ASP.Net MVC

Entity Framework 6

Edmx, Navigation, Lazy loading, Annotations, Outils VS, CRUD, T4, Accès concurrents, Attachement, opération classiques Asp.Net, héritage TPT et TPH, Migrations

Principes

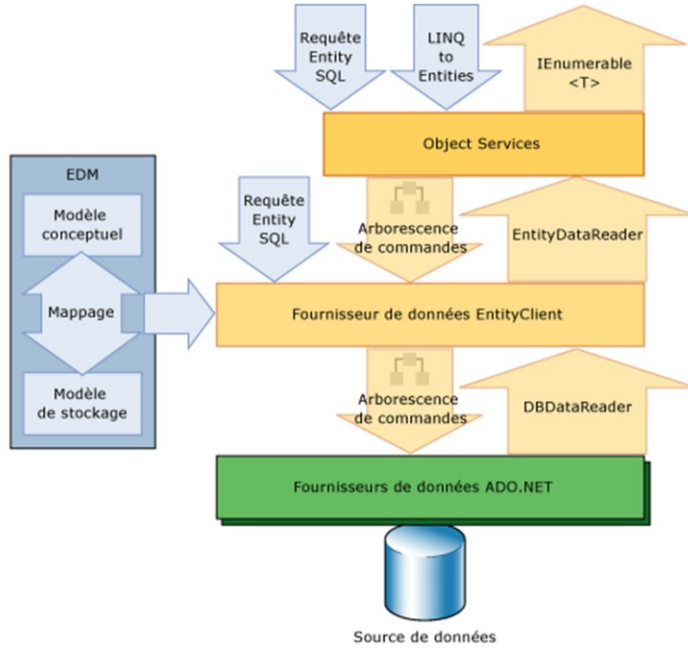
- Entity Framework traduit les opérations LINQ et autres en commandes spécifiques à la source de données
  - Via les services Object Services
- EDM (Entity Data Model) réalise le modèle conceptuel, le modèle de stockage et le mappage entre les deux
  - L'EDM est externe et peut être modifié en fonction des besoins
  - Le mapping par attributs de LINQ To SQL demandait une recompilation
  - Fichier CSDL (Conceptual Schema Definition Language) : définit le modèle conceptuel ;
  - Fichier SSDL (Store Schema Definition Language : définit le modèle de stockage, également appelé modèle logique ;
  - Fichier MSL (Mapping Specification Language) : définit le mappage entre le modèle de stockage et le modèle conceptuel.
  - Fichier EDMX (Entity Data Model Xml) : créé par Visual Studio, il regroupe en un seul document les 3 précédents
- Code first : sans fichier EDM
- Sorti du framework .Net
  - Évolue beaucoup plus vite que .Net…
- Installer un package NuGet dans le projet DLL Métier & dans l'application : `PM> Install-Package EntityFramework`

Mapping

- Les relations sont représentées par des *propriétés de navigation* :
  - Une relation à 1, par une référence
  - Une relation à N par une collection
- Les classes sont partielles
  - Elles peuvent être étendues pour ajouter des méthodes métier
- La classe de base des entités permet leur suivi par l'*Object Services*
  - Matérialisation
  - Suivi des modifications

Architecture

- Architecture Entity Framework



Fournisseurs de données EntityClient

- SQL Server : de base dans .Net 3.5 SP1
- Produits tierces pour les autres BD :

- Tous les fournisseurs à jour :

```
http://msdn.microsoft.com/en-us/data/dd363565.aspx
```

CSDL

- Définit les classes d'entités et d'associations dans le modèle objet généré
  - `EntityType` : type d'entité (table)

```
<EntityType Name="Category">
  <Key> <PropertyRef Name="CategoryID" /> </Key>
  <Property Name="CategoryID" Type="Int32" Nullable="false"
            annotation:StoreGeneratedPattern="Identity" />
  <Property Name="CategoryName" Type="String" MaxLength="15"
            FixedLength="false" Unicode="true" Nullable="false" />
  <Property Name="Description" Type="String" MaxLength="Max"
            FixedLength="false" Unicode="true" />
  <Property Name="Picture" Type="Binary" MaxLength="Max"
            FixedLength="false" />
  <NavigationProperty Name="Products" ToRole="Products"
    Relationship="Self.FK_Products_Categories" FromRole="Categories"/>
</EntityType>
```

CSDL

- `Association` : relation entre table (entités)

```
<Association Name="FK_Products_Categories">
  <End Role="Categories" Type="Self.Category" Multiplicity="0..1" />
  <End Role="Products" Type="Self.Product" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Categories"> <PropertyRef Name="CategoryID" />
    </Principal>
    <Dependent Role="Products"> <PropertyRef Name="CategoryID" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

- `EntityContainer` : tables + relations =DbContext

```
<EntityContainer Name="NorthwindEntities"
                 annotation:LazyLoadingEnabled="true">
  <EntitySet Name="Categories" EntityType="Self.Category" />
  <EntitySet Name="Products" EntityType="Self.Product" />
  <AssociationSet Name="FK_Products_Categories"
                  Association="Self.FK_Products_Categories">
    <End Role="Categories" EntitySet="Categories" />
    <End Role="Products" EntitySet="Products" />
  </AssociationSet>
</EntityContainer>
```

SSDL

- Similaire au CSDL mais SSDL utilise les types du modèle de stockage
  - `EntityType` : type d'entité (table)

```
<Association Name="FK_Products_Categories">
  <End Role="Categories" Type="Self.Category" Multiplicity="0..1" />
  <End Role="Products" Type="Self.Product" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Categories"> <PropertyRef Name="CategoryID" />
    </Principal>
    <Dependent Role="Products"> <PropertyRef Name="CategoryID" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

SSDL

- `Association` : relation entre tables

```
<Association Name="FK_Products_Categories">
  <End Role="Categories" Type="Self.Categories" Multiplicity="0..1" />
  <End Role="Products" Type="Self.Products" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="Categories">
      <PropertyRef Name="CategoryID" />
    </Principal>
    <Dependent Role="Products">
      <PropertyRef Name="CategoryID" />
    </Dependent>
  </ReferentialConstraint>
</Association>
```

- `EntityContainer` : tables + associations = BD

```
<EntityContainer Name="NorthwindModelStoreContainer">
  <EntitySet Name="Categories" EntityType="Self.Categories"
             Schema="dbo" store:Type="Tables" />
  <EntitySet Name="Products" EntityType="Self.Products" Schema="dbo"
             store:Type="Tables" />
  <AssociationSet Name="FK_Products_Categories"
                  Association="Self.FK_Products_Categories">
    <End Role="Categories" EntitySet="Categories" />
    <End Role="Products" EntitySet="Products" />
  </AssociationSet>
</EntityContainer>
```

MSL

- Connecte CSDL et SSDL

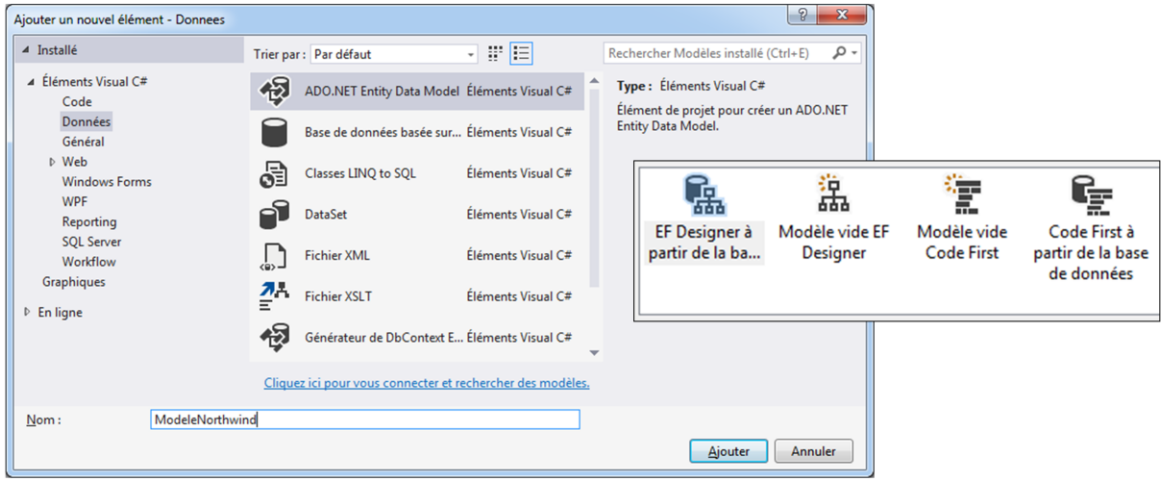
```
<Mapping Space="C-S" xmlns="http://schemas.microsoft.com/ado/2009/11/mapping/cs">
 <EntityContainerMapping
   StorageEntityContainer="NorthwindModelStoreContainer"
   CdmEntityContainer="NorthwindEntities">
   <EntitySetMapping Name="Categories">
    <EntityTypeMapping TypeName="NorthwindModel.Category">
      <MappingFragment StoreEntitySet="Categories"> <ScalarProperty Name="CategoryID" ColumnName="CategoryID" />
        <ScalarProperty Name="CategoryName" ColumnName="CategoryName" />
        <ScalarProperty Name="Description" ColumnName="Description"/>
       <ScalarProperty Name="Picture" ColumnName="Picture" />
      </MappingFragment>
    </EntityTypeMapping>
   </EntitySetMapping>
   <EntitySetMapping Name="Products">
    <EntityTypeMapping TypeName="NorthwindModel.Product">
      <MappingFragment StoreEntitySet="Products">
       <ScalarProperty Name="ProductID" ColumnName="ProductID" />
       <ScalarProperty Name="ProductName" ColumnName="ProductName" />
       <ScalarProperty Name="SupplierID" ColumnName="SupplierID" />
       <ScalarProperty Name="CategoryID" ColumnName="CategoryID" />
       <ScalarProperty Name="QuantityPerUnit" ColumnName="QuantityPerUnit" />
       <ScalarProperty Name="UnitPrice" ColumnName="UnitPrice" />
       <ScalarProperty Name="UnitsInStock" ColumnName="UnitsInStock"/>
       <ScalarProperty Name="UnitsOnOrder" ColumnName="UnitsOnOrder"/>
       <ScalarProperty Name="ReorderLevel" ColumnName="ReorderLevel"/>
       <ScalarProperty Name="Discontinued" ColumnName="Discontinued"/>
      </MappingFragment> </EntityTypeMapping> </EntitySetMapping> </EntityContainerMapping></Mapping>
```

Génération d'un EDM

- Outils Visual Studio
  - Générer un EDM à partir d'une BD existante
  - Générer une BD en partant d'un EDM vide
  - Des fichiers T4 (modifiables) exploitent l'EDM pour faciliter l'adaptation du code généré (entités et DbContext)
- Code first
  - Entités = classes, les tables sont générées à l'aide des métadonnées
  - Relations misent en place par une API ou par convention
- Nécessité d'une clef par table
  - Si une table n'a pas de clef, les outils EDM essaient de déduire une clé pour l'entité correspondante

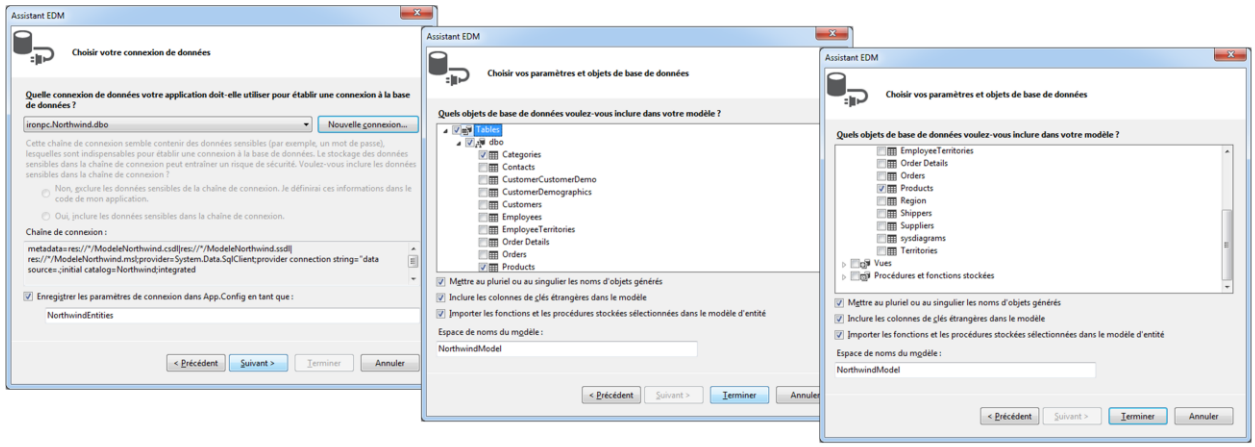
Génération d'un EDM

- Création du fichier edmx
  - En général dans une bibliothèque de classes
  - La chaîne de connexion doit être présente dans le fichier de configuration du programme client



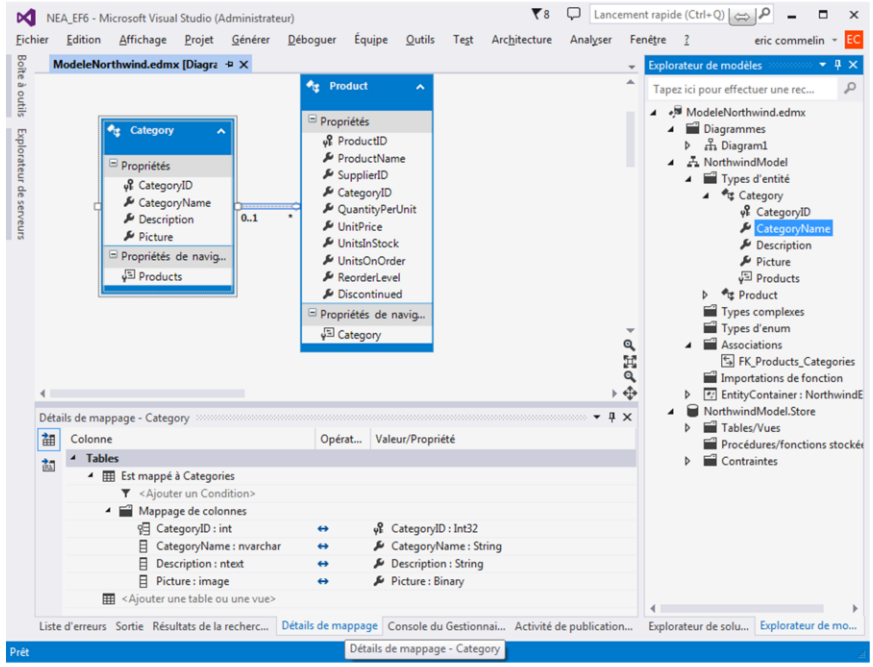
Génération d'un EDM

- Sélection de la base de données et des entités à créer



Génération d'un EDM

- Modification éventuelle du mapping



Ça marche déjà, mais…

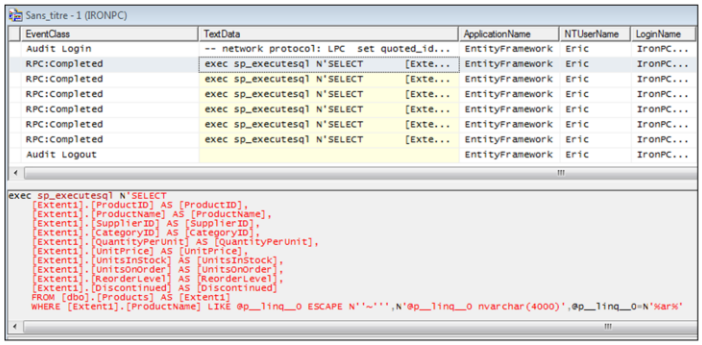
- Combien de requêtes ?

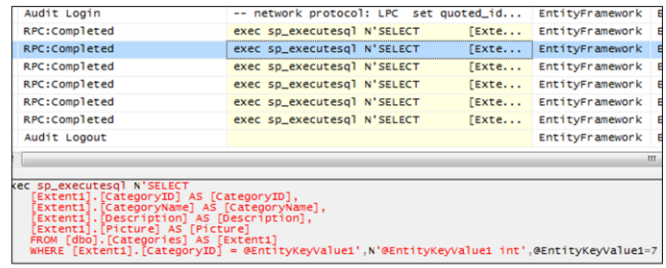
```
Console.Write( "Rechercher : " );
var r = Console.ReadLine();
using( var dc = new NorthwindEntities() ) {
  foreach( var p in dc.Products.Where( p => p.ProductName.Contains( r ) ) ) {
    Console.WriteLine( "{0} ({1})", p.ProductName,
                                    p.Category.CategoryName );
  }
}
```

```
Rechercher : ar
Uncle Bob's Organic Dried Pears (Produce)
Carnarvon Tigers (Seafood)
Sir Rodney's Marmalade (Confections)
Guaraná Fantástica (Beverages)
Mascarpone Fabioli (Dairy Products)
Chartreuse verte (Beverages)
Escargots de Bourgogne (Seafood)
Tarte au sucre (Confections)
Mozzarella di Giovanni (Dairy Products)
Röd Kaviar (Seafood)
```

Ça marche déjà, mais…

- SqlServer Profiler est le meilleur ami du développeur EF !





Propriétés de navigation – Lazy loading

- Le lazy loading peut causer des problèmes de performance (et inversement)
  - Solution : Include

```
dc.Products.Include( "Category" ).Where( . . .
```

```
using System.Data.Entity;
[. . .]
dc.Products.Include( p => p.Category ).Where( . . .
```

  - Solution : l'interdire

```
dc.Configuration.LazyLoadingEnabled = false;
```

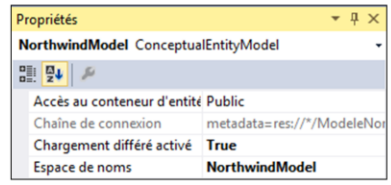
    - Ne pas confondre avec

```
dc.Configuration.ProxyCreationEnabled = false;
```

      - Les proxies permettent le lazy loading mais également le suivi des modifications
      - Mais sans proxy, pas de lazy loading

Propriétés de navigation – Lazy loading

- Personnaliser la classe du DC
  - Lazy loading : par le designer



  - Proxies : personnalisation des tt dans VotreContext.Context.tt

```
<#=Accessibility.ForType(container)#> partial class <#=code.Escape(container)#> : DbContext
{
    public <#=code.Escape(container)#>()
        : base("name=<#=container.Name#>")
    {
      this.Configuration.ProxyCreationEnabled = false;            ← ICI
<#
if (!loader.IsLazyLoadingEnabled(container))
{
#>
        this.Configuration.LazyLoadingEnabled = false;
<#
}
```

Annotation d'entités

- On annote les champs & propriétés
  - Ces membres sont déjà déclarés dans une classe partielle générée par le designer d'entités "data base first"
  - Une classe secondaire doit donc être associée à l'entité. La classe secondaire déclarera à nouveau les propriétés et les annotera
  - Seuls les attributs de la classe "metadata" seront utilisés
  - La classe "metadata" est souvent interne à la classe à annoter

⚠Les champs doivent tout de même être publics !

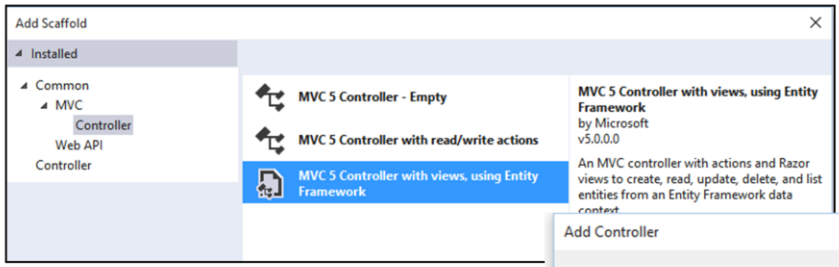
Annotation d'entités

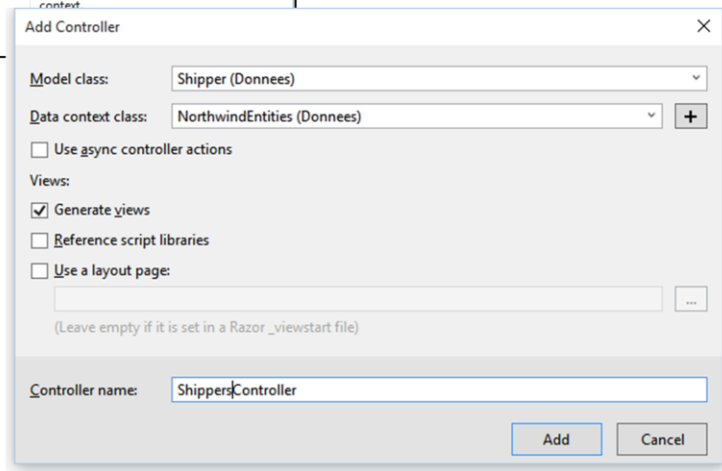
- Exemple

```
[System.ComponentModel.DataAnnotations.MetadataType( typeof( ShipperMetadata ) )]
public partial class Shipper {
  // Les propriétés ont déjà été définies par le designer EF
  internal class ShipperMetadata {    // Suffixe par convention
    [Required( "La compagnie doit avoir un nom" )]
    [MaxLength( 40, "40 Caractères maxi !" )]
    [Display( Name = "Société" )]
    public object CompanyName = null; // Peu importe le type de la
                                      // propriété ou du champ,
                                      // seul son nom compte
  }
}
```

Assistance MVC Visual Studio

- Ajouter un nouveau contrôleur





- Semble prometteur mais n'est que rarement réellement adapté

Assistance MVC Visual Studio

```
public class ShippersController : Controller {
  private NorthwindEntities db = new NorthwindEntities();
  public ActionResult Index() {
    return View( db.Shippers.ToList() );
  }
  public ActionResult Details( int? id ) {
    if( id == null ) {
      return new HttpStatusCodeResult( HttpStatusCode.BadRequest );
    }
    Shipper shipper = db.Shippers.Find( id );
    if( shipper == null ) {
      return HttpNotFound();
    }
    return View( shipper );
  }
  public ActionResult Create() {
    return View();
  }
  [HttpPost]
  [ValidateAntiForgeryToken]
  public ActionResult Create( [Bind( Include = "ShipperID,CompanyName,Phone" )] Shipper shipper ) {
    if( ModelState.IsValid ) {
      db.Shippers.Add( shipper );
      db.SaveChanges();
      return RedirectToAction( "Index" );
    }
    return View( shipper );
  }

==>
```

Assistance MVC Visual Studio

```
public ActionResult Edit( int? id ) {
  if( id == null ) {
    return new HttpStatusCodeResult( HttpStatusCode.BadRequest );
  }
  Shipper shipper = db.Shippers.Find( id );
  if( shipper == null ) {
    return HttpNotFound();
  }
  return View( shipper );
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit( [Bind( Include = "ShipperID,CompanyName,Phone" )] Shipper shipper ) {
  if( ModelState.IsValid ) {
    db.Entry( shipper ).State = EntityState.Modified;
    db.SaveChanges();
    return RedirectToAction( "Index" );
  }
  return View( shipper );
}
public ActionResult Delete( int? id ) {
  if( id == null ) {
    return new HttpStatusCodeResult( HttpStatusCode.BadRequest );
  }
  Shipper shipper = db.Shippers.Find( id );
  if( shipper == null ) {
    return HttpNotFound();
  }
  return View( shipper );
} ==>
```

Assistance MVC Visual Studio

```
  [HttpPost, ActionName( "Delete" )]
  [ValidateAntiForgeryToken]
  public ActionResult DeleteConfirmed( int id ) {
    Shipper shipper = db.Shippers.Find( id );
    db.Shippers.Remove( shipper );
    db.SaveChanges();
    return RedirectToAction( "Index" );
  }
  protected override void Dispose( bool disposing ) {
    if( disposing ) {
      db.Dispose();
    }
    base.Dispose( disposing );
  }
}
```

Assistance MVC Visual Studio

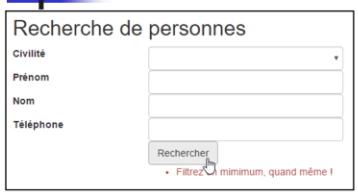
- Les vues sont également rarement adaptées

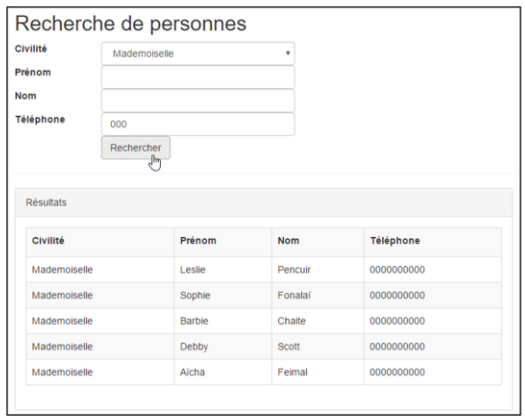
```
@using( Html.BeginForm() ) {
  @Html.AntiForgeryToken()
  <div class="form-horizontal">
    <h4>Shipper</h4> <hr />
    @Html.ValidationSummary( true, "", new { @class = "text-danger" } )
    @Html.HiddenFor( model => model.ShipperID )
     <div class="form-group">
      @Html.LabelFor( model => model.CompanyName, htmlAttributes: new { @class = "control-label col-md-2" } )
      <div class="col-md-10">
        @Html.EditorFor( model => model.CompanyName, new { htmlAttributes = new { @class = "form-control" } } )
        @Html.ValidationMessageFor( model => model.CompanyName, "", new { @class = "text-danger" } )
      </div>
    </div>
     <div class="form-group">
      @Html.LabelFor( model => model.Phone, htmlAttributes: new { @class = "control-label col-md-2" } )
      <div class="col-md-10">
        @Html.EditorFor( model => model.Phone, new { htmlAttributes = new { @class = "form-control" } } )
        @Html.ValidationMessageFor( model => model.Phone, "", new { @class = "text-danger" } )
      </div>
    </div>
     <div class="form-group">
      <div class="col-md-offset-2 col-md-10"> <input type="submit" value="Save" class="btn btn-default" /> </div>
    </div>
  </div>
}
<div> @Html.ActionLink( "Back to List", "Index" ) </div>
```

Exercice - Recherche de personnes

- Partir de la base Annuaire fournie
  - Mettre en place un DBContext EF
    - Les noms des colonnes ne sont pas vraiment standards ni adaptés à EF. Corrigez le tir.
  - Réaliser un formulaire de recherche permettant de filtrer
    - Par titre (liste à choix multiples), nom, prénom et téléphone
    - La table de référence des titres devra être trouvée en cache.
    - Les labels de saisie et les entêtes de colonne doivent être spécifiés dans des annotations.
  - L'interrogation de la base de données passera par une requête Linq plus ou moins riche en fonction des critères saisis ou non par l'utilisateur
    - Les résultats seront également mis en cache

Exercice - Recherche de personnes



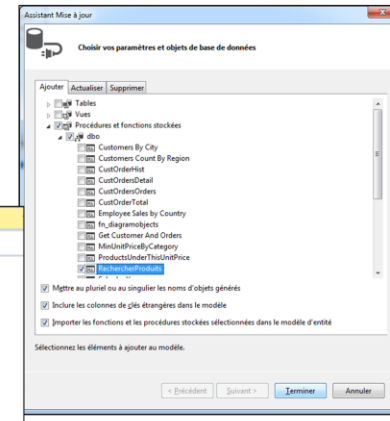


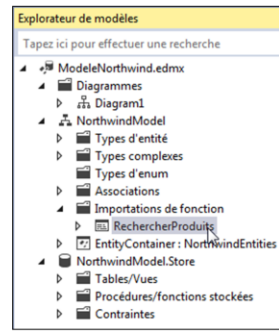
Les procédures stockées

- Faire figurer la procédure dans le modèle

```
ALTER PROCEDURE [dbo].[RechercherProduits]
  @productName nvarchar(40)
AS
BEGIN
  SELECT * FROM Products WHERE ProductName LIKE N'%'+@productName+N'%'
END
```

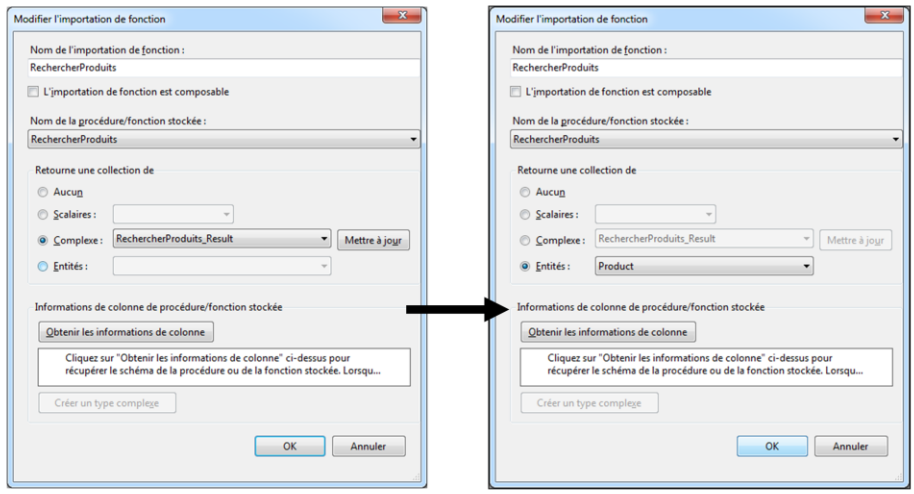
- Ajouter la procédure en faisant un clic droit dans l'éditeur de modèle, choisir Mettre à jour le modèle à partir de la base de données, la procédure apparaît dans l'explorateur de modèle





Les procédures stockées

- Mapper à une méthode et un type
  - Pas mappée automatiquement
    - Double cliquer pour modifier l'importation de fonction



Les procédures stockées

- Utilisation & optimisation
  - Trop de requêtes si lasy loading activé
  - `NullReferenceException` si désactivé

```
foreach( var p in dc.RechercherProduits( "ton" ) )
  Console.WriteLine( "{0} ({1})", p.ProductName, p.Category.CategoryName );
```

  - Correction

```
var mesProduits = dc.RechercherProduits( "ton" ).ToList();
var lesCategoriesId = mesProduits
  .Select( p => p.CategoryID )
  .Distinct()
  .ToList();
var lesCategories = dc.Categories
  .Where( c => lesCategoriesId.Contains( c.CategoryID ) )
  .ToList();          ← Pour forcer l'exécution immédiate
foreach( var p in mesProduits ) {
  Console.WriteLine( "{0} ({1})", p.ProductName,
                                  p.Category.CategoryName );
}
```

Sans_titre - 1 (IRONPC)

EventClass	TextData
Audit Login	-- network protocol: LPC set quote...
RPC:Completed	exec [dbo].[RechercherProduits] @pr...
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quote...
SQL:BatchStarting	SELECT [Extent1].[CategoryID]...
SQL:BatchCompleted	SELECT [Extent1].[CategoryID]...

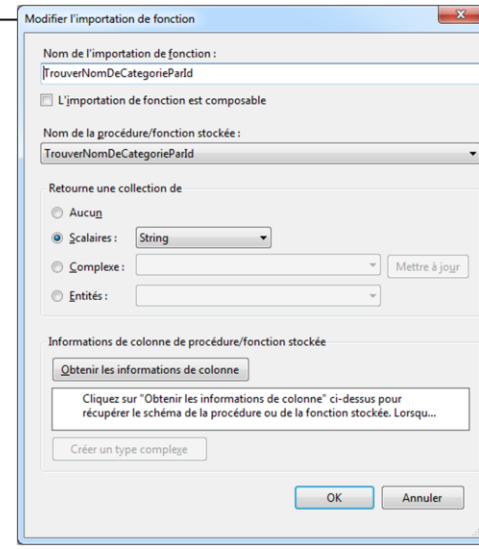
```
SELECT
    [Extent1].[CategoryID] AS [CategoryID],
    [Extent1].[CategoryName] AS [CategoryName],
    [Extent1].[Description] AS [Description],
    [Extent1].[Picture] AS [Picture]
    FROM [dbo].[Categories] AS [Extent1]
    WHERE [Extent1].[CategoryID] IN (2, 6, 8)
```

Les procédures stockées

- Résultats personnalisés et scalaires

```
CREATE PROCEDURE [dbo].[TrouverNomDeCategorieParId]
  @id int
AS
BEGIN
  SELECT c.CategoryName FROM Categories c WHERE c.CategoryID = @id
END
```

  - Importation de fonction : mettre à jour le type du résultat

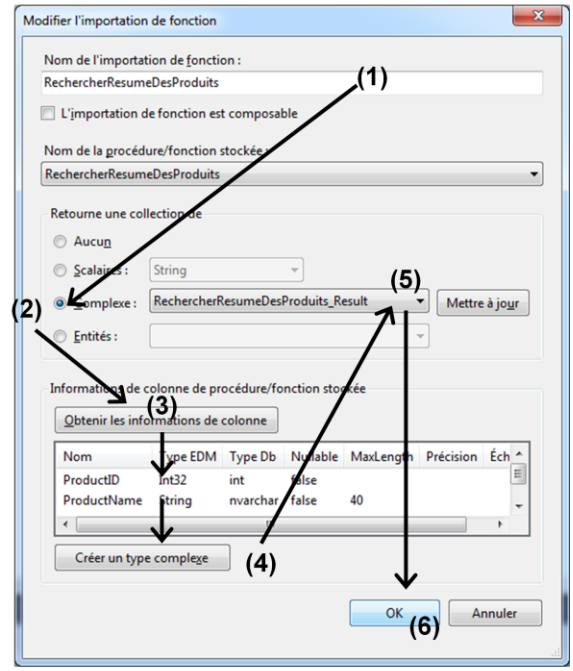


Les procédures stockées

- Idem pour les résultats personnalisés
  - Si la procédure stockée retourne un ou plusieurs résultats ne correspondant pas à un type d'entité

```
foreach( RechercherResumeDesProduits_Result rp
        in dc.RechercherResumeDesProduits( "ar" ) )
  Console.WriteLine( "{0} {1} ({2})", rp.ProductID,
                     rp.ProductName, rp.CategoryName );
```

```
CREATE PROCEDURE dbo.RechercherResumeDesProduits(
                                        @n nvarchar(40) ) AS
  SELECT p.ProductID, p.ProductName, c.CategoryName
  FROM Products p
  INNER JOIN Categories c ON p.CategoryID = c.CategoryID
  WHERE ProductName LIKE '%' + @n + '%'
RETURN
```



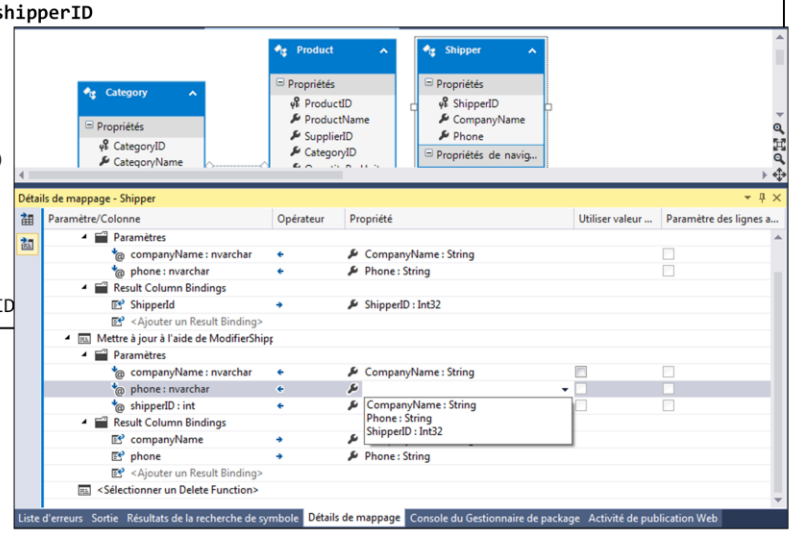
Les procédures stockées - CRUD

- Utiliser des procédures stockées à la place des commandes de mise à jour
  - Dans le détail de mapping, mapper une entité à des fonctions
  - Choisir pour chaque opération Delete, Insert et Update les procédures stockées correspondantes
  - Mapper les paramètres aux propriétés (valeurs actuelles ou valeurs originales) de l'entité

Les procédures stockées - CRUD

- Exemple

```
CREATE PROCEDURE dbo.ModifierShipper(
  @companyName nvarchar(40),
  @phone nvarchar(24),
  @shipperID int ) AS
    UPDATE Shippers SET CompanyName=@companyName + 'OK', Phone=@phone
           WHERE ShipperID=@shipperID
    SELECT * FROM Shippers WHERE ShipperID=@shipperID

CREATE PROCEDURE dbo.AjouterShippers(
  @companyName nvarchar(40),
  @phone nvarchar(24)
  ) AS
    INSERT INTO Shippers (CompanyName, Phone)
      VALUES (@companyName, @phone)
    SELECT @@IDENTITY AS ShipperID

CREATE PROCEDURE dbo.EffacerShipper(
                  @shipperID int ) AS
    DELETE Shippers WHERE ShipperID=@shipperID
```



Les procédures stockées - CRUD

- Utilisation

```
Shipper s = dc.Shippers.First();
s.CompanyName += "X";
dc.SaveChanges();
Console.WriteLine( s.CompanyName );                                    ← Speedy ExpressXOK
dc.Entry( s ).State = System.Data.Entity.EntityState.Detached;
s = dc.Shippers.First();
Console.WriteLine( s.CompanyName );                                    ← Speedy ExpressXOK

s = new Shipper() { CompanyName = "Pinpon Express", Phone = "18" };
dc.Shippers.Add( s );
Console.WriteLine( s.ShipperID );                                      ← 0
dc.SaveChanges();
Console.WriteLine( s.ShipperID );                                      ← 4
```

Classes d'entités et logique métier

- Extrait du code généré

```
public partial class Category {
  public Category() {
    this.Products = new HashSet<Product>();
  }
  public int CategoryID { get; set; }
  public string CategoryName { get; set; }
  public string Description { get; set; }
  public byte[] Picture { get; set; }
  public virtual ICollection<Product> Products { get; set; }
}
```

- La classe est très simple
  - On peut facilement ajouter des propriétés calculées (PrixTTC, ...)
  - On peut facilement ajouter des méthodes métier
- La classe est trop simple
  - Impossible de tester les valeurs affectées aux propriétés

Classes d'entités et logique métier

- Le modèle T4 est trop simple :

```
public string Property(EdmProperty edmProperty) {
  return string.Format(
    CultureInfo.InvariantCulture,
    "{0} {1} {2} {{ {3}get; {4}set; }}",
    Accessibility.ForProperty(edmProperty),
    _typeMapper.GetTypeName(edmProperty.TypeUsage),
    _code.Escape(edmProperty),
    _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
    _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
}
```

  - On peut personnaliser la façon dont la classe est générée
    - C'est là tout l'intérêt de passer par T4

Classes d'entités et logique métier

- Modification de ModeleNorthwind.tt

```
public string Property(EdmProperty edmProperty) {
  return string.Format(
    CultureInfo.InvariantCulture,
    @"partial void On{2}Changing( {1} nvValeur );
    private {1} _{2};
    {0} {1} {2} {{
      {3}get {{ return _{2}; }}
      {4}set {{
        On{2}Changing( value );
        _{2} = value;
      }}
    }}",
    Accessibility.ForProperty(edmProperty),
    _typeMapper.GetTypeName(edmProperty.TypeUsage),
    _code.Escape(edmProperty),
    _code.SpaceAfter(Accessibility.ForGetter(edmProperty)),
    _code.SpaceAfter(Accessibility.ForSetter(edmProperty)));
}
```

CategoryName devient

```
partial void OnCategoryNameChanging( string nvValeur );
private string _CategoryName;
public string CategoryName {
  get { return _CategoryName; }
  set {
    OnCategoryNameChanging( value );
    _CategoryName = value;
  }
}
```

Classes d'entités et logique métier

- Application à la mise en œuvre de `INotifyPropertyChanging`

```
using System.ComponentModel;
partial class Category : INotifyPropertyChanging {
  public event PropertyChangingEventHandler PropertyChanging;
  partial void OnCategoryNameChanging( string nvValeur ) {
    if( PropertyChanging != null )
      PropertyChanging( this, new PropertyChangingEventArgs( "CategoryName" ) );
  }

  // Pas forcément nécessaire de moe INotifyPropertyChanging :
  partial void OnDescriptionChanging( string nvValeur ) {
    if( nvValeur != null && nvValeur.Length < 5 ) {
      throw new ArgumentException( "Si description il y a, celle-ci doit faire au moins 5 caractères" );
    }
  }
}
```

Accès aux entités en écriture

- EF gère l'état des objets via un `DbChangeTracker`
  - Assure l'unicité des objets en mémoire
  - Détient la liste des objets modifiés en mémoire
    - Liste consultable
    - Liste modifiable
      - Concept d'attachement
  - Peut être contourné pour créer des objets "détachés"

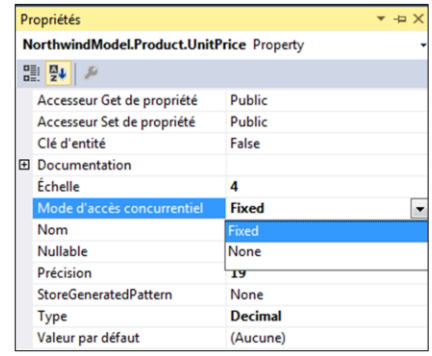
```
dc.Categories.AsNoTracking().Where( ... );
```

Accès aux entités en écriture

- Utilisation
  - Les entités sont modifiées en mémoire
    - Le contexte fait les mises à jour en base à la demande
    - Il est possible d'accéder aux valeurs originales et modifiées
      - Les valeurs originales peuvent être explicitement remplacées par les nouvelles
  - Les modifications apportées à un objet peuvent être annulées en le "détachant" de son contexte
  - Un objet de type entité peut être attaché au contexte
    - Objets sauvés dans le `ViewState` d'ASP.Net WebForms
    - Objets retournés pas un service distant

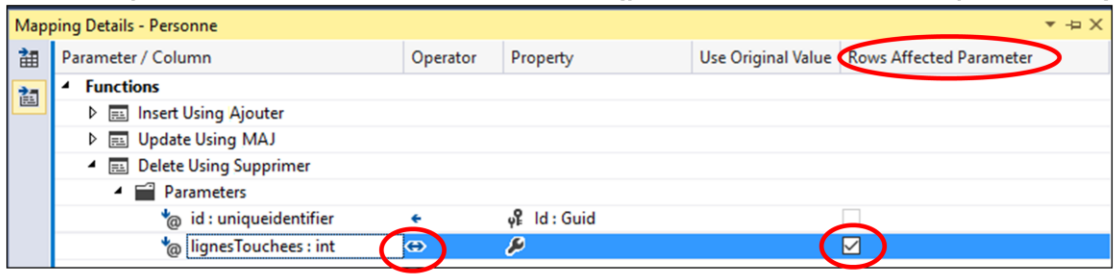
Gestion des accès concurrents

- Gestion des accès concurrents
  - EF permet de préciser, pour chaque colonne, si un test de conflit d'accès concurrent optimiste est requis
    - Une exception est alors générée lors de la tentative de mise à jour
      - L'exception est soulevée dès la première erreur
      - La mise à jour est traitée dans le cadre d'une transaction
  - Le conflit peut être résolu
    - En perdant les valeurs en mémoire
    - En acceptant les valeurs en base comme valeurs originales



Gestion des accès concurrents

- Gestion des accès concurrents par des procédure stockées pour les actions CRUD
  - La lecture se fait en utilisant une vue retournant des entités à la place d'une table
  - Pas de "fixed" si des procédures stockées sont utilisées
    - Celle-ci doivent retourner le nombre d'enregistrements touchés pour permettre la détection du conflit (paramètre "out" de la procédure)



Gestion des accès concurrents

- Principales API
  - Du `DbContext`

```
DbEntityEntry Entry( object )
DbEntityEntry<TEntity> Entry<TEntity>( TEntity entity )
DbChangeTracker ChangeTracker { get; }
```

  - De son `ChangeTracker`

```
void DbChangeTracker.DetectChanges()
IEnumerable<DbEntityEntry> Entries()
IEnumerable<DbEntityEntry<T>> Entries<T>()
bool HasChanges()
```

  - Des `DbUpdateConcurrencyException`

```
IEnumerable<DbEntityEntry> Entries      ← En fait il n'y en aura qu'une…
```

  - Des `DbEntityEntry<TEntity>`

```
public DbPropertyValues CurrentValues { get; }
DbPropertyValues OriginalValues { get; }
TEntity Entity { get; }
EntityState State { get; set; }                    ← Added Deleted Detached Modified Unchanged
DbPropertyValues GetDatabaseValues()               ← Rend null si la clé n'est plus présente en base
void Reload()
```

    - `DbEntityEntry` (non-générique) existe également

Gestion des accès concurrents - Exemple

- D'abord, provoquer le pb

```
var chai = dc.Products.First( p => p.ProductName == "Chai" );
chai.UnitPrice = 1; // 18 => 1
var tofu = dc.Products.First( p => p.ProductName == "Tofu" );
tofu.UnitPrice = 1; // 23.25 => 1
using( var qq1DAutre = new NorthwindEntities() ) {
  var chai2 = qq1DAutre.Products.First( p => p.ProductName == "Chai" );
  chai2.UnitPrice = 2;
  var tofu2 = qq1DAutre.Products.First( p => p.ProductName == "Tofu" );
  tofu2.UnitPrice = 2;
  var nb = qq1DAutre.SaveChanges(); // Passent à 2$
  Console.WriteLine( "QQ1 fait {0} changements", nb );
}
```

Gestion des accès concurrents - Exemple

```
bool yAEuUnPb;
do {
  try {
    yAEuUnPb = false;
    int nb = dc.SaveChanges();
    Console.WriteLine( "C'est fini : {0} écriture(s) en base", nb );
  }
  catch( DbUpdateConcurrencyException erreur ) {   // Une erreur à la fois
    Console.WriteLine( "Contrôle d'accès concurrent : {0} erreur(s)", erreur.Entries.Count() );
    var pb = erreur.Entries.Single();              // Toujours 1 seule erreur
    Console.WriteLine( "Entité : {0}, {1}", pb.Entity.GetType().Name, pb.State );
    Console.WriteLine( "  - Nom : {0}", ((Product)pb.Entity).ProductName );
    Console.WriteLine( "  - Prix original : {0}", pb.OriginalValues["UnitPrice"] );
    Console.WriteLine( "  - Prix voulu : {0}", pb.CurrentValues["UnitPrice"] );
    var enBase = pb.GetDatabaseValues();   ← OK… Il faudrait aussi tester null : a été effacé en base… La flemme…
    Console.WriteLine( "  - En base : {0}", enBase["UnitPrice"] );
    Console.Write( "Écrire (e) ou abandonner l'édition (autre) ? " );
    if( Console.ReadLine() == "e" )
      pb.OriginalValues.SetValues( enBase );       // Je retenterai la MàJ
    else
      pb.Reload();                                 // Perte des modifs, l'entité est Unchanged
    Console.WriteLine( "Et un pb de réglé..." );
    yAEuUnPb = true;
  }
} while( yAEuUnPb );
dc.Entry( chai ).State = EntityState.Detached;
Console.WriteLine( "Chai : {0}", dc.Products.Find( chai.ProductID ).UnitPrice );
dc.Entry( tofu ).State = EntityState.Detached;
Console.WriteLine( "Tofu : {0}", dc.Products.Find( tofu.ProductID ).UnitPrice );
```

Gestion des accès concurrents - Exemple

- Traces

```
QQ1 fait 2 changements
Contrôle d'accès concurrent : 1 erreur(s)
Entité : Product, Modified
  - Nom : Chai
  - Prix original : 18,0000
  - Prix voulu : 1
  - En base : 2,0000
Écrire en base (e) ou abandonner l'édition (autre) ? a
Et un pb de réglé...
Contrôle d'accès concurrent : 1 erreur(s)
Entité : Product, Modified
  - Nom : Tofu
  - Prix original : 23,2500
  - Prix voulu : 1
  - En base : 2,0000
Écrire (e) ou abandonner l'édition (autre) ? e
Et un pb de réglé...
C'est fini : 1 écriture(s) en base
Chai : 2,0000
Tofu : 1,0000
```

Attachement

- Principe
  - Permet d'ajouter une entité au `ChangeTracker`
  - L'entité ayant été obtenue "autrement"
    - Paramètres d'une méthode ;
    - Champs cachés ;
    - Clé(s) primaire(s) connue(s) ;
    - ...
  - Peut permettre d'éviter de relire en base avant une mise à jour ou une suppression
  - Changer le `EntityState` d'une nouvelle `DbEntityEntry` créée à l'état `Unchanged` simplement en la recherchant

Attachement

- Exemple ASP.Net avec Model Binding

```
@using WebApplication1.Models
@model IEnumerable<ProductPourListeViewModel>
@{ ViewBag.Title = "Exemple d'attachement"; }
<h2>@ViewBag.Title</h2>
<p>@ViewBag.Message</p>
<table class="table table-striped">
  <tr>
    <th>Action</th>
    <th>Catégorie</th>
    <th>Produit</th>
    <th>Prix U. (USD)</th>
  </tr>
  @foreach( var p in Model ) {
    <tr>
      <td>
        @using( Html.BeginForm( "Supprimer", "ExempleAttachement" ) ) {
          <input type="submit" value="Supprimer" class="btn btn-danger" />
          <input type="hidden" name="ProductID" value="@p.ProductID" />
          <input type="hidden" name="UnitPrice" value="@p.UnitPrice" />
        }
      </td>
      <td>@p.CategoryName</td>
      <td>@p.ProductName</td>
      <td>@($"{p.UnitPrice:0.00}")</td>
    </tr>
  }
</table>
```

```
public class ProductPourListeViewModel {
  public int ProductID { get; set; }
  public string CategoryName { get; set; }
  public string ProductName { get; set; }
  public decimal? UnitPrice { get; set; }
}
```

Exemple d'attachement

Action	Catégorie	Produit	Prix U. (USD)
Supprimer	Beverages	La gnôle à Papy, 95°	2,00
Supprimer	Dairy Products	Geitost	2,50

Attachement - Contrôleur

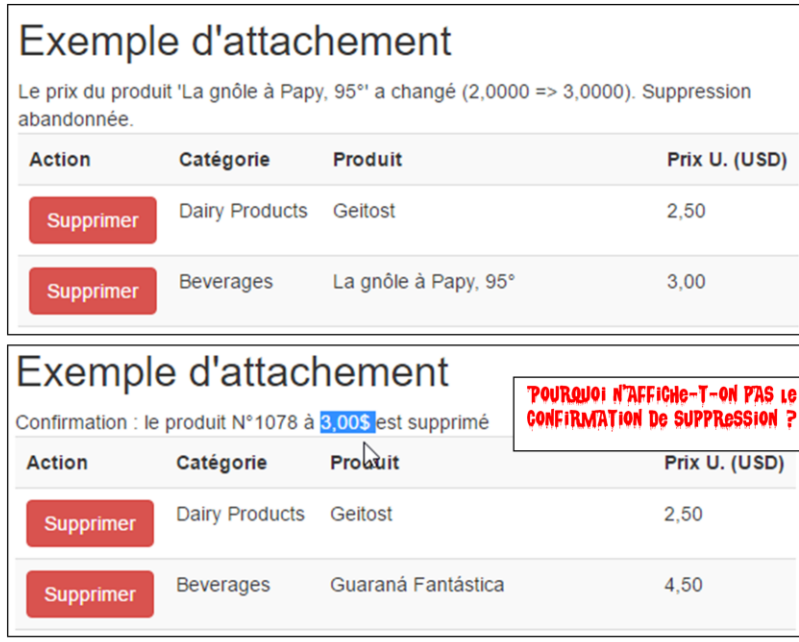
POURQUOI NE PAS UTILISER PRODUCT COMME VIEW MODEL ?

POURQUOI NE PAS FAIRE UN .TOLIST ?

```
public class ExempleAttachementController : Controller {
  private NorthwindEntities DC = new NorthwindEntities();
  public ActionResult Index() {
    var products = DC.Products.OrderBy( p => p.UnitPrice ).Select( p =>
      new ProductPourListeViewModel() { ProductID = p.ProductID, CategoryName = p.Category.CategoryName, [...]} );
    return View( "Index", products );
  }
  public ActionResult Supprimer( Product victime ) {
    if( Request.HttpMethod == "GET" || victime.ProductID == 0 ) return Index();
    DC.Entry<Product>( victime ).State = System.Data.Entity.EntityState.Deleted;
    try {
      DC.SaveChanges();
      ViewBag.Message = String.Format("Confirmation : le produit N°{0} à {1:0.00}$ est supprimé",
                                      victime.ProductID, victime.UnitPrice );
      return Index();
    }
    catch( DbUpdateConcurrencyException cac ) {
      var erreur = cac.Entries.Single();
      var nvValeurs = erreur.GetDatabaseValues(); ← OK… OK… Il faudrait aussi tester null : a été effacé en base…
      ViewBag.Message = string.Format( "Le prix du produit '{0}' a changé ({1} => {2}). Suppression abandonnée.",
                                       nvValeurs["ProductName"], victime.UnitPrice, nvValeurs["UnitPrice"] );
      DC.Entry( victime ).State = System.Data.Entity.EntityState.Detached;
      return Index();
    }
  }
  protected override void Dispose( bool disposing ) {
    if( disposing ) DC.Dispose();
    base.Dispose( disposing );
  }
}
```

Attachement

- Traces



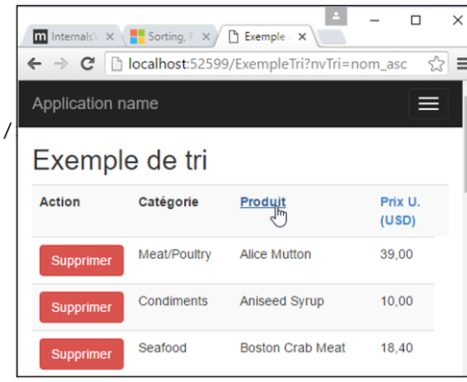
Tri

- Ce qu'on va faire
  - Trier…
  - Assurer le maintien de l'ordre de tri :
    - Passé à la vue par le ViewBag
    - Passé au contrôleur par l'URL ou un champ caché
      - Permet d'assurer l'inversion si la même colonne est recliquée
  - Un Html.BeginForm ne conserve la `QueryString` de l'URL d'origine **que** s'il conserve l'URL d'origine, càd si on ne précise pas d'action

Tri

- Vue (extrait)

```
<table class="table table-striped">
  <tr>
    <th>Action</th>
    <th>Catégorie</th>
    <th>@Html.ActionLink( "Produit", "Index",
                          new { tri = ViewBag.Tri == "nom_asc" ? "nom_desc"
                                                               : "nom_asc" } )
    </th>
    <th>@Html.ActionLink( "Prix U. (USD)", "Index", new { tri = ViewBag.Tri == "prix_asc" ? "prix_desc" :
                                                                                            "prix_asc" } )</th>
  </tr>
  @foreach( var p in Model ) {
    <tr>
      <td>
        @using( Html.BeginForm( "Supprimer", "ExempleTri" ) ) {
          <input type="submit" value="Supprimer" class="btn btn-danger" />
          <input type="hidden" name="ProductID" value="@p.ProductID" />
          <input type="hidden" name="UnitPrice" value="@p.UnitPrice" />
          <input type="hidden" name="tri" value="@ViewBag.Tri" />
        }
      </td>
      <td>@p.CategoryName</td>
      <td>@p.ProductName</td>
      <td>@($"{p.UnitPrice:0.00}")</td>
    </tr>
  }
</table>
```



Tri

- Contrôleur – extraits 1

```
public ActionResult Index( string tri ) {
  var products = DC.Products.AsQueryable();                    ← Un IQueryable<Product>, pas un DbSet<Product>
  switch( tri ) {
    case "nom_desc":
      products = products.OrderByDescending( p => p.ProductName );
      break;
    case "prix_asc":
      products = products.OrderBy( p => p.UnitPrice );
      break;
    case "prix_desc":
      products = products.OrderByDescending( p => p.UnitPrice );
      break;
    default:
      products = products.OrderBy( p => p.ProductName );
      tri = "nom_asc";
      break;
  }
  ViewBag.Tri = tri;
  var viewModel = products.Select( p =>
    new ProductPourListeViewModel() {
      ProductID = p.ProductID,
      CategoryName = p.Category.CategoryName,
      ProductName = p.ProductName,
      UnitPrice = p.UnitPrice } );
  return View( "Index", viewModel );
}
```

Tri

- Contrôleur – extraits 2

```
public ActionResult Supprimer( Product victime, string tri ) {
  if( Request.HttpMethod == "GET" || victime.ProductID == 0 )
    return Index( tri );
  victime.UnitPrice = 1234.01m;                                          ← Tricheur !
  DC.Entry<Product>( victime ).State = System.Data.Entity.EntityState.Deleted;
  try {
    DC.SaveChanges();
    ViewBag.Message = String.Format("Confirmation : le produit N°{0} à {1:0.00}$ est supprimé",
                                    victime.ProductID, victime.UnitPrice );
    return Index( tri );
  }
  catch( DbUpdateConcurrencyException cac ) {
    var erreur = cac.Entries.Single();
    var nvValeurs = erreur.GetDatabaseValues();
    ViewBag.Message = string.Format(
      "Le prix du produit '{0}' a changé ({1} => {2}). Suppression abandonnée.",
      nvValeurs["ProductName"], victime.UnitPrice, nvValeurs["UnitPrice"] );
    DC.Entry( victime ).State = System.Data.Entity.EntityState.Detached;
    return Index( tri );
  }
}
```

Filtrage

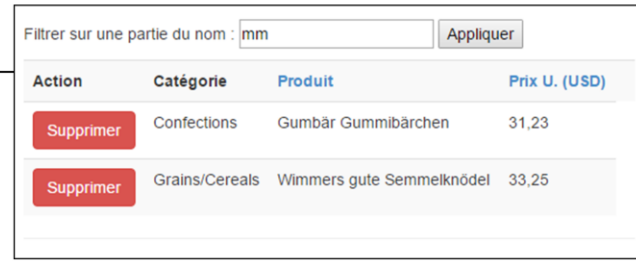
- L'ordre de tri et le filtre sont maintenus
  - Passés à la vue par le ViewBag
  - Passés au contrôleur par l'URL ou un champ caché
- Vue (extraits)
  - Utiliser un GET permet de supporter les marques-page sur un résultat de recherche mais oblige à ajouter le paramètre de tri

```
@using( Html.BeginForm("Index", "ExempleFiltre", FormMethod.Get) ) {
  <p>
    Filtrer sur une partie du nom : <input type="text" name="filtre" value="@ViewBag.Filtre" />
    <input type="submit" value="Appliquer" />
  </p>
  <input type="hidden" name="tri" value="@ViewBag.Tri" />
}

<p>@ViewBag.Message</p>
```

Filtrage

- Vue, début
  - Passer le filtre dans le lien avec `ActionLink`

```
<table class="table table-striped">
  <tr>
    <th>Action</th>
    <th>Catégorie</th>
    <th>@Html.ActionLink( "Produit", "Index",
                          new {
                            filtre = ViewBag.Filtre,
                            tri = ViewBag.Tri == "nom_asc" ? "nom_desc" : "nom_asc"
                          } )
    </th>
    <th>@Html.ActionLink( "Prix U. (USD)", "Index",
                         new {
                           filtre = ViewBag.Filtre,
                           tri = ViewBag.Tri == "prix_asc" ? "prix_desc" : "prix_asc"
                         } )
    </th>
  </tr>
```



Filtrage

- Vue, suite et fin
  - Envoyer le filtre dans les entêtes lors d'un POST avec des champs (ne pollue pas l'URL) cachés ou avec les options de `BeginForm` (pollue l'URL)

```
@foreach( var p in Model ) {
   <tr>
     <td>
       @using( Html.BeginForm( "Supprimer", "ExempleFiltre",
                               new {filtre = ViewBag.Filtre} )){                    ← Au choix
         <input type="submit" value="Supprimer" class="btn btn-danger" />
         <input type="hidden" name="ProductID" value="@p.ProductID" />
         <input type="hidden" name="UnitPrice" value="@p.UnitPrice" />
         <input type="hidden" name="tri" value="@ViewBag.Tri" />
         @*<input type="hidden" name="filtre" value="@ViewBag.Filtre" />*@          ← Au choix
       }
     </td>
     <td>@p.CategoryName</td>
     <td>@p.ProductName</td>
     <td>@($"{p.UnitPrice:0.00}")</td>
   </tr>
  }
</table>
```

Filtrage - Contrôleur

```
public ActionResult Index( string tri, string filtre ) {
  var products = DC.Products.AsQueryable();
  if( !string.IsNullOrWhiteSpace( filtre ) ) {
    filtre = filtre.Trim(); ViewBag.Filtre = filtre;
    products = products.Where( p => p.ProductName.Contains( filtre ) );
  }
  switch( tri ) { [ . . . ] }
  ViewBag.Tri = tri;
  var viewModel = products.Select( p =>
   new ProductPourListeViewModel() { ProductID = p.ProductID, CategoryName = p.Category.CategoryName, [ . . . ] );
  return View( "Index", viewModel );
}
public ActionResult Supprimer( Product victime, string tri, string filtre ) {
  if( Request.HttpMethod == "GET" || victime.ProductID == 0 ) return Index( tri, filtre );
  victime.UnitPrice = 1234.01m;                                                    ← Tricheur !
  DC.Entry<Product>( victime ).State = System.Data.Entity.EntityState.Deleted;
  try {
    DC.SaveChanges();
    ViewBag.Message = String.Format("Confirmation : le produit N°{0} à {1:0.00}$ est supprimé",
                                    victime.ProductID, victime.UnitPrice );
    return Index( tri, filtre );
  }
  catch( DbUpdateConcurrencyException cac ) {
    var erreur = cac.Entries.Single();
    var nvValeurs = erreur.GetDatabaseValues();
    ViewBag.Message = string.Format( "Le prix du produit '{0}' a changé ({1} => {2}). Suppression abandonnée.",
                                     nvValeurs["ProductName"], victime.UnitPrice, nvValeurs["UnitPrice"] );
    DC.Entry( victime ).State = System.Data.Entity.EntityState.Detached;
    return Index( tri, filtre );
  }
}
```

Pagination

- Gare aux performances
  - Il n'est pas recommandé d'utiliser directement le `IQueryable` du `DbContext` dans la vue
  - Il est recommandé d'utiliser le cache d'Asp.Net
  - Linq permet tout aussi bien de trier et filtrer sur une collection en mémoire que sur un `IQueryable` issu du DbContext
- Conserver le numéro de page ?
  - Lors d'une suppression réussie ou non
  - Pas lors d'un tri
  - Pas lors d'un filtrage
- Conserver le filtre et le tri lors des changements de page

Pagination – Contrôleur, début

```
public class ExemplePaginationController : Controller {
  private NorthwindEntities DC = new NorthwindEntities();
  private const int TaillePage = 3;
  public ActionResult Index( string tri, string filtre, int page = 0, bool forcerLaRequeteEnBase = false ) {
    string clé = "Produits/" + filtre;
    var produitsEnCache = forcerLaRequeteEnBase ? null : (IList<ProductPourListeViewModel>)HttpContext.Cache[clé];
    filtre = (filtre ?? string.Empty).Trim();
    if( produitsEnCache == null ) {
      ViewBag.Message += " Pas en cache !";
      var rqEnBase = DC.Products.AsQueryable();      ← Comme ça j'ai un IQueryable<Product>, pas un DbSet<Product>
      if( !string.IsNullOrWhiteSpace( filtre ) )
        rqEnBase = rqEnBase.Where( p => p.ProductName.Contains( filtre ) );
      produitsEnCache = rqEnBase.Select( p => new ProductPourListeViewModel() {[...]} ).ToList();
      HttpContext.Cache[clé] = produitsEnCache;
    }
    else ViewBag.Message += " En cache !";
    IEnumerable<ProductPourListeViewModel> produitsTrieesEtPaginés;
    switch( tri ) {
      case "nom_desc":
        produitsTrieesEtPaginés = produitsEnCache.OrderByDescending( p => p.ProductName );
        break;
      // [... Comme précédement ...]
    }
    produitsTrieesEtPaginés = produitsTrieesEtPaginés.Skip( page * TaillePage ).Take( TaillePage );
    ViewBag.Filtre = filtre;
    ViewBag.Tri = tri;
    ViewBag.Page = page;
    ViewBag.NbProduits = produitsEnCache.Count;
    ViewBag.TaillePage = TaillePage;
    return View( "Index", produitsTrieesEtPaginés );
  }
```

Pagination – Contrôleur, suite

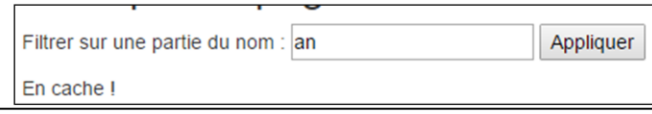
- La suppression conserve la page mais annule le cache, quoi qu'il arrive

```
public ActionResult Supprimer( Product victime, string tri, int page, string filtre ) {
  if( Request.HttpMethod == "GET" || victime.ProductID == 0 )
    return Index( tri, filtre, page );
  victime.UnitPrice = 1234.01m;                                    ← Toujours le même tricheur !
  DC.Entry<Product>( victime ).State = System.Data.Entity.EntityState.Deleted;
  try {
    DC.SaveChanges();
    ViewBag.Message = String.Format("Confirmation : le produit N°{0} à {1:0.00}$ est supprimé",
                                    victime.ProductID, victime.UnitPrice );
    return Index( tri, filtre, page, forcerLaRequeteEnBase: true );
  }
  catch( DbUpdateConcurrencyException cac ) {
    var erreur = cac.Entries.Single(); var nvValeurs = erreur.GetDatabaseValues();
    ViewBag.Message = string.Format(
      "Le prix du produit '{0}' a changé ({1} => {2}). Suppression abandonnée.",
      nvValeurs["ProductName"], victime.UnitPrice, nvValeurs["UnitPrice"] );
    DC.Entry( victime ).State = System.Data.Entity.EntityState.Detached;
    return Index( tri, filtre, page, forcerLaRequeteEnBase: true );
  }
 }
 protected override void Dispose( bool disposing ) {
  if( disposing ) DC.Dispose();
  base.Dispose( disposing );
 }
}
```

Pagination

- Vue, début
  - Le `ViewBag` devient trop rempli pour rester utilisable directement

```
@using WebApplication1.Models
@model IEnumerable<ProductPourListeViewModel>
@{
  ViewBag.Title = "Exemple de pagination";
  int page = ViewBag.Page;
  int nbProduits = ViewBag.NbProduits;
  int taillePage = ViewBag.TaillePage;
  int nbPages = nbProduits / taillePage + (nbProduits % taillePage == 0 ? 0 : 1);
  string tri = ViewBag.Tri;
  string filtre = ViewBag.Filtre;
  string message = ViewBag.Message;
}
<h2>@ViewBag.Title</h2>
```



  - Filtrer ne maintient pas la pagination
    - ⚠ Un post maintient la `QueryString` si on ne la contredit pas

```
@using( Html.BeginForm( new { page = 0, tri = tri } ) ) {                ← Sans, la QueryString serait conservée
  <p>Filtrer sur une partie du nom : <input type="text" name="filtre" value="@filtre" />
     <input type="submit" value="Appliquer" /></p>
}
<p>@message</p>
```

Pagination

- Vue, suite
  - Pagination par un bouton / lien par page
  - Filtre et tri sont maintenus

```
<p>Page N°@(page + 1) / @nbPages - @nbProduits produit(s)</p>
<p>
  @for( int p = 0; p < nbPages; p++ ) {
    if( p == page ) {
      <a class="btn btn-primary">@(p + 1)</a>
    }
    else {
      @Html.ActionLink(
        (p + 1).ToString(),
        "Index",
        new {
          page = p,
          tri = tri,
          filtre = filtre },
        new { @class = "btn btn-default" } );
    }
    @Html.Raw(" ")
  }
</p>
<hr />
```

Pagination

- Pagination précédent / suivant

```
@if( page == 0 ) {
  <a class="btn btn-default" disabled>|&lt;</a>
  <a class="btn btn-default" disabled>&lt;</a>
}
else {
  <a class="btn btn-default" href="@Url.Action( "Index", new { page = 0, tri = tri, filtre = filtre } )">|&lt;</a>
  <a class="btn btn-default" href="@Url.Action( "Index", new { page = page - 1, tri = tri,
                                                              filtre = filtre } )">&lt;</a>
}
@if( page == nbPages - 1 ) {
  <a class="btn btn-default" disabled>&gt;</a>
  <a class="btn btn-default" disabled>&gt;|</a>
}
else {
  <a class="btn btn-default" href="@Url.Action( "Index", new { page = page + 1, tri, filtre = filtre } )">&gt;</a>
  <a class="btn btn-default" href="@Url.Action( "Index", new { page = nbPages - 1, tri,
                                                              filtre = filtre } )">&gt;|</a>
}
```

Pagination

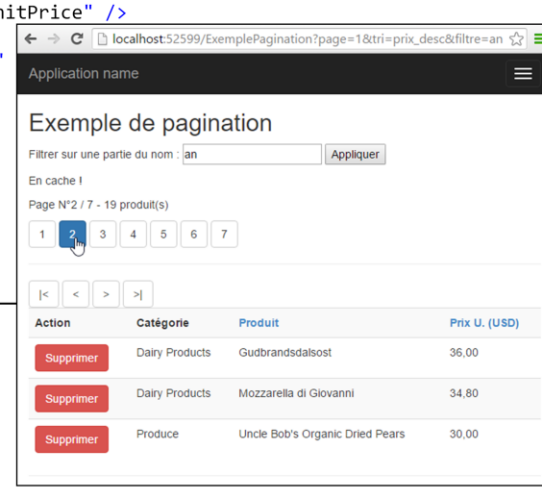
- Vue, suite
  - Les tris ne maintiennent pas la page courante

```
<table class="table table-striped">
  <tr>
    <th>Action</th>
    <th>Catégorie</th>
    <th>@Html.ActionLink( "Produit", "Index",
                         new { filtre = filtre, tri = tri == "nom_asc" ? "nom_desc" : "nom_asc" } )</th>
    <th>@Html.ActionLink( "Prix U. (USD)", "Index",
                         new { filtre = filtre, tri = tri == "prix_asc" ? "prix_desc" : "prix_asc" } )</th>
  </tr>
```

Pagination

- Vue, suite (et fin)
  - La suppression maintient, en plus, la page

```
@foreach( var p in Model ) {
    <tr>
      <td>
        @using( Html.BeginForm( "Supprimer", "ExemplePagination" ) ) {
          <input type="submit" value="Supprimer" class="btn btn-danger" />
          <input type="hidden" name="ProductID" value="@p.ProductID" />
          <input type="hidden" name="UnitPrice" value="@p.UnitPrice" />
          <input type="hidden" name="tri" value="@tri" />
          <input type="hidden" name="filtre" value="@filtre"
          <input type="hidden" name="page" value="@page" />
        }
      </td>
      <td>@p.CategoryName</td>
      <td>@p.ProductName</td>
      <td>@($"{p.UnitPrice:0.00}")</td>
    </tr>
  }
</table>
```



Exercice - Début

- Partant d'une copie de l'exercice précédent
  - Ajouter le tri par nom ou par civilité
    - Le filtre doit bien sûr être conservé
    - Remarques :
      - La page doit réaliser le même traitement (filtre + tri) sur POST (filtre) et sur GET (tri). Il sera plus pratique d'utiliser une seule méthode d'action pour gérer les deux cas
      - Les critères étant "nombreux", utiliser un objet personne comme modèle et comme filtre sera plus pratique que 4 champs
  - Ajouter la pagination
    - 5 lignes à la fois pour commencer
    - En cas de crise de courage
      - Ajouter une vue "Préférences" qui permet à l'utilisateur de de choisir la taille des pages.
        - Où stocker cette préférence utilisateur ? (plusieurs possibilités)

Exercice - fin

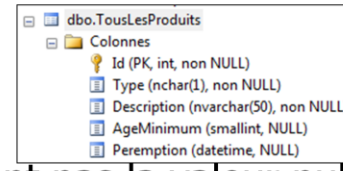
- Édition des personnes
  - Une ligne peut passer en mode édition
  - Il n'est alors plus possible de filtrer, de paginer ou de trier tant que l'édition n'est pas validée ou abandonnée
- Si vous faites une rechute de courage
  - Ajouter une photo à chaque personne
    - La photo, optionnelle, doit être stockée dans une autre table (pourquoi ?)
  - La photo doit être affichée sous forme de petite vignette. Il faut donc utiliser une action spéciale pour retailler l'image.
  - Lors de l'édition, offrir la possibilité de conserver, de remplacer, ou de supprimer cette photo.

Héritage

- TPH
  - Mapping de l'héritage avec une table par hiérarchie
    - Une table définit tous les champs d'une hiérarchie d'héritage
    - Les champs n'appartenant pas à la classe de base doivent accepter la valeur nulle
    - Des valeurs nulles ou non dans certaines colonnes permettrons de déterminer le type de chaque enregistrement
    - Une colonne de discriminant peut également être utilisée. La valeur de la colonne indiquant le type de l'objet à mapper
- ⚠Possibilités d'incohérence (erreur 3012)
  - Un discriminant ne doit pas pouvoir être changé sur une instance de façon à modifier la façon dont la ligne aurait dû être mappée
  - Les colonnes de discriminant peuvent ne pas être mappées
  - Les colonnes dont la valeur nulle sert de discriminant doivent être mappées à un champ non-annulable

TPH

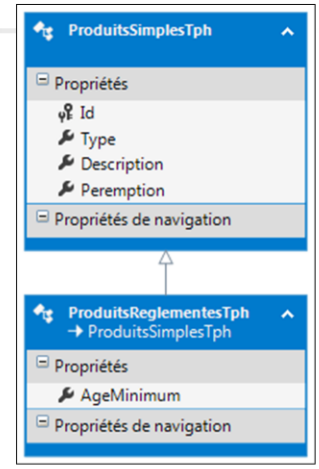
- Exemple
  - Les produits simples : Id, Nom, Description
  - Les produits périssables : Idem, plus une Péremption (date)
  - Les produits réglementés : produit simple plus un AgeMinimum (entier court)
- Création de la table unique



  - La table définit tous les champs
  - Les champs du type de base n'acceptent pas la valeur nulle
  - Les champs des types dérivés acceptent la valeur nulle
  - Une colonne Type (nchar) servira de discriminant
    - S : produit simple ; R : produit réglementé ; P : produit périssable
  - Créer un fichier edmx pour mapper la base

TPH

- Créer les entités
  - Renommer l'entité TousLesProduits en ProduitsSimples
  - Clic droit dans le concepteur, ajouter une entité "ProduitsReglementes"
  - Choisir ProduitsSimples comme type de base
    - Noter qu'il est alors impossible de changer le "nom du jeu d'entités"
  - Couper le champ AgeMinimum de ProduitsSimples et le coller dans ProduitsReglementes
  - Procéder de la même façon pour créer l'entité ProduitsPerissables

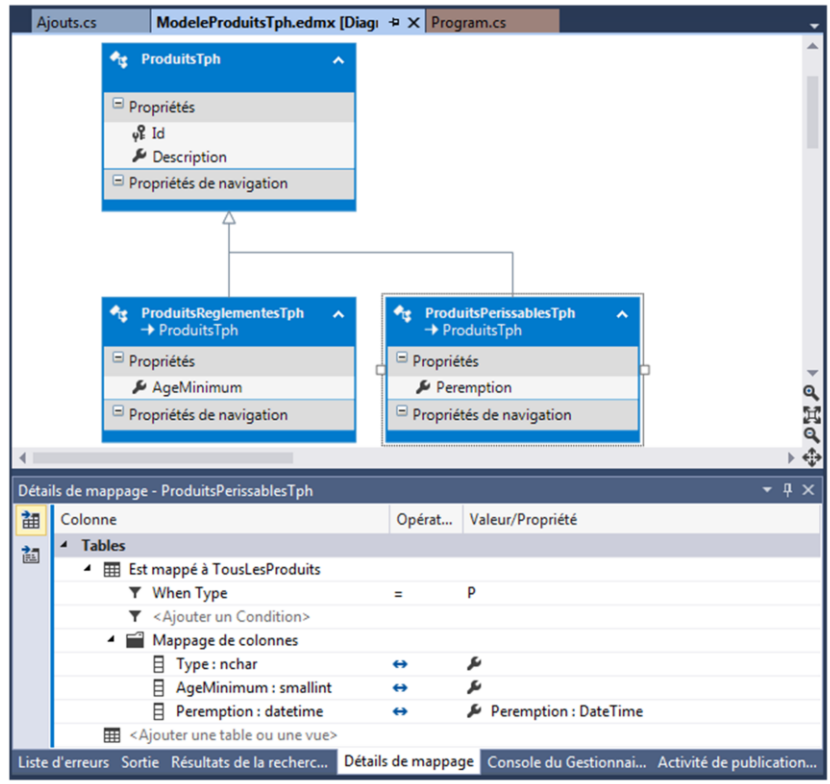


TPH

- Mettre en place les discriminants
  - Dans le détail du mapping de ProduitsSimples, ajouter la condition Type = S
  - Dans le détail du mapping de ProduitsReglementes, ajouter le mapping à la table TousLesProduits
  - Ajouter une condition Type = R
  - Dans le détail du mapping de ProduitsPerissables, ajouter le mapping à la table TousLesProduits
  - Ajouter une condition Type = P
    - ⚠ Attention à l'erreur 3012 !
    - ⚠ Auriez-vous oublié de supprimer le mapping de la colonne "Type" servant par ailleurs de discriminant ?

TPH

- Vue finale



TPH

- Logique métier : étendre la classe partielle

```
partial class ProduitsSimplesTph {
  public ProduitsSimplesTph( string description ) { Description = description; }
  protected void Afficher( string nomType ) {
    Console.WriteLine( "{0} :\n  - {1}", nomType, Description );
  }
  public virtual void Afficher() { Afficher( "Produit simple" ); }
}
partial class ProduitsReglementesTph : ProduitsSimplesTph {
  public ProduitsReglementesTph( string description, short age )
    : base( description ) {
    AgeMinimum = age;
  }
  public override void Afficher() {
    Afficher( "Produit réglementé" );
    Console.WriteLine( "  - Pas avant {0} ans", AgeMinimum );
  }
}
partial class ProduitsPerissablesTph : ProduitsSimplesTph {
  public ProduitsPerissablesTph( string description, DateTime p )
    : base( description ) {
    Peremption = p;
  }
  public override void Afficher() {
    Afficher( "Produit périssable" );
    Console.WriteLine( "  - Consommable jusqu'au {0:d}", Peremption );
  }
}
```

TPH

- Tests

	Id	Type	Description	AgeMinimum	Peremption
	1	P	Yahourt	NULL	10/02/2019 00:0...
	2	R	Derrick, l'intégr...	85	NULL
▶	3	S	Crayon	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

```
ProduitsPerissablesTph cobol = new ProduitsPerissablesTph( "Stage COBOL", DateTime.Parse( "31/12/9999" ) );
dc.ProduitsTphs.Add( cobol );
dc.SaveChanges();
dc.Entry( cobol ).State = EntityState.Detached;    ← Re - requêtera la BD

var rq = from p in dc.ProduitsTphs
         orderby p.Description
         select p;
foreach( ProduitsTph p in rq )
  p.Afficher();
Console.WriteLine( "\nLes dates de péremption :" );
foreach( var d in dc.ProduitsTphs.OfType<ProduitsPerissablesTph>()
                                 .Select( pp => pp.Peremption ) )
  Console.WriteLine( "  - {0:d}", d );
```

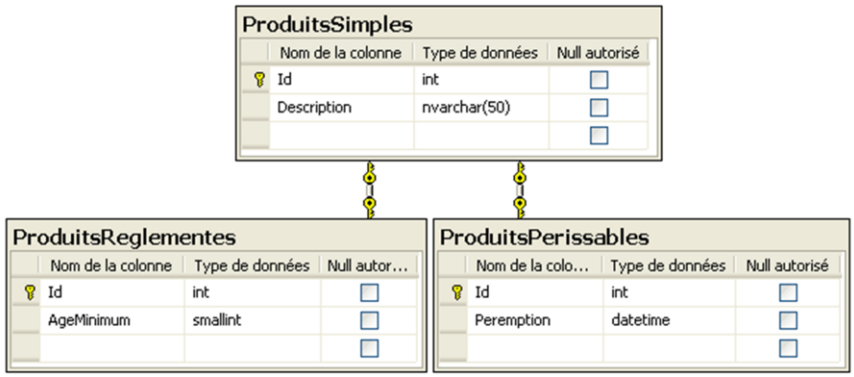
```
Produit simple :
  - Crayon
Produit réglementé :
  - Derrick, l'intégrale en 378 DVDs
  - Pas avant 85 ans
Produit périssable :
  - Stage COBOL
  - Consommable jusqu'au 31/12/9999
Produit périssable :
  - Yahourt
  - Consommable jusqu'au 10/02/2019

Les dates de péremption :
  - 10/02/2019
  - 31/12/9999
```

TPT

- Principe
  - Le type de base est stocké dans une table
  - Des tables supplémentaires sont utilisées pour les types dérivés : les champs propres aux types dérivés ne sont présents que dans les tables qui représentent ces types dérivés
  - Une instance est identifiée par une propriété de clé qui est la même toutes les tables.
  - Les produits simples : Id, Nom, Description
  - Les produits périssables : plus une Peremption
  - Les produits réglementés : plus un AgeMinimum

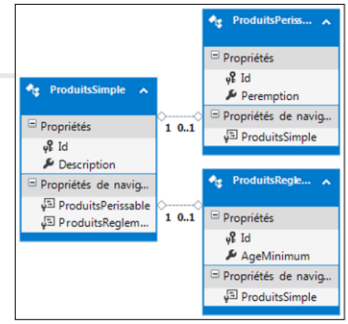
TPT

- Base de données
  - La relation entre ProduitsSimples et ProduitsReglementes et ProduitsPerissable doit être une relation 1:0..1
  - Les trois tables doivent avoir une relation sur leur clef primaire pour permettre l'héritage TPT

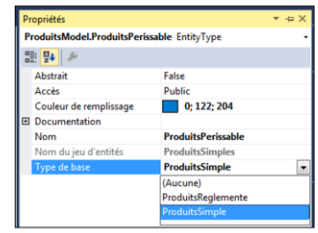


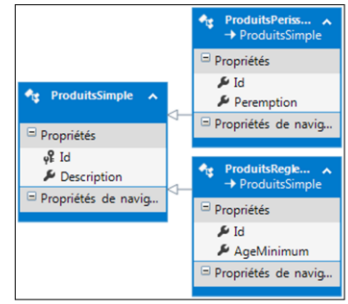
TPT

- Démarche de création
  - Créer Produits.edmx à partir de la base de données Produits



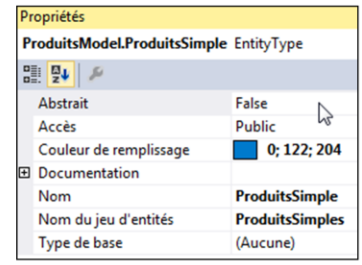
  - Dans les propriétés de ProduitsPerissables et de ProduitsReglementes, affecter "Type de base" à "ProduitsSimples". Des flèches d'héritage apparaissent en plus des flèches d'association
  - Supprimer les associations entre ProduitsPerissables, ProduitsReglementes et ProduitsSimples. Ses propriétés de navigation correspondantes disparaissent également
  - Supprimer les propriétés Id ProduitsReglementes et de ProduitsPerissables pour qu'elle ne soit pas dupliquée avec celle héritée de ProduitsSimples



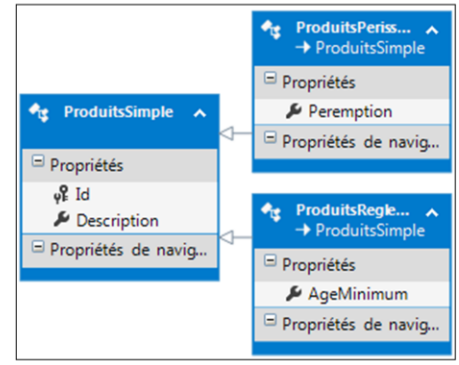


TPT

- Démarche de création, suite
  - Rendre **éventuellement** ProduitsSimples abstrait. Attention : les mappages de fonctions ne sont pas permis sur les types abstraits



  - Dans le mappage de table (fenêtre '*Détails de Mapping*') de ProduitsPerissables et de ProduitsReglementes, la propriété Id (héritée de ProduitsSimples) doit tout de même être mappée à Id (colonne de ProduitsReglementes ou de ProduitsPerissables)



TPT

- Logique métier : étendre la classe partielle

```
partial class ProduitsSimplesTph {
  public ProduitsSimplesTph( string description ) { Description = description; }
  protected void Afficher( string nomType ) {
    Console.WriteLine( "{0} :\n  - {1}", nomType, Description );
  }
  public virtual void Afficher() { Afficher( "Produit simple" ); }
}
partial class ProduitsReglementesTph : ProduitsSimplesTph {
  public ProduitsReglementesTph( string description, short age )
    : base( description ) {
    AgeMinimum = age;
  }
  public override void Afficher() {
    Afficher( "Produit réglementé" );
    Console.WriteLine( "  - Pas avant {0} ans", AgeMinimum );
  }
}
partial class ProduitsPerissablesTph : ProduitsSimplesTph {
  public ProduitsPerissablesTph( string description, DateTime p )
    : base( description ) {
    Peremption = p;
  }
  public override void Afficher() {
    Afficher( "Produit périssable" );
    Console.WriteLine( "  - Consommable jusqu'au {0:d}", Peremption );
  }
}
```

Page de support

TPT

- Tests

```
using( var dc = new ProduitsTptEntities() ) {

  ProduitsPerissable cobol = new ProduitsPerissable(
    "Stage COBOL",
    DateTime.Parse( "31/12/9999" ) );
  dc.ProduitsSimples.Add( cobol );
  dc.SaveChanges();
  dc.Entry( cobol ).State = EntityState.Detached;

  var rq = from p in dc.ProduitsSimples
           orderby p.Description
           select p;
  foreach( ProduitsSimple p in rq )
    p.Afficher();
  Console.WriteLine( "\nLes dates de péremption :" );
  foreach( var d in dc.ProduitsSimples
                      .OfType<ProduitsPerissable>()
                      .Select( pp => pp.Peremption ) ) {
    Console.WriteLine( "  - {0:d}", d );
  }
}
```

```
Produit simple :
  - Crayon
Produit périssable :
  - Yahourt
  - consommable jusqu'au 10/02/2019
Produit réglementé :
  - Derrick, l'intégrale en 378 DVDs
  - pas moins de 85 ans
Produit périssable :
  - Stage Cobol
  - consommable jusqu'au 31/12/9999
```

☹ Zut !

☹ L'idéal aurait été de penser à renommer le type d'entité "ProduitSimple" en "Produit" tout court… J'ai oublié [1]

[1] Et si vous croyez que je vais refaire mes copies d'écran pour ça, vous vous mettez le doigt dans l'œil jusqu'au coude

Code first

- Principe
  - Vous définissez vos classes de modèle
  - Vous définissez vos collections dans votre contexte
  - EF génère la base à partir des métadonnées de vos types
- Mise en œuvre
  - Modèle & attributs
    - Un Id entier sera automatiquement un auto incrément

```
public class Tache {
  //[System.ComponentModel.DataAnnotations.Key]    // OK car nommé "Id"
  public Guid Id { get; set; }
  [MaxLength( 50 ), Required]
  public string Nom { get; set; }
  public Tache() {                    // Ctor sans argument obligatoire
    Id = Guid.NewGuid();
  }
}
```

☺ Contexte et chaine de connexion : base créée en cas de besoin

```
public class AFaireContext : DbContext {
  public static readonly string CnxAFaire;
  static AFaireContext() {
    CnxAFaire = ConfigurationManager.ConnectionStrings["CnxAFaire"].ConnectionString;
  }
  public AFaireContext() : base( CnxAFaire ) {
  }
  public DbSet<Tache> Taches { get; set; }
}
```

Code first

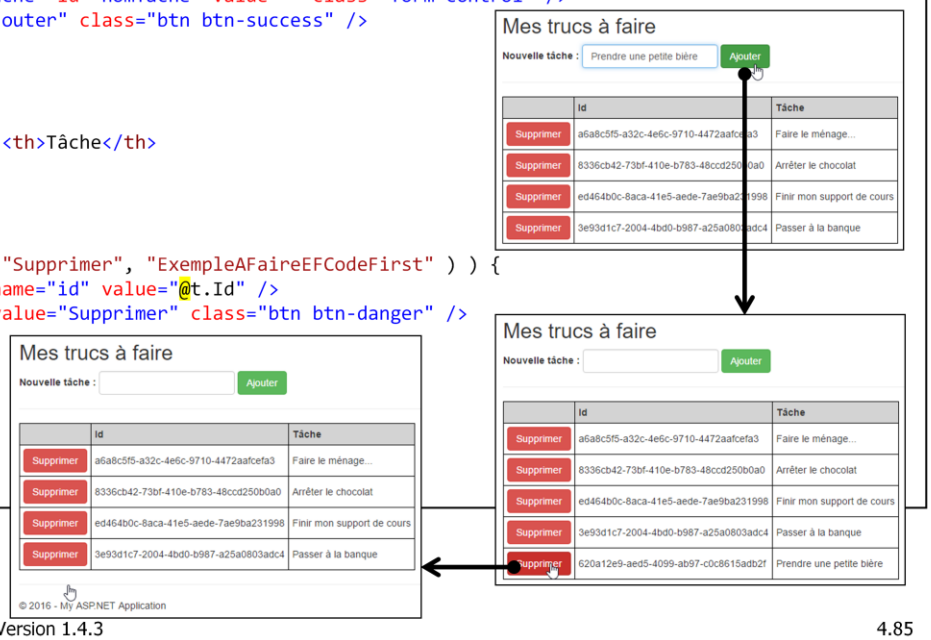
- Utilisation - contrôleur

```
public class ExempleAFaireEFCodeFirstController : Controller {
  AFaireContext DC = new AFaireContext();
  public ActionResult Index() {
    return View( "Index", DC.Taches );
  }
  public ActionResult Ajouter( string nomTache ) {
    if( nomTache == null ) return RedirectToAction( "Index" );
    if( !string.IsNullOrWhiteSpace( nomTache ) ) {
      Tache nvTache = new Tache() { Nom = nomTache };
      DC.Taches.Add( nvTache );
      DC.SaveChanges();
    }
    return Index();
  }
  public ActionResult Supprimer( Guid? id ) {
    if( id == null ) return RedirectToAction( "Index" );
    DC.Entry( new Tache() { Id = id.Value } ).State = System.Data.Entity.EntityState.Deleted;
    DC.SaveChanges();
    return Index();
  }
  protected override void Dispose( bool disposing ) {
    if( disposing )
      DC.Dispose();
    base.Dispose( disposing );
  }
}
```

Code first

- Utilisation - vue

```
@using( Html.BeginForm( "Ajouter", "ExempleAFaireEFCodeFirst", FormMethod.Post, new { @class = "form-inline" })) {
  <label for="nomTache">Nouvelle tâche :</label>
  <input type="text" name="nomTache" id="nomTache" value="" class="form-control" />
  <input type="submit" value="Ajouter" class="btn btn-success" />
}
<hr />
<table class="std">
  <tr>
    <th> </th> <th>Id</th> <th>Tâche</th>
  </tr>
  @foreach( var t in Model ) {
    <tr>
      <td>
        @using( Html.BeginForm( "Supprimer", "ExempleAFaireEFCodeFirst" ) ) {
          <input type="hidden" name="id" value="@t.Id" />
          <input type="submit" value="Supprimer" class="btn btn-danger" />
        }
      </td>
      <td>@t.Id</td>
      <td>@t.Nom</td>
    </tr>
  }
</table>
```



Migrations

- Principe
  - L'édition d'un fichier edmx en équipe est "problématique"
  - Les changements dans le modèle peuvent plus facilement être suivis : Migrations/Configuration.cs
    - Archivage du code de migration
    - Archivage du code de régression
- Mise œuvre

```
PM> Enable-Migrations –EnableAutomaticMigrations -ContextTypeName AFaireContext
```

    - Avec un seul contexte `–contextTypeName` est optionnel
  - En suite
    - Add-Migration : enregistre les changements du modèle dans la configuration
    - Update-Database : applique les changements en base

Migrations

- Exemple
  - Ajouter une propriété à la classe Tache

```
public DateTime Echeance { get; set; }
```

  - Compiler
  - Créer les informations de migration

```
PM> Enable-Migrations –EnableAutomaticMigrations
```

- PM> Enable-Migrations –EnableAutomaticMigrations
  - La classe Migrations/Configuration.cs peut permettre de réalimenter la base après une migration

```
protected override void Seed( Donnees.AFaireContext context ) {
  context.Taches.AddOrUpdate(new Tache(){Nom="Finir mon support de cours", Echeance=DateTime.Today.AddDays(-1)});
  context.Taches.AddOrUpdate( new Tache() {Nom="Arrêter le chocolat", Echeance = DateTime.Now.AddYears(100)});
  context.Taches.AddOrUpdate( new Tache() {Nom = "Faire le ménage", Echeance = DateTime.Today.AddYears( 1 )  } );
  context.Taches.AddOrUpdate( new Tache() { Nom = "Passer à la banque", Echeance = DateTime.Today.AddDays(1) } );
}
```

Migrations

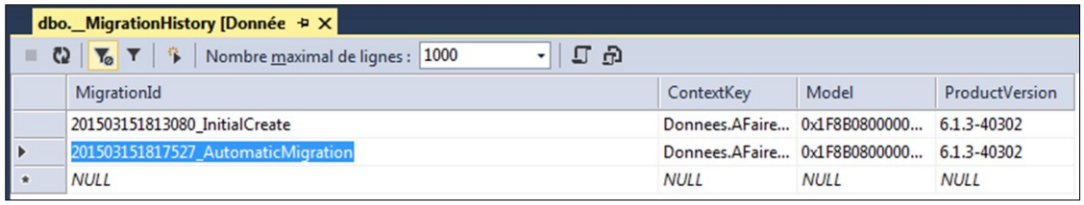
- Exemple, suite
  - Le schéma de la base n'a pas encore évolué :

```
PM> Update-Database
```

    - Le code d'alimentation est exécuté par la commande Update Database
    - En fonction des modifications du modèle, les anciennes lignes peuvent être perdues
  - Revenir à une ancienne version :

```
PM> Update-Database -Target:201503151813080_InitialCreate
```

    - Les migrations peuvent avoir des noms plus personnels…



Migrations

- Ajouts, encore

```
public class Tache { // [ . . . ]
  [Required] public int NbParticipants { get; set; }
  [ForeignKey( "Outil" )] public virtual ICollection<Outil> Outils { get; set; }
}
[Table( "Outils" )]
public class Outil {
  public Guid Id { get; set; }
  public string Nom { get; set; }
  public virtual ICollection<Tache> Taches { get; set; }
  public Outil() { Id = Guid.NewGuid(); }
}

public class AFaireContext : DbContext { // [ . . . ]
  public DbSet<Outil> Outils { get; set; }
  protected override void OnModelCreating( DbModelBuilder mb ) {
    base.OnModelCreating( mb );
    mb.Entity<Outil>()
      .HasMany<Tache>( outil => outil.Taches )          ← Fluent API
      .WithMany( tache => tache.Outils )                ← Fluent API
      .Map( config => {                                 ← Fluent API
        config.MapLeftKey( "OutilId" );                 ← Fluent API
        config.MapRightKey( "TacheId" );                ← Fluent API
        config.ToTable( "OutilsTaches" );               ← Fluent API
      } );
  }
}
```

- **NbParticipants est non-nul ➔ initialiser à 1 pour les lignes existantes (pas à 0)**

Migrations - seed

```
internal sealed class Configuration : DbMigrationsConfiguration<AFaireContext> {
  public Configuration() {
    AutomaticMigrationsEnabled = true;
    ContextKey = "WebApplication1.ModelsAFaireContext";
  }
  protected override void Seed( AFaireContext context ) {
    var m = new Outil() { Nom = "Marteau" }; var b = new Outil() { Nom = "Balai" };
    var c = new Outil() { Nom = "Courage" }; var p = new Outil() { Nom = "Perceuse" };
    // Pas nécessaire car référencés par des Taches suivies par le DC
    //context.Outils.AddOrUpdate( outil => outil.Nom, m, b, c, p );
    var t1 = new Tache() {
      Nom = "Finir mon support de cours", NbParticipants = 1, Echeance = DateTime.Today.AddDays( -1 ),
      Outils = new HashSet<Outil>() { c }
    };
    var t2 = new Tache() {
      Nom = "Arrêter le chocolat", NbParticipants = 0, Echeance = DateTime.Now.AddYears( 100 ),
      Outils = new HashSet<Outil>() { c }
    };
    var t3 = new Tache() {
      Nom = "Faire le ménage...", NbParticipants = 2, Echeance = DateTime.Today.AddYears( 1 ),
      Outils = new HashSet<Outil>() { b }
    };
    var t4 = new Tache() {
      Nom = "Passer à la banque", NbParticipants = 3, Echeance = DateTime.Today.AddDays( 1 ),
      Outils = new HashSet<Outil>() { c, m, p }
    };
    context.Taches.AddOrUpdate( t => t.Nom, t1, t2, t3, t4 );
    // Relations non restaurées si elles ont été supprimées mais les entités supprimées seront restaurées
  }
}
```

Migrations

- Ajouter le code de migration

```
PM> Add-Migration AddOutilsEtNbParticipants
```

  - AddNbParticipants apparaît dans le dossier Migrations, préfixé par un horodatage : 201503221357573_AddOutilsEtNbParticipants.cs

```
public partial class AddOutilsEtNbParticipants : DbMigration {
  public override void Up() {
    CreateTable("dbo.Outils", c => new {Id = c.Guid( nullable: false ), Nom = c.String(), } ).PrimaryKey(t=>t.Id);
    CreateTable("dbo.OutilsTaches",
      c => new {OutilId = c.Guid( nullable: false ), TacheId = c.Guid( nullable: false ), } )
      .PrimaryKey( t => new { t.OutilId, t.TacheId } )
      .ForeignKey( "dbo.Outils", t => t.OutilId, cascadeDelete: true )
      .ForeignKey( "dbo.Taches", t => t.TacheId, cascadeDelete: true )
      .Index( t => t.OutilId ).Index( t => t.TacheId );
    AddColumn( "dbo.Taches", "NbParticipants",
               c => c.Int( nullable: false, defaultValue: 1 ) );
  }
  public override void Down() {
    DropForeignKey( "dbo.OutilsTaches", "TacheId", "dbo.Taches" );
    DropForeignKey( "dbo.OutilsTaches", "OutilId", "dbo.Outils" );
    DropIndex( "dbo.OutilsTaches", new[] { "TacheId" } );
    DropIndex( "dbo.OutilsTaches", new[] { "OutilId" } );
    DropColumn( "dbo.Taches", "NbParticipants" );
    DropTable( "dbo.OutilsTaches" );
    DropTable( "dbo.Outils" );
  }
}
```

Seul ce `defaultValue: 1` justifie la migration non-automatique

Migrations

- Mettre à jour la base à l'aide du code de migration personnalisé
  - Pour faire évoluer la base

```
PM> Update-Database
```

  - Pour revenir

```
PM> Update-Database -Target 201503151817527_AutomaticMigration –Force
```

    - L'option -Force permet d'autoriser la perte de données

- La colonne NbParticipants des lignes existantes est bien passée à 1

Id	Nom	Eche...	NbParticipants
00000000-0000-0000-0000-000000000001	TEST	23/03...	1
a6a8c5f5-a32c-4e6c-9710-4472aafcefa3	Faire le ménage...	22/03...	2
8336cb42-73bf-410e-b783-48ccd250b0a0	Arrêter le chocolat	22/03...	0
ed464b0c-8aca-41e5-aede-7ae9ba231998	Finir mon support de co...	21/03...	1
3e93d1c7-2004-4bd0-b987-a25a0803adc4	Passer à la banque	23/03...	3
NULL	*NULL*	*NULL*	*NULL*

Code first pendant le développement

- Principe
  - Création de la base à partir du modèle
  - Évolution de la base à partir du modèle
  - Réalimentation de la base à chaque démarrage / à chaque changement du modèle

```
[Table( "Annonces" )]                    ← Automatique de toutes façons
public class Annonce {
  public Guid Id { get; set; }
  [Required, MaxLength( 200 )]
  public string Titre { get; set; }
  [Required]
  public string Texte { get; set; }
  public Guid RedacteurId { get; set; } ← Clé étrangère automatique
  public virtual Redacteur Redacteur { get; set; }
}
[Table( "Redacteurs" )]
public class Redacteur {
  public Guid Id { get; set; }
  [Required, MaxLength( 50 )]
  public string Nom { get; set; }
  public virtual ICollection<Annonce> Annonces { get; set; }
}
```

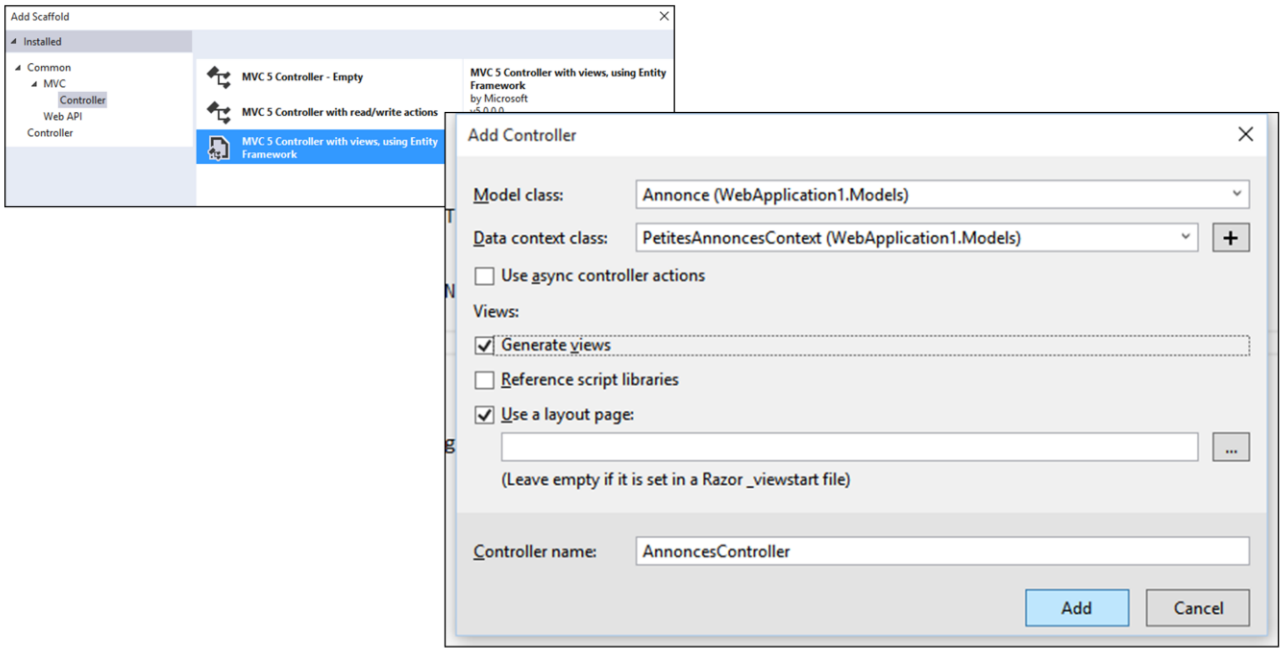
Code first pendant le développement

- Contexte + seed, apprécié des développeurs

```
public class PetitesAnnoncesContext : DbContext {
  static PetitesAnnoncesContext() {
    Database.SetInitializer<PetitesAnnoncesContext>( new PetitesAnnoncesContextInitialiseur() );
  }
  public PetitesAnnoncesContext() : base( "name=CnxPetitesAnnonces" ) { }
  public virtual DbSet<Annonce> Annonces { get; set; }
  public virtual DbSet<Redacteur> Redacteurs { get; set; }
  protected override void OnModelCreating( DbModelBuilder mb ) {
    mb.Entity<Annonce>().HasRequired( a => a.Redacteur ).WithMany( r => r.Annonces ).WillCascadeOnDelete( true );
  }
}
public class PetitesAnnoncesContextInitialiseur : DropCreateDatabaseAlways<PetitesAnnoncesContext> {
  protected override void Seed( PetitesAnnoncesContext context ) {          ← ∃ DropCreateDatabaseIfModelChanges

    var r1 = new Redacteur() { Id = Guid.NewGuid(), Nom = "R1",
      Annonces = new HashSet<Annonce>() {
        new Annonce() { Id=Guid.NewGuid(), Texte="Texte 1", Titre="Titre 1" },
        new Annonce() { Id=Guid.NewGuid(), Texte="Texte 2", Titre="Titre 2" } }
    };
    var r2 = new Redacteur() { Id = Guid.NewGuid(), Nom = "R2",
      Annonces = new HashSet<Annonce>() {new Annonce() { Id=Guid.NewGuid(), Texte="Texte 3", Titre="Titre 3" } }
    };
    context.Redacteurs.Add( r1 );
    context.Redacteurs.Add( r2 );
    base.Seed( context );
  }
}
```

Code first pendant le développement

- Exemple d'utilisation (scaffolfing mais allégé !)



Code first pendant le développement

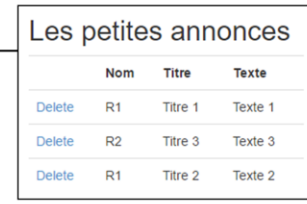
- Contrôleur allégé

```
public class AnnoncesController : Controller {
  private PetitesAnnoncesContext db = new PetitesAnnoncesContext();
  public ActionResult Index() {
    var annonces = db.Annonces.Include( a => a.Redacteur );
    return View( annonces.ToList() );
  }
  public ActionResult Delete( Guid? id ) {
    if( id == null )
      return new HttpStatusCodeResult( HttpStatusCode.BadRequest );
    Annonce annonce = db.Annonces.Find( id );
    if( annonce == null )
      return HttpNotFound();
    return View( annonce );
  }
  [HttpPost, ActionName( "Delete" ), ValidateAntiForgeryToken]
  public ActionResult DeleteConfirmed( Guid id ) {
    Annonce annonce = db.Annonces.Find( id );
    db.Annonces.Remove( annonce );
    db.SaveChanges();
    return RedirectToAction( "Index" );
  }
  protected override void Dispose( bool disposing ) {
    if( disposing )
      db.Dispose();
    base.Dispose( disposing );
  }
}
```

Code first pendant le développement

- Vue Index allégée

```
@model IEnumerable<WebApplication1.Models.Annonce>
@{ ViewBag.Title = "Les petites annonces"; }
<h2>@ViewBag.Title</h2>
<table class="table">
  <tr>
    <th></th>
    <th> @Html.DisplayNameFor( model => model.Redacteur.Nom ) </th>
    <th> @Html.DisplayNameFor( model => model.Titre ) </th>
    <th> @Html.DisplayNameFor( model => model.Texte ) </th>
  </tr>
  @foreach( var item in Model ) {
    <tr>
      <td>
        @Html.ActionLink( "Delete", "Delete", new { id = item.Id } )
      </td>
      <td> @Html.DisplayFor( modelItem => item.Redacteur.Nom ) </td>
      <td> @Html.DisplayFor( modelItem => item.Titre ) </td>
      <td> @Html.DisplayFor( modelItem => item.Texte ) </td>
    </tr>
  }
</table>
```



Code first pendant le développement

- Vue Delete allégée

```
@model WebApplication1.Models.Annonce
@{ ViewBag.Title = "Suppression"; }
<h2>@ViewBag.Title</h2>
<h3>Confirmez-vous la suppression ?</h3>
<div>
  <h4>Annonce</h4>
  <hr />
  <dl class="dl-horizontal">
    <dt> @Html.DisplayNameFor( model => model.Redacteur.Nom ) </dt>
    <dd> @Html.DisplayFor( model => model.Redacteur.Nom ) </dd>
    <dt> @Html.DisplayNameFor( model => model.Titre ) </dt>
    <dd> @Html.DisplayFor( model => model.Titre ) </dd>
    <dt> @Html.DisplayNameFor( model => model.Texte ) </dt>
    <dd> @Html.DisplayFor( model => model.Texte ) </dd>
  </dl>
  @using( Html.BeginForm() ) {
    @Html.AntiForgeryToken()
    <div class="form-actions no-color">
      <input type="submit" value="Confirmer la suppression" class="btn btn-default" /> |
      @Html.ActionLink( "Retour aux petites annonces", "Index" )
    </div>
  }
</div>
```