Ces vidéo et procédure pas à pas présentent le développement Code First ciblant une nouvelle base de données. Ce scénario inclut le ciblage d'une base de données qui n'existe pas et qui sera créée par Code First, ou d'une base de données vide à laquelle Code First ajoutera de nouvelles tables. 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 d'attributs sur vos classes et propriétés ou en utilisant une API Fluent.

(autres options vidéo y compris le téléchargement)

Configuration requise

Vous devez avoir installé Visual Studio 2010 ou Visual Studio 2012 pour effectuer cette procédure pas à pas.

Si vous utilisez Visual Studio 2010, vous devez également installer NuGet.

1. Créer l'application

Pour simplifier, vous allez créer une application console de base qui utilise Code First pour l'accès aux données.

Ouvrez Visual Studio.

Fichier -> Nouveau -> Projet...

Dans le menu gauche, sélectionnez **Fenêtres**, puis **Application console**.

Entrez **CodeFirstNewDatabaseSample** comme nom.

Cliquez sur **OK**.

2. Créer le modèle

Définissez un modèle très simple à l'aide de classes. Vous les définissez dans le fichier Program.cs, mais dans une application réelle, vous devez fractionner les classes dans des fichiers distincts et éventuellement un projet distinct.

Sous la définition de classe Program dans le fichier 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; }
}
```

Notez que les deux propriétés de navigation (Blog.Posts et Post.Blog) sont virtuelles. Cela active la fonctionnalité de chargement différé d'Entity Framework. Le chargement différé signifie que le contenu de ces propriétés est automatiquement chargé à partir de la base de données lorsque vous y accéder.

3. Créer un contexte

Maintenant il est temps de définir un contexte dérivé, qui représente une session de base de données, permettant d'interroger et d'enregistrer des données. Vous définissez un contexte qui dérive de System.Data.Entity.DbContext et expose un DbSet<TEntity> typé pour chaque classe dans le modèle.

Vous allez commencer à utiliser les types d'Entity Framework. Vous devez donc ajouter le package EntityFramework NuGet.

Projet –> Gérer les packages NuGet...

Remarque : si vous ne disposez pas de l'option **Gérer les packages NuGet...**, vous devez installer la version la plus récente de NuGet.

Sélectionnez l'onglet **En ligne**.

Sélectionnez le package **EntityFramework**.

Cliquez sur **Installer**.

Ajoutez une instruction using pour System.Data.Entity au début du fichier Program.cs.

```
using System.Data.Entity;
```

Sous la classe Post dans le fichier Program.cs, ajoutez le contexte dérivé suivant.

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

Voici la liste complète de ce que le fichier Program.cs doit 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; }
    }
}
```

Il s'agit du code nécessaire pour stocker et récupérer des données. Évidemment, du code s'exécute en arrière-plan, et vous l'examinerez dans un instant, mais observez d'abord le code en action.

4. Lire et écrire des données

Implémentez la méthode Main dans le fichier Program.cs, de la façon indiquée ci-dessous. Ce code crée une nouvelle instance du 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 triés alphabétiquement 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.

```
Entrer un nom pour un nouveau Blog : Blog ADO.NET
Tous les blogs dans la base de données :
Blog ADO.NET
Appuyer sur une touche pour quitter...
```

Où sont mes données ?

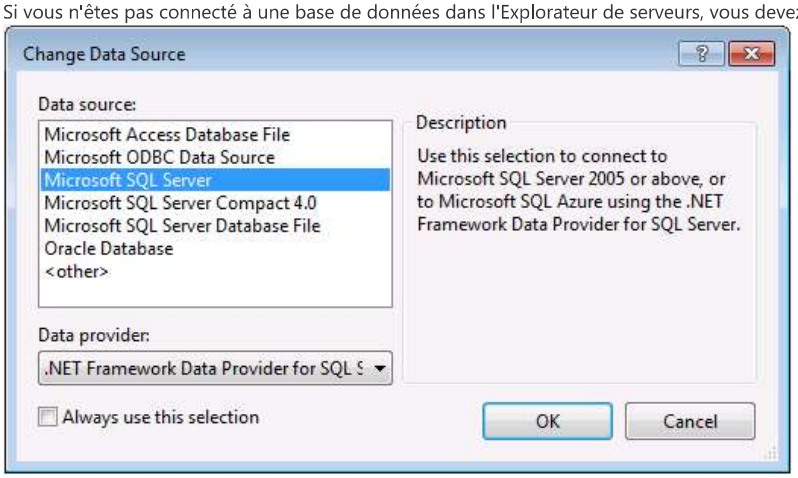
Par convention DbContext a créé une base de données pour vous.

- Si une instance SQL Express locale 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 utilise la base de données LocalDb (installée par défaut avec Visual Studio 2012)
- La base de données est nommée d'après le nom complet du contexte dérivé, dans ce cas **CodeFirstNewDatabaseSample.BloggingContext**

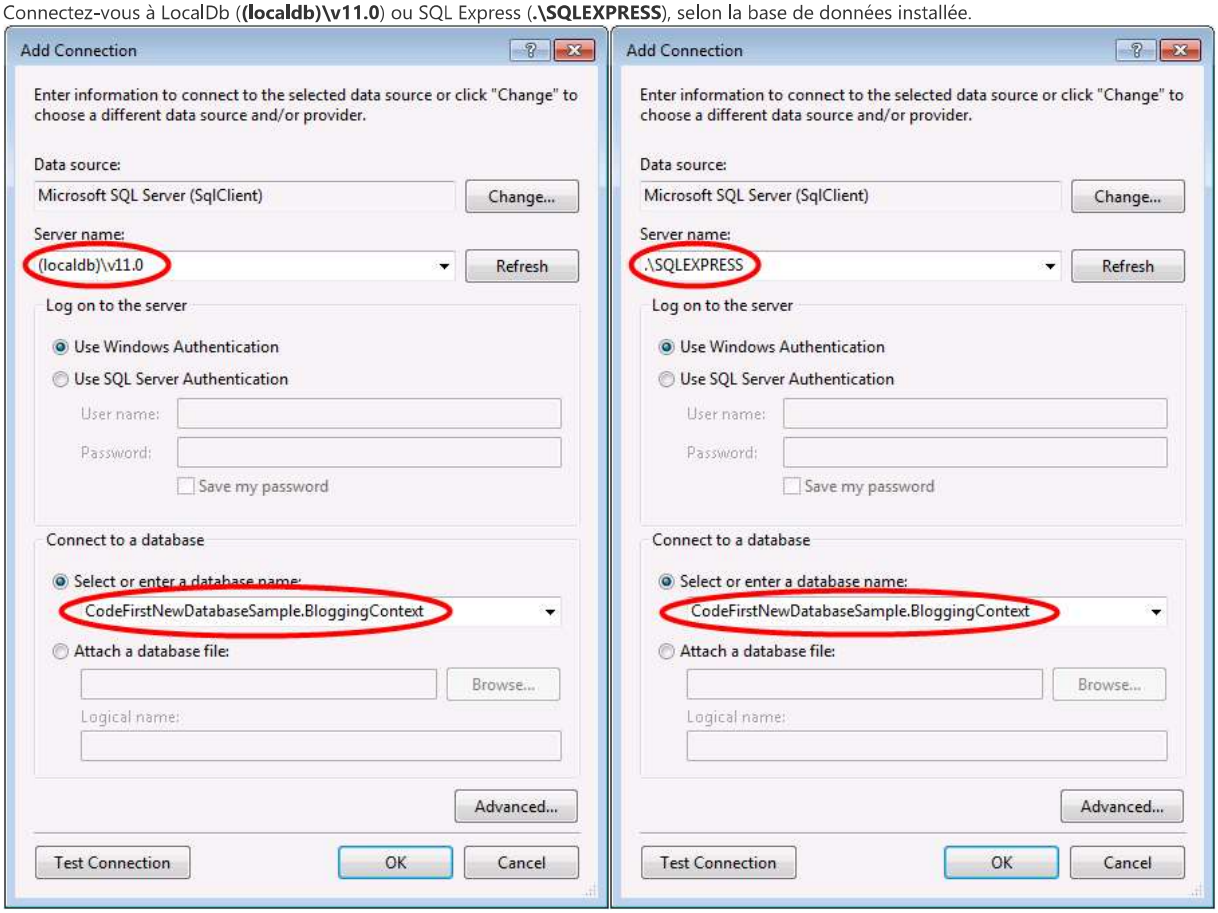
Il s'agit uniquement des conventions utilisées par défaut et il existe différentes méthodes pour modifier la base de données utilisée par Code First. Pour plus d'informations, consultez la rubrique **Comment DbContext détecte le modèle et la connexion de base de données.**

Connectez-vous à cette base de données à l'aide de l'Explorateur de serveurs dans Visual Studio.

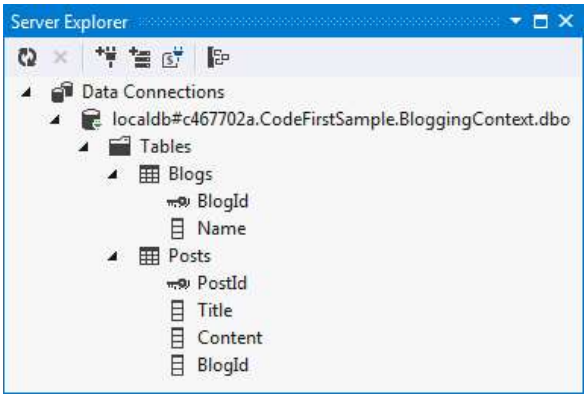
- **Afficher -> Explorateur de serveurs**
- Cliquez avec le bouton droit sur **Connexions de données**, puis cliquez sur **Ajouter une connexion...**.
- Si vous n'êtes pas connecté à une base de données dans l'Explorateur de serveurs, vous devez sélectionner Microsoft SQL Server comme source de données



Connectez-vous à LocalDb (**(localdb)\v11.0**) ou SQL Express (**.\SQLEXPRESS**), selon la base de données installée.



Examinez maintenant le schéma créé par Code First.



DbContext a créé les classes à inclure dans le modèle en examinant les propriétés DbSet définies. Il utilise ensuite l'ensemble par défaut de conventions Code First pour déterminer les noms de table et de colonne, pour déterminer les types de données, pour rechercher les clés primaires, etc. Dans une étape ultérieure de cette procédure, vous découvrirez comment remplacer ces conventions.

5. Traitement des modifications de modèle

Maintenant il est temps d'apporter certaines modifications à votre modèle, et vous devez également mettre à jour le schéma de base de données. Pour ce faire, vous allez utiliser une fonctionnalité appelée Migrations Code First, ou Migrations pour simplifier.

Migrations vous permet de disposer d'un jeu ordonné d'étapes qui décrivent comment mettre à niveau mettre à niveau (et mettre à niveau vers une version antérieure) votre 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 BloggingContext.

Outils -> Gestionnaire de package de bibliothèque -> Console Gestionnaire de package

Exécutez la commande **Enable-Migrations** dans la console du gestionnaire de package.

Un dossier Migrations a été ajouté au projet, qui contient deux éléments :

Configuration.cs – Ce fichier contient les paramètres qui seront utilisés par Migrations pour migrer BloggingContext. Aucune modification n'est nécessaire pour effectuer cette procédure pas à pas. Mais vous pouvez spécifier les données de départ, inscrire des fournisseurs pour d'autres bases de données, modifier l'espace de noms dans lequel les migrations sont générées, etc., dans ce fichier.

<timestamp>_InitialCreate.cs – Il s'agit de la première migration. Elle 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 inclut les tables Blogs et Posts. Bien que vous ayez laissé Code First créer automatiquement ces tables, maintenant que vous avez opté pour Migrations, elles ont été converties en une migration. Code First a également enregistré dans la base de données locale que cette Migration a déjà été appliquée. L'horodateur sur le nom de fichier est utilisé à des fins de tri.

Modifiez votre modèle, ajoutez une propriété Url à la classe 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 depuis la dernière migration et génère un modèle automatique de nouvelle migration avec les modifications détectées. Affectez un nom à la migration ; dans ce cas « AddUrl ».

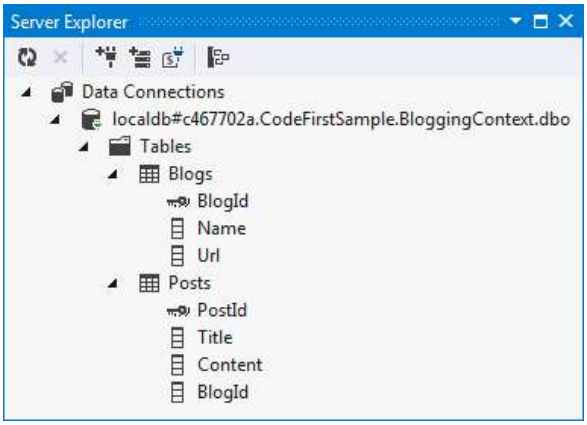
Le code généré automatiquement indique que vous devez ajouter une colonne Url, qui contient les données de chaîne, à la table dbo.Blogs. Le cas échéant, modifiez le code généré automatiquement. Cela n'est pas nécessaire dans ce cas.

```
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 les migrations en attente à la base de données. La migration InitialCreate a déjà été appliquée. Par conséquent, Migrations applique uniquement la nouvelle migration AddUrl.

Conseil : utilisez le commutateur **–Verbose** lorsque vous appelez Update-Database pour voir le code SQL exécuté sur la base de données.

Colonne Url ajoutée à la table Blogs de la base de données :



6. Annotations de données

Jusqu'à présent vous avez laissé Entity Framework détecter le modèle à l'aide des conventions utilisées par défaut. Cependant, il peut arriver que les classes ne respectent pas les conventions et vous devez être en mesure d'effectuer d'autres tâches de configuration. Il existe deux options à cet effet. Vous allez découvrir les annotations de données dans cette section et l'API Fluent dans la section suivante.

Ajoutez une classe User à votre modèle

```
public class User
{
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

Vous devez également ajouter une instruction SET à votre 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 vous essayez d'ajouter une migration, vous obtenez le message d'erreur suivant : « *EntityType « User » n'a pas de clé définie. Définissez la clé de cet EntityType.* », car Entity Framework n'a aucun moyen de savoir que Username doit être la clé principale de la classe User.

Vous allez utiliser des annotations de données. Vous devez donc ajouter une instruction using au début du fichier Program.cs.

```
using System.ComponentModel.DataAnnotations;
```

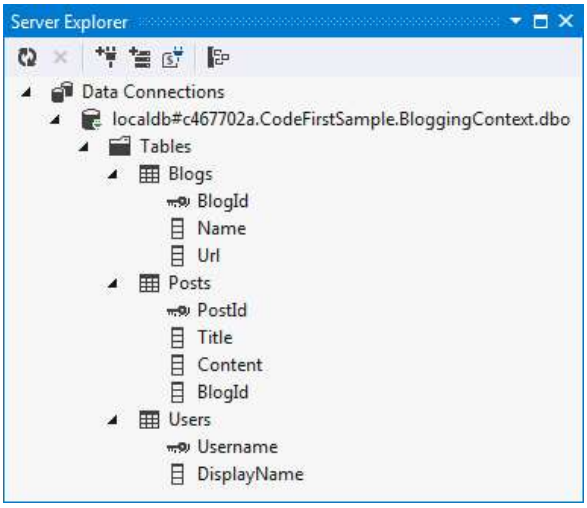
Maintenant annotez la propriété Username pour indiquer qu'il s'agit de la clé primaire.

```
public class User
{
    [Key]
    public string Username { get; set; }
    public string DisplayName { get; set; }
}
```

Utilisez la commande **Add-Migration AddUser** pour générer un modèle automatique de migration de façon à 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 ajoutée à la base de données :



Voici la liste complète des annotations prises en charge par Entity Framework :

- KeyAttribute
- StringLengthAttribute
- MaxLengthAttribute
- ConcurrencyCheckAttribute
- RequiredAttribute
- TimestampAttribute
- ComplexTypeAttribute
- ColumnAttribute
- TableAttribute
- InversePropertyAttribute
- ForeignKeyAttribute
- DatabaseGeneratedAttribute
- NotMappedAttribute

7. API Fluent

Dans la section précédente vous avez examiné comment utiliser les annotations de données pour compléter ou remplacer ce qui a été détecté par convention. Une autre méthode pour configurer le modèle consiste à utiliser l'API Fluent Code First.

La configuration de la plupart des modèles peut être effectuée à l'aide d'annotations de données simples. L'API Fluent constitue un moyen plus avancé pour spécifier une configuration de modèle qui couvre toute la configuration pouvant être effectuée par des annotations de données, en plus d'une configuration avancée impossible avec les annotations de données. Les annotations de données et l'API Fluent peuvent être utilisées ensemble.

Pour accéder à l'API Fluent vous remplacez la méthode OnModelCreating dans DbContext. Supposez que vous souhaitez renommer la colonne dans laquelle est stocké User.DisplayName display_name.

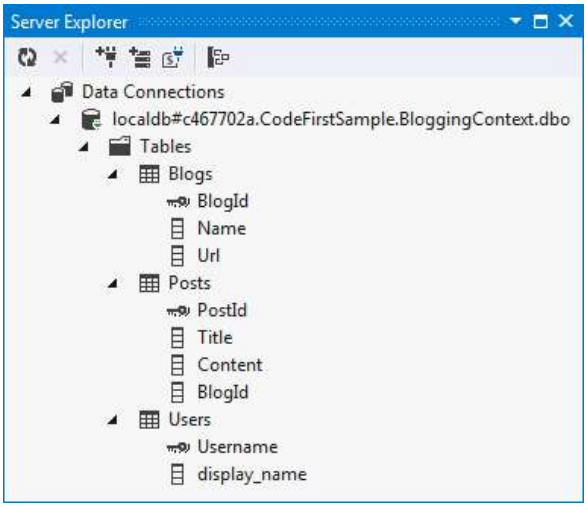
Remplacez la méthode OnModelCreating sur BloggingContext par le code suivant

```
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 un modèle automatique de migration de façon à 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 display_name :



Résumé

Dans cette procédure pas à pas, vous avez examiné le développement Code First à l'aide d'une nouvelle base de données. Vous avez défini un modèle à l'aide de classes, puis utilisé ce modèle pour créer une base de données et stocker et récupérer des données. Une fois la base de données créée, vous avez utilisé Migrations Code First pour modifier le schéma à mesure que le modèle a évolué. Vous avez également appris à configurer un modèle à l'aide des annotations de données et de l'API Fluent.