anciens

- [Arthur Vickers](#)
- [Brice Lambson](#)
- [Diego Vega](#)
- [Rowan Miller](#)
- [Pawel Kadluczka](#)
- [Alex James](#)
- [Zlatko Michailov](#)

## Blogueurs de la communauté EF

- [Julie Lerman](#)
- [Shawn Wildermuth](#)

# Études de cas Microsoft pour Entity Framework

13/09/2018 • 8 minutes to read

Les études de cas sur cette page mettent en évidence quelques projets de production réel qui employaient Entity Framework.

## NOTE

Les versions détaillées de ces études de cas ne sont plus disponibles sur le site Web Microsoft. Par conséquent, les liens ont été supprimés.

## Epicor

Epicor est une société de logiciels d'envergure mondiale (avec des développeurs plus de 400) qui développe des solutions de planification ERP (Enterprise Resource) pour les entreprises dans plus de 150 pays. Produit phare, Epicor 9, est basé sur une Architecture orientée services (SOA) à l'aide de .NET Framework. Face à nombreuses demandes de client pour fournir la prise en charge de requête LINQ (Language Integrated) et qui veulent également à réduire la charge sur leurs serveurs principaux SQL Server, l'équipe a décidé de mettre à niveau vers Visual Studio 2010 et .NET Framework 4.0. À l'aide d'Entity Framework 4.0, ils pouvaient atteindre ces objectifs et de simplifier considérablement le développement et maintenance. En particulier, riche T4 prise en charge d'Entity Framework leur permettait de prendre le contrôle de leur code généré et générer automatiquement des fonctionnalités de l'enregistrement des performances telles que les requêtes précompilées et la mise en cache.

« Nous avons effectué des tests de performances récemment avec le code existant, et nous avons pu réduire les demandes vers SQL Server 90 %. Cela est dû à ADO.NET Entity Framework 4. » – Études de produits Erik Johnson, vice-président,

## Solutions de la véracité

Avoir acquis un système de logiciels de planification d'événements qui devait être difficiles à maintenir et à étendre à long terme, les Solutions véracité utilisé Visual Studio 2010 au pour réécrire comme une Application d'Internet riche puissante et facile à utiliser basé sur Silverlight 4. À l'aide des Services RIA .NET, ils ont été en mesure de générer rapidement une couche de service sur Entity Framework qui éviter la duplication de code et d'autorisation pour la validation courants et de la logique d'authentification au sein des niveaux.

« Nous étions vendues sur Entity Framework lors de sa première apparition et Entity Framework 4 s'est avérée être encore meilleures. Outils a été amélioré, et il est plus facile à manipuler les fichiers .edmx qui définissent le modèle conceptuel, le modèle de stockage et le mappage entre ces modèles... Avec Entity Framework, je peux obtenir cette couche d'accès aux données fonctionne en une journée et générerez-le out en entrant le long. Entity Framework est notre couche d'accès aux données de fait ; Je ne sais pas pourquoi tout le monde n'aurait pas l'utiliser. » – Joe McBride, Senior Developer

## Solutions NEC affichage d'Amérique

NEC souhaitait accéder au marché à des fins publicitaires en fonction du lieu de numérique avec une solution à bénéficier des annonceurs et propriétaires de réseau et à augmenter sa propre chiffre d'affaires. Pour ce faire, il lancé une paire d'applications web qui automatisent les processus manuels requis dans une campagne de publicité traditionnel. Les sites ont été créés à l'aide d'ASP.NET, Silverlight 3, AJAX et WCF, ainsi que de Entity Framework dans la couche d'accès aux données pour communiquer avec SQL Server 2008.

« Avec SQL Server, nous avons pensé nous aurions pu obtenir le débit que nous avions besoin pour traiter les annonceurs et les réseaux avec des informations en temps réel et la fiabilité afin de garantir que les informations contenues dans nos applications critiques sont toujours accessibles »-Mike Corcoran, Directeur informatique

## Dimensions de Darwin

À l'aide d'une large gamme de produits Microsoft, l'équipe de Darwin la valeur pour Evolver - un portail en ligne avatar que les clients peuvent utiliser pour créer les avatars étonnantes, plus réalistes pour une utilisation dans les pages de mise en réseau sociales, des animations et des jeux de créer. Les avantages de productivité d'Entity Framework et en extrayant des composants tels que Windows Workflow Foundation (WF) et Windows Server AppFabric (un cache d'application hautement évolutif en mémoire), l'équipe a été en mesure de livrer un produit incroyable dans moins de 35 % temps de développement. En dépit de membres de l'équipe divisé en plusieurs pays, l'équipe suivant un processus de développement agile avec les versions hebdomadaires.

« Nous essayez de ne pas créer de technologie considèrent. En tant que start-up, il est essentiel que nous exploiter la technologie qui enregistre le temps et argent. .NET a le choix pour le développement rapide et économique. » – Zachary Olsen, architecte

## Argenterie

Avec plus de 15 ans d'expérience dans le développement de point de vente (PDV) solutions pour les groupes de restaurant petites et moyennes entreprises, l'équipe de développement chez argenterie entrepris d'améliorer leur produit avec davantage de fonctionnalités de niveau entreprise afin d'attirer plus grande chaînes de restaurant. À l'aide de la dernière version des outils de développement de Microsoft, ils ont été en mesure de générer la nouvelle solution quatre fois plus rapidement que jamais. Nouvelles fonctionnalités clés telles que LINQ et Entity Framework simplifié la déplacer à partir de Crystal Reports vers SQL Server 2008 et SQL Server Reporting Services (SSRS) pour le stockage de leurs données et besoins de rapport.

« Gestion des données efficace est essentielle à la réussite d'argenterie – et c'est pourquoi nous avons décidé d'adopter SQL Reporting ». -Nicholas Romanidis, directeur informatique / d'ingénierie logicielle

# Contribuer à Entity Framework 6

13/09/2018 • 2 minutes to read

Entity Framework 6 est développé à l'aide d'un modèle open source sur GitHub. Bien que le principal objectif de l'équipe Entity Framework Microsoft est sur l'ajout de nouvelles fonctionnalités à Entity Framework Core, et nous ne pensons pas toutes les fonctionnalités principales à ajouter à Entity Framework 6, nous avons toujours accepter les contributions.

Contributions de produit, Veuillez commencer pour le [page wiki de contribution dans notre référentiel GitHub](#).

Pour les contributions à la documentation, redémarrez lecture le [Guide de contribution](#) dans notre référentiel de documentation.

# Obtenir de l'aide à l'aide de Entity Framework

11/10/2019 • 2 minutes to read



## Questions sur l'utilisation d'EF

La meilleure façon d'obtenir de l'aide à l'aide de Entity Framework consiste à [poster une question sur Stack Overflow](#) à l'aide de la balise **Entity-Framework**.

Si vous n'êtes pas familiarisé avec Stack Overflow, [Lisez les instructions pour poser des questions](#). En particulier, n'utilisez pas Stack Overflow pour signaler des bogues, poser des questions sur le plan ou suggérer de nouvelles fonctionnalités.

## Rapports de bogues et demandes de fonctionnalités

Si vous avez trouvé un bogue que vous pensez être résolu, si vous avez une fonctionnalité que vous souhaitez voir implémentée ou une question à laquelle vous n'avez pas trouvé de réponse, créez un problème dans [le référentiel GITHUB EF6](#).

# Glossaire Entity Framework

07/11/2019 • 6 minutes to read

## Code First

Création d'un modèle de Entity Framework à l'aide de code. Le modèle peut cibler une base de données existante ou une nouvelle base de données.

## Contexte

Classe qui représente une session avec la base de données, ce qui vous permet d'interroger et d'enregistrer des données. Un contexte dérive de la classe DbContext ou ObjectContext.

## Convention (Code First)

Règle que Entity Framework utilise pour déduire la forme de votre modèle à partir de vos classes.

## Database First

Création d'un modèle de Entity Framework, à l'aide du concepteur EF, ciblant une base de données existante.

## Chargement hâtif

Modèle de chargement de données associées dans laquelle une requête pour un type d'entité charge également des entités associées dans le cadre de la requête.

## EF Designer

Concepteur visuel dans Visual Studio qui vous permet de créer un modèle de Entity Framework à l'aide de zones et de lignes.

## Entité

Classe ou objet qui représente des données d'application, telles que des clients, des produits et des commandes.

## Entity Data Model

Modèle qui décrit les entités et les relations entre eux. EF utilise le modèle EDM pour décrire le modèle conceptuel sur lequel les programmes de développement sont utilisés. EDM s'appuie sur le modèle de relation d'entité introduit par Dr. Peter Chen. L'EDM a été développé à l'origine avec l'objectif principal de devenir le modèle de données commun à travers une suite de technologies de développement et de serveur de Microsoft. Le modèle EDM est également utilisé dans le cadre du protocole OData.

## Chargement explicite

Modèle de chargement de données associées où les objets connexes sont chargés en appelant une API.

## API Fluent

API qui peut être utilisée pour configurer un modèle de Code First.

## Association de clé étrangère

Association entre des entités où une propriété qui représente la clé étrangère est incluse dans la classe de l'entité dépendante. Par exemple, Product contient une propriété CategoryId.

## Relation d'identification

Relation où la clé primaire de l'entité principale fait également partie de la clé primaire de l'entité dépendante. Dans ce type de relation, l'entité dépendante ne peut pas exister dans l'entité principale.

## Association indépendante

Association entre des entités où il n'y a aucune propriété représentant la clé étrangère dans la classe de l'entité dépendante. Par exemple, une classe Product contient une relation à Category, mais aucune propriété CategoryId. Entity Framework effectue le suivi de l'état de l'Association indépendamment de l'état des entités aux terminaisons des deux associations.

## Chargement différé

Modèle de chargement de données associées où les objets connexes sont chargés automatiquement lors de l'accès à une propriété de navigation.

## Model First

Création d'un modèle de Entity Framework à l'aide du concepteur EF, qui est ensuite utilisé pour créer une nouvelle base de données.

## Propriété de navigation

Propriété d'une entité qui référence une autre entité. Par exemple, Product contient une propriété de navigation Category et Category contient une propriété de navigation Products.

## POCO

Acronyme de Plain-Old CLR Object. Classe d'utilisateur simple qui n'a pas de dépendances avec n'importe quelle infrastructure. Dans le contexte d'EF, une classe d'entité qui ne dérive pas de EntityObject, implémente toutes les interfaces ou transporte tous les attributs définis dans EF. De telles classes d'entité qui sont découpées de l'infrastructure de persistance sont également dites « ignorant la persistance ».

## Relation inverse

L'extrémité opposée d'une relation, par exemple Product. Catégorie et catégorie. Production.

## Entité de suivi automatique

Entité générée à partir d'un modèle de génération de code qui permet le développement multicouche.

## Type de table par béton (TPC)

Méthode de mappage de l'héritage où chaque type non abstrait de la hiérarchie est mappé à une table distincte dans la base de données.

## TPH (table par hiérarchie)

Méthode de mappage de l'héritage dans lequel tous les types de la hiérarchie sont mappés à la même table dans la base de données. Une ou plusieurs colonnes de discriminateur sont utilisées pour identifier le type auquel chaque ligne est associée.

## Table par type (TPT)

Méthode de mappage de l'héritage où les propriétés communes de tous les types de la hiérarchie sont mappées à la même table dans la base de données, mais les propriétés propres à chaque type sont mappées à une table distincte.

## Détection de type

Processus d'identification des types qui doivent faire partie d'un modèle de Entity Framework.

# Base de données exemple School

13/09/2018 • 15 minutes to read

Cette rubrique contient le schéma et les données pour la base de données School. La base de données School est utilisé dans divers emplacements de la documentation d'Entity Framework.

## NOTE

Le serveur de base de données qui est installé avec Visual Studio est différent selon la version de Visual Studio que vous utilisez. Consultez [Visual Studio versions](#) pour plus d'informations sur les éléments à utiliser.

Voici les étapes pour créer la base de données :

- Ouvrir Visual Studio
- **Vue -> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **des connexions de données -> ajouter une connexion...**
- Si vous n'avez pas connecté à une base de données à partir de l'Explorateur de serveurs avant que vous devrez sélectionner **Microsoft SQL Server** comme source de données
- Se connecter à la base de données locale ou de SQL Express, en fonction de celles que vous avez installé
- Entrez **School** en tant que le nom de la base de données
- Sélectionnez **OK** et vous demandera si vous souhaitez créer une base de données, sélectionnez **Oui**
- La nouvelle base de données s'affiche désormais dans l'Explorateur de serveurs
- Si vous utilisez Visual Studio 2012 ou version ultérieure
  - Avec le bouton droit sur la base de données dans l'Explorateur de serveurs, puis sélectionnez **nouvelle requête**
  - Copiez le code SQL suivant dans la nouvelle requête, puis avec le bouton droit sur la requête, puis sélectionnez **Execute**
- Si vous utilisez Visual Studio 2010
  - Sélectionnez **données -> Transact SQL éditeur -> nouvelle connexion à la requête...**
  - Entrez **. \SQLEXPRESS** en tant que nom du serveur et cliquez sur **OK**
  - Sélectionnez le **STESample** de base de données dans la liste déroulante vers le bas en haut de l'éditeur de requête
  - Copiez le code SQL suivant dans la nouvelle requête, puis avec le bouton droit sur la requête, puis sélectionnez **exécuter SQL**

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

-- Create the Department table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[Department]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Department]([DepartmentID] [int] NOT NULL,
[Name] [nvarchar](50) NOT NULL,
[Budget] [money] NOT NULL,
[StartDate] [datetime] NOT NULL,
[Administrator] [int] NULL,
CONSTRAINT [PK_Department1] PRIMARY KEY CLUSTERED
```

```

(
[DepartmentID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the Person table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[Person]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Person]([PersonID] [int] IDENTITY(1,1) NOT NULL,
[LastName] [nvarchar](50) NOT NULL,
[FirstName] [nvarchar](50) NOT NULL,
[HireDate] [datetime] NULL,
[EnrollmentDate] [datetime] NULL,
[Discriminator] [nvarchar](50) NOT NULL,
CONSTRAINT [PK_School.Student] PRIMARY KEY CLUSTERED
(
[PersonID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the OnsiteCourse table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[OnsiteCourse]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[OnsiteCourse]([CourseID] [int] NOT NULL,
[Location] [nvarchar](50) NOT NULL,
[Days] [nvarchar](50) NOT NULL,
[Time] [smalldatetime] NOT NULL,
CONSTRAINT [PK_OnsiteCourse] PRIMARY KEY CLUSTERED
(
[CourseID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the OnlineCourse table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[OnlineCourse]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[OnlineCourse]([CourseID] [int] NOT NULL,
[URL] [nvarchar](100) NOT NULL,
CONSTRAINT [PK_OnlineCourse] PRIMARY KEY CLUSTERED
(
[CourseID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

--Create the StudentGrade table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[StudentGrade]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[StudentGrade]([EnrollmentID] [int] IDENTITY(1,1) NOT NULL,
[CourseID] [int] NOT NULL,
[StudentID] [int] NOT NULL,
[Grade] [decimal](3, 2) NULL,
CONSTRAINT [PK_StudentGrade] PRIMARY KEY CLUSTERED
(
[EnrollmentID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

```

```

-- Create the CourseInstructor table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[CourseInstructor]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[CourseInstructor]([CourseID] [int] NOT NULL,
[PersonID] [int] NOT NULL,
CONSTRAINT [PK_CourseInstructor] PRIMARY KEY CLUSTERED
(
[CourseID] ASC,
[PersonID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the Course table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[Course]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Course]([CourseID] [int] NOT NULL,
[Title] [nvarchar](100) NOT NULL,
[Credits] [int] NOT NULL,
[DepartmentID] [int] NOT NULL,
CONSTRAINT [PK_School.Course] PRIMARY KEY CLUSTERED
(
[CourseID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Create the OfficeAssignment table.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[OfficeAssignment]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[OfficeAssignment]([InstructorID] [int] NOT NULL,
[Location] [nvarchar](50) NOT NULL,
[Timestamp] [timestamp] NOT NULL,
CONSTRAINT [PK_OfficeAssignment] PRIMARY KEY CLUSTERED
(
[InstructorID] ASC
)WITH (IGNORE_DUP_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]
END
GO

-- Define the relationship between OnsiteCourse and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_OnsiteCourse_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[OnsiteCourse]'))
ALTER TABLE [dbo].[OnsiteCourse] WITH CHECK ADD
CONSTRAINT [FK_OnsiteCourse_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO

ALTER TABLE [dbo].[OnsiteCourse] CHECK
CONSTRAINT [FK_OnsiteCourse_Course]
GO

-- Define the relationship between OnlineCourse and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_OnlineCourse_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[OnlineCourse]'))
ALTER TABLE [dbo].[OnlineCourse] WITH CHECK ADD
CONSTRAINT [FK_OnlineCourse_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO

ALTER TABLE [dbo].[OnlineCourse] CHECK
CONSTRAINT [FK_OnlineCourse_Course]

```

```

CONSTRAINT [FK_OnlineCourse_Course]
GO

-- Define the relationship between StudentGrade and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_StudentGrade_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[StudentGrade]'))
ALTER TABLE [dbo].[StudentGrade] WITH CHECK ADD
CONSTRAINT [FK_StudentGrade_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO

ALTER TABLE [dbo].[StudentGrade] CHECK
CONSTRAINT [FK_StudentGrade_Course]
GO

--Define the relationship between StudentGrade and Student.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_StudentGrade_Student]')
AND parent_object_id = OBJECT_ID(N'[dbo].[StudentGrade]'))
ALTER TABLE [dbo].[StudentGrade] WITH CHECK ADD
CONSTRAINT [FK_StudentGrade_Student] FOREIGN KEY([StudentID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[StudentGrade] CHECK
CONSTRAINT [FK_StudentGrade_Student]
GO

-- Define the relationship between CourseInstructor and Course.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_CourseInstructor_Course]')
AND parent_object_id = OBJECT_ID(N'[dbo].[CourseInstructor]'))
ALTER TABLE [dbo].[CourseInstructor] WITH CHECK ADD
CONSTRAINT [FK_CourseInstructor_Course] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Course] ([CourseID])
GO

ALTER TABLE [dbo].[CourseInstructor] CHECK
CONSTRAINT [FK_CourseInstructor_Course]
GO

-- Define the relationship between CourseInstructor and Person.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_CourseInstructor_Person]')
AND parent_object_id = OBJECT_ID(N'[dbo].[CourseInstructor]'))
ALTER TABLE [dbo].[CourseInstructor] WITH CHECK ADD
CONSTRAINT [FK_CourseInstructor_Person] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[CourseInstructor] CHECK
CONSTRAINT [FK_CourseInstructor_Person]
GO

-- Define the relationship between Course and Department.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_Course_Department]')
AND parent_object_id = OBJECT_ID(N'[dbo].[Course]'))
ALTER TABLE [dbo].[Course] WITH CHECK ADD
CONSTRAINT [FK_Course_Department] FOREIGN KEY([DepartmentID])
REFERENCES [dbo].[Department] ([DepartmentID])
GO

ALTER TABLE [dbo].[Course] CHECK CONSTRAINT [FK_Course_Department]
GO

--Define the relationship between OfficeAssignment and Person.
IF NOT EXISTS (SELECT * FROM sys.foreign_keys
WHERE object_id = OBJECT_ID(N'[dbo].[FK_OfficeAssignment_Person]')
AND parent_object_id = OBJECT_ID(N'[dbo].[OfficeAssignment]'))
ALTER TABLE [dbo].[OfficeAssignment] WITH CHECK ADD
CONSTRAINT [FK_OfficeAssignment_Person] FOREIGN KEY([InstructorID])
REFERENCES [dbo].[Person] ([PersonID])

```

```

GO
ALTER TABLE [dbo].[OfficeAssignment] CHECK
CONSTRAINT [FK_OfficeAssignment_Person]
GO

-- Create InsertOfficeAssignment stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[InsertOfficeAssignment]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[InsertOfficeAssignment]
@InstructorID int,
@Location nvarchar(50)
AS
INSERT INTO dbo.OfficeAssignment (InstructorID, Location)
VALUES (@InstructorID, @Location);
IF @@ROWCOUNT > 0
BEGIN
SELECT [Timestamp] FROM OfficeAssignment
WHERE InstructorID=@InstructorID;
END
'
END
GO

--Create the UpdateOfficeAssignment stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[UpdateOfficeAssignment]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[UpdateOfficeAssignment]
@InstructorID int,
@Location nvarchar(50),
@OrigTimestamp timestamp
AS
UPDATE OfficeAssignment SET Location=@Location
WHERE InstructorID=@InstructorID AND [Timestamp]=@OrigTimestamp;
IF @@ROWCOUNT > 0
BEGIN
SELECT [Timestamp] FROM OfficeAssignment
WHERE InstructorID=@InstructorID;
END
'
END
GO

-- Create the DeleteOfficeAssignment stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[DeleteOfficeAssignment]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[DeleteOfficeAssignment]
@InstructorID int
AS
DELETE FROM OfficeAssignment
WHERE InstructorID=@InstructorID;
'
END
GO

-- Create the DeletePerson stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[DeletePerson]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'

```

```

CREATE PROCEDURE [dbo].[DeletePerson]
@PersonID int
AS
DELETE FROM Person WHERE PersonID = @PersonID;
'

END
GO

-- Create the UpdatePerson stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[UpdatePerson]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[UpdatePerson]
@PersonID int,
@LastName nvarchar(50),
@FirstName nvarchar(50),
@HireDate datetime,
@EnrollmentDate datetime,
@Discriminator nvarchar(50)
AS
UPDATE Person SET LastName=@LastName,
FirstName=@FirstName,
HireDate=@HireDate,
EnrollmentDate=@EnrollmentDate,
Discriminator=@Discriminator
WHERE PersonID=@PersonID;
'

END
GO

-- Create the InsertPerson stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[InsertPerson]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[InsertPerson]
@LastName nvarchar(50),
@FirstName nvarchar(50),
@HireDate datetime,
@EnrollmentDate datetime,
@Discriminator nvarchar(50)
AS
INSERT INTO dbo.Person (LastName,
FirstName,
HireDate,
EnrollmentDate,
Discriminator)
VALUES (@LastName,
@FirstName,
@HireDate,
@EnrollmentDate,
@Discriminator);
SELECT SCOPE_IDENTITY() as NewPersonID;
'

END
GO

-- Create GetStudentGrades stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[GetStudentGrades]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[GetStudentGrades]
@studentID int
AS

```

```

SELECT EnrollmentID, Grade, CourseID, StudentID FROM dbo.StudentGrade
WHERE StudentID = @StudentID
'
END
GO

-- Create GetDepartmentName stored procedure.
IF NOT EXISTS (SELECT * FROM sys.objects
WHERE object_id = OBJECT_ID(N'[dbo].[GetDepartmentName]')
AND type in (N'P', N'PC'))
BEGIN
EXEC dbo.sp_executesql @statement = N'
CREATE PROCEDURE [dbo].[GetDepartmentName]
@ID int,
@Name nvarchar(50) OUTPUT
AS
SELECT @Name = Name FROM Department
WHERE DepartmentID = @ID
'
END
GO

-- Insert data into the Person table.
USE School
GO
SET IDENTITY_INSERT dbo.Person ON
GO
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (1, 'Abercrombie', 'Kim', '1995-03-11', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (2, 'Barzdukas', 'Gytis', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (3, 'Justice', 'Peggy', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (4, 'Fakhouri', 'Fadi', '2002-08-06', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (5, 'Harui', 'Roger', '1998-07-01', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (6, 'Li', 'Yan', null, '2002-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (7, 'Norman', 'Laura', null, '2003-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (8, 'Olivotto', 'Nino', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (9, 'Tang', 'Wayne', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (10, 'Alonso', 'Meredith', null, '2002-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (11, 'Lopez', 'Sophia', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (12, 'Browning', 'Meredith', null, '2000-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (13, 'Anand', 'Arturo', null, '2003-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (14, 'Walker', 'Alexandra', null, '2000-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (15, 'Powell', 'Carson', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (16, 'Jai', 'Damien', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (17, 'Carlson', 'Robyn', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (18, 'Zheng', 'Roger', '2004-02-12', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (19, 'Bryant', 'Carson', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (20, 'Suarez', 'Robyn', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (21, 'Holt', 'Roger', null, '2004-09-01', 'Student');

```

```

INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (22, 'Alexander', 'Carson', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (23, 'Morgan', 'Isaiah', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (24, 'Martin', 'Randall', null, '2005-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (25, 'Kapoor', 'Candace', '2001-01-15', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (26, 'Rogers', 'Cody', null, '2002-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (27, 'Serrano', 'Stacy', '1999-06-01', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (28, 'White', 'Anthony', null, '2001-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (29, 'Griffin', 'Rachel', null, '2004-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (30, 'Shan', 'Alicia', null, '2003-09-01', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (31, 'Stewart', 'Jasmine', '1997-10-12', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (32, 'Xu', 'Kristen', '2001-07-23', null, 'Instructor');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (33, 'Gao', 'Erica', null, '2003-01-30', 'Student');
INSERT INTO dbo.Person (PersonID, LastName, FirstName, HireDate, EnrollmentDate, Discriminator)
VALUES (34, 'Van Houten', 'Roger', '2000-12-07', null, 'Instructor');
GO
SET IDENTITY_INSERT dbo.Person OFF
GO

-- Insert data into the Department table.
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (1, 'Engineering', 350000.00, '2007-09-01', 2);
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (2, 'English', 120000.00, '2007-09-01', 6);
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (4, 'Economics', 200000.00, '2007-09-01', 4);
INSERT INTO dbo.Department (DepartmentID, [Name], Budget, StartDate, Administrator)
VALUES (7, 'Mathematics', 250000.00, '2007-09-01', 3);
GO

-- Insert data into the Course table.
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (1050, 'Chemistry', 4, 1);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (1061, 'Physics', 4, 1);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (1045, 'Calculus', 4, 7);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (2030, 'Poetry', 2, 2);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (2021, 'Composition', 3, 2);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (2042, 'Literature', 4, 2);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (4022, 'Microeconomics', 3, 4);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (4041, 'Macroeconomics', 3, 4);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (4061, 'Quantitative', 2, 4);
INSERT INTO dbo.Course (CourseID, Title, Credits, DepartmentID)
VALUES (3141, 'Trigonometry', 4, 7);
GO

-- Insert data into the OnlineCourse table.
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES ('2020', 'http://www.finertschool.net/Poetry');

```

```

VALUES (2050, 'http://www.fineartschool.net/poetry'),
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES (2021, 'http://www.fineartschool.net/Composition');
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES (4041, 'http://www.fineartschool.net/Macroeconomics');
INSERT INTO dbo.OnlineCourse (CourseID, URL)
VALUES (3141, 'http://www.fineartschool.net/Trigonometry');

--Insert data into OnsiteCourse table.
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (1050, '123 Smith', 'MTWH', '11:30');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (1061, '234 Smith', 'TWHF', '13:15');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (1045, '121 Smith', 'MWHF', '15:30');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (4061, '22 Williams', 'TH', '11:15');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (2042, '225 Adams', 'MTWH', '11:00');
INSERT INTO dbo.OnsiteCourse (CourseID, Location, Days, [Time])
VALUES (4022, '23 Williams', 'MWF', '9:00');

-- Insert data into the CourseInstructor table.
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (1050, 1);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (1061, 31);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (1045, 5);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (2030, 4);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (2021, 27);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (2042, 25);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (4022, 18);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (4041, 32);
INSERT INTO dbo.CourseInstructor(CourseID, PersonID)
VALUES (4061, 34);
GO

--Insert data into the OfficeAssignment table.
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (1, '17 Smith');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (4, '29 Adams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (5, '37 Williams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (18, '143 Smith');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (25, '57 Adams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (27, '271 Williams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (31, '131 Smith');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (32, '203 Williams');
INSERT INTO dbo.OfficeAssignment(InstructorID, Location)
VALUES (34, '213 Smith');

-- Insert data into the StudentGrade table.
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 2, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2030, 2, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 2, 3);

```

```
VALUES (2021, 3, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2030, 3, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 6, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2042, 6, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 7, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2042, 7, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2021, 8, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (2042, 8, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 9, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 10, null);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 11, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 12, null);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 12, null);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 14, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 13, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 13, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 14, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 15, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 16, 2);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 17, null);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 19, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 20, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 21, 2);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 22, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4041, 22, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4061, 22, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (4022, 23, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1045, 23, 1.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 24, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 25, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1050, 26, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 26, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 27, 3);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1045, 28, 2.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1050, 28, 3.5);
```

```
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 29, 4);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1050, 30, 3.5);
INSERT INTO dbo.StudentGrade (CourseID, StudentID, Grade)
VALUES (1061, 30, 4);
GO
```

# Extensions et les outils entity Framework

13/09/2018 • 2 minutes to read

## IMPORTANT

Extensions sont construites par une variété de sources et pas conservées dans le cadre d'Entity Framework. Quand vous envisagez une extension tierce, veillez à évaluer notamment la qualité, la gestion des licences, la compatibilité et la prise en charge pour vérifier qu'elle répond à vos besoins.

Entity Framework a été un O/RM populaires depuis de nombreuses années. Voici quelques exemples d'outils gratuits et payants et d'extensions développées pour celle-ci :

- [EF Power Tools Community Edition](#)
- [EF Profiler](#)
- [Profiler des ORM](#)
- [LINQPad](#)
- [LBLGen Pro](#)
- [Outils Huagati DBML/EDMX](#)
- [Entity Developer](#)