Cyril Seguenot

2018

Résumé

Support d’accompagnement d’une formation à ADO.Net faite en présentiel

Formation à ADO.Net

Support d'exercices

Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l’auteur ou de ses ayants droit ou ayants cause est illicite selon le Code de la propriété intellectuelle (Art L 122-4) et constitue une contrefaçon réprimée par le Code pénal.

Seules sont autorisées (Art L122-5) les copies ou reproductions strictement réservées à l’usage privé du copiste et non destinées à une utilisation collective, ainsi que les analyses et courtes citations justifiées par le caractère critique, pédagogique ou d’information de l’œuvre à laquelle elles sont incorporées, sous réserve, toutefois, du respect des dispositions des articles L 122-10 à L 122-12 du même Code, relatives à la reproduction par reprographie.

©Cyril Seguenot 2018

Table des matières

[1 Northwind2 3](#_Toc502612480)

[1.1 Mise en place du projet 3](#_Toc502612481)

[1.2 Affichage de données 4](#_Toc502612482)

[1.3 Insertion, mise à jour et suppression de données 6](#_Toc502612483)

[1.4 Insertion et suppression en masse 8](#_Toc502612484)

[2 Northwind2 avec DataSet 10](#_Toc502612485)

[2.1 Affichage de données 10](#_Toc502612486)

[2.2 Insertion, mise à jour et suppression de données 11](#_Toc502612487)

[3 Northwind2 avec Entity Framework 13](#_Toc502612488)

[3.1 Affichage de données 13](#_Toc502612489)

[3.2 Insertion, mise à jour et suppression de données 14](#_Toc502612490)

[4 Tests unitaires 15](#_Toc502612491)

# Northwind2

Cet exercice utilise la base de données Northwind2. Les données seront stockées en mémoire sous forme de collections d’objets (pas d’utilisation de DataSet).

## Mise en place du projet

**Etape 1 : Création du projet et connexion à la base**

* Dans la solution Exercices, si ce n’est pas déjà fait, ajouter le projet TConsole dont les sources sont fournies ci-dessous :



* Ajouter un projet de type application console, nommé Northwind2.
* Y ajouter des références vers :
  + Le projet TConsole
  + L’assembly System.Configuration (pour pouvoir accéder à la chaîne de connexion)
* Dans le fichier App.config, ajouter une connexion à la base Northwind2

**Etape 2 : Création des classes**

Ajuter les éléments suivants au projet :

* Un dossier Pages qui contiendra les classes décrivant les pages de l’application
* Un fichier Entites.cs, qui contiendra les entités POCO  
   /!\ Pour l’exercice 2, il est important que les noms des propriétés de chaque entité correspondent à ceux des colonnes de la table associée dans la base de données
* Une classe Contexte qui contiendra les méthodes de requêtes ADO.Net.
* Une classe Northwind2App, dérivée de ConsoleApplication et contenant le code ci-dessous

public class Northwind2App : ConsoleApplication

{

   private static Northwind2App \_instance;

   /// <summary>

   /// Obtient l'instance unique de l'application

   /// </summary>

   public static Northwind2App Instance

   {

      get

      {

         if (\_instance == null)

            \_instance = new Northwind2App();

         return \_instance;

      }

   }

   // Constructeur

   public Northwind2App()

   {

      // Définition des options de menu à ajouter dans tous les menus de pages

      MenuPage.DefaultOptions.Add(

         new Option("a", "Accueil", () => \_instance.NavigateHome()));

   }

}

La propriété Instance est un singleton. Elle garantit qu’on ne peut créer qu’une instance unique de l’appli. Cette instance est créée au premier appel de la propriété.

Le constructeur crée une entrée de menu qui sera par défaut ajoutée dans tous les menus de l’appli.

**Etape 3 : Page d’accueil et fonction Main**

Dans le dossier Pages, ajouter une classe PageAccueil dérivée de MenuPage, avec le titre « Accueil »

Lui ajouter une entrée de menu « 0. Quitter l’application » qui exécute le code suivant : « Environnement.Exit(0) »

Dans la fonction Main de la classe Program, utiliser le singleton Northwind2app.Instance pour

* Définir le titre de l’appli
* Ajouter la page d’accueil, puis naviguer vers elle
* Lancer l’appli

## Affichage de données

**Etape 1 : Page Fournisseurs**

Créer la classe PageFournisseur dérivée de MenuPage, avec le titre « Fournisseurs »

Ajouter une instance de cette classe à la liste des pages de l’appli

Dans la page d’accueil, créer une entrée de menu pour naviguer vers la page des fournisseurs

**Etape 2 : liste des pays des fournisseurs**

Dans la classe Contexte, créer une méthode GetPaysFournisseurs qui renvoie la liste des pays des fournisseurs (sans doublons)

Dans la page des fournisseurs, ajouter une entrée de menu « 1. Liste des pays » permettant d’afficher cette liste de pays sous forme de tableau, à l’aide de la classe ConsoleTable.

**Etape 3 : liste des fournisseurs d’un pays donné**

Dans la classe Contexte, créer une méthode GetFournisseurs qui renvoie la liste des fournisseurs (Id et nom d’entreprise) d’un pays passé en paramètre.

Dans la page des fournisseurs, ajouter une entrée de menu « 2. Fournisseurs d’un pays » pour faire saisir un pays à l’utilisateur et afficher la liste des fournisseurs correspondante.

**Etape 4 : Nombre de produits proposé dans un pays**

Dans le contexte, créer une méthode GetNbProduits qui renvoie le nombre de produits fournis par l’ensemble des fournisseurs du pays choisi.

Dans la page des fournisseurs, ajouter une entrée de menu « 3. Nombre de produits d’un pays » pour faire saisir un pays à l’utilisateur et afficher le nombre de produits correspondant.

**Etape 5 : utilisation d’une fonction sql**

Dans la base de données, créer une fonction sql ufn\_GetNbProduits qui renvoie le nombre de produits fournis par l’ensemble des fournisseurs du pays choisi.

Dans le contexte, modifier GetNbProduits pour qu’elle appelle la fonction ufn\_GetNbProduits.

**Etape 6 : Récupération des clients et commandes en une seule requête**

Dans le contexte, créer une méthode GetClientsCommandes permettant de récupérer en une seule requête la liste des clients (Id et nom de société) et de leurs commandes associées (Id, date de commande, date d’envoi, frais, nombre d’articles différents et montant total).

Créer une nouvelle page PageClientsCommandes dérivant de Page, et ouverte par une entrée de menu « 2. Clients et commandes » dans la page d’accueil.

Redéfinir sa méthode Display en affichant la liste des clients obtenue par la méthode de contexte GetClientsCommandes

Demander la saisie d’un id de client

Au moyen d’une requête Linq, extraire la liste de commandes de ce client, puis l’afficher en tableau.

Tester avec le client « CACTU ». Comment s’affiche la date de livraison de la commande lorsqu’elle est nulle dans la base ? Comment mieux gérer ce cas ?

## Insertion, mise à jour et suppression de données

Cet exercice est la suite du précédent. Continuer à travailler dans le même projet.

L’objectif est de créer une saisie complète de produits, avec

* Une page affichant la liste des produits, et permettant de supprimer des produits
* Une page de création d’un nouveau produit ou de mise à jour d’un produit existant

**Etape 1 : Affichage des produits par catégorie**

Créer une nouvelle page PageProduits dérivant de MenuPage, et ouverte par une entrée de menu « 3. Produits » dans la page d’accueil.

Dans la page des produits, ajouter une entrée de menu « 1. Liste des produits », appelant une méthode AfficherProduits. Celle-ci doit :

* Afficher la liste des catégories (id, nom, description)
* Demander à l’utilisateur de saisir un id de catégorie
* Afficher les produits de cette catégorie (Id, nom, prix unitaire, unités en stock), triée par Id

Créer les entités et méthodes de contexte nécessaires à la méthode AfficherProduits.

Remarque : dans la console, pour saisir facilement l’id de catégorie, activer le mode d’édition rapide dans les propriétés de la fenêtre. L’id peut être copié en le sélectionnant avec la souris et en appuyant sur Entrée, puis collé par clic droit.

**Etape 2 : Création d’un nouveau produit**

Dans la page des produits, ajouter une entrée de menu « 2. Créer un nouveau produit », appelant une méthode CréerProduit. Celle-ci doit :

* Afficher la liste des catégories (id, nom, description)
* Demander à l’utilisateur de saisir les informations du produit, ligne à ligne et dans l’ordre suivant : id de la catégorie, nom, id du fournisseur, prix unitaire, unités en stock
* Appeler la méthode d’ajout du produit en base
* Afficher un message en vert : « Produit créé avec succès »

Créer une méthode de contexte AjouterProduit qui sera appelée par la méthode CréerProduits afin d’enregistrer le nouveau produit dans la base.

**Etape 3 : Modification d’un produit**

Dans la page des produits, ajouter une entrée de menu « 3. Modifier un produit », appelant une méthode ModifierProduit, qui doit :

* Appeler la méthode d’affichage des produits, et récupérer l’id de la catégorie choisie
* Demander la saisie de l’id du produit à modifier
* Demander la saisie les informations du produit, ligne à ligne et dans l’ordre suivant : nom, id de la catégorie, id du fournisseur, prix unitaire, unités en stock
* Appeler la méthode de mise à jour du produit en base
* Afficher un message en vert « Produit modifié avec succès »

Pour la mise à jour dans la base :

* Renommer la méthode de contexte AjouterProduit en AjouterModifierProduit
* Lui ajouter un paramètre énuméré pour le type d’opération (Ajout ou Modification)
* Exécuter la commande adéquate en fonction de l’opération choisie

**Etape 4 : Suppression d’un produit**

Dans la page des produits, ajouter une entrée de menu « 4. Supprimer un produit », appelant une méthode SupprimerProduit, qui doit :

* Appeler la méthode d’affichage des produits, et récupérer l’id de la catégorie choisie
* Demander la saisie de l’id du produit à supprimer
* Appeler la méthode de suppression du produit en base
* Afficher un message en vert « Produit supprimer avec succès »

**Etape 5 : Gestion des erreurs**

Lorsqu’on tente de supprimer un produit qui est référencé par une ligne de commande, l’application plante. Pour y remédier :

* Retrouver le numéro de l’erreur SQL correspondante en exécutant la requête dans SSMS
* Dans la méthode SupprimerProduit de PageProduit, entourer l’appel à la méthode du contexte d’un bloc try – catch, et à l’intérieur du bloc catch, appeler une méthode interne GérerErreurSql
* Dans GérerErreurSql, intercepter les erreurs de type System.Data.SqlClient.SqlException, et afficher en rouge un message spécifique pour l’erreur précédente. Les autres erreurs doivent être renvoyées telles quelles, pour faire planter l’application

**Etape 6 : Transaction**

Créer une méthode de contexte AjouterProduitCategorie qui exécute deux commandes au sein d’une transaction :

* La première commande crée une catégorie « Others » si elle n’existe pas déjà, et si le produit à ajouter a un Id de catégorie = Guid.Empty). L’Id de la catégorie Others doit être défini en dur dans le code en tant que champ statique de la classe Contexte.
* La seconde ajoute le nouveau produit, en lui affectant la catégorie « Others » si sa catégorie vaut Guid.Empty.

Dans PageProduits, créer une entrée de menu associée à une méthode CréerProduitSansCatégorie, qui demande la saisie d’un produit sans catégorie, et appelle Contexte.AjouterProduitCategorie.  
Tester les scénarios suivants :

* Tenter d’ajouter un produit sans catégorie et avec un id de fournisseur qui n’existe pas. Vérifier que la transaction est bien annulée et que la catégorie « Others » n’est pas créée.
* Tenter d’ajouter un produit sans catégorie, et un id de fournisseur correct. Vérifier que la transaction est bien validée et que la catégorie « Others » est créée.

## Insertion et suppression en masse

Cet exercice est la suite du précédent. Continuer à travailler dans le même projet.

**Etape 1 : Mise en place de l’enregistrement global**

Pour optimiser les traitements, on ne souhaite plus enregistrer les ajouts et suppressions de produits à chaque clic sur les boutons + ou -, mais plutôt de façon globale quand on clique sur un bouton Enregistrer. Pour cela, dans la page PageProduits :

* Ajouter un bouton « Enregistrer » à côté des boutons + et –
* Créer 2 listes privées nommées \_produitsAjoutés et \_produitsSupprimés, destinées à stocker les produits à ajouter et à supprimer avant de faire l’enregistrement en base
* Dans les gestionnaires d’évènements clic des boutons + et -, remplacer les appels aux méthodes de contexte par l’ajout des produits dans les listes crées précédemment.   
  NB/ La DataGridView doit continuer à refléter l’ajout et la suppression des produits, même s’ils ne sont pas encore enregistrés en base.

**Etape 2 : Création du type table pour l’insertion de produits en masse**

Dans la base, créer un type table TypeTableProduit qui vous permettra de stocker en mémoire les produits à ajouter.

**Etape 3 : Création de la méthode d’insertion en masse**

Créer une méthode de contexte prenant en paramètre une liste de produits, et réalisant l’insertion de ces produits en masse dans la base. Cette méthode utilisera le type table créé précédemment et exécutera la requête au sein d’une transaction.

**Etape 4 : Appel de la méthode d’insertion**

Sur le clic du bouton Enregistrer, appeler la méthode de contexte que vous venez de créer, en lui passant en paramètre la liste des produits ajoutés.

Tester en créant plusieurs produits d’affilée, puis en cliquant sur Enregistrer. Vérifier dans la base que les produits ont bien été ajoutés.

**Etape 5 : Rafraîchissement de la liste**

Pour rafraîchir les Id des produits dans la DataGridView, après appel de la méthode d’insertion de masse, faire en sorte de recharger la liste complète des produits.

**Etape 6 : Création du type table pour la suppression de produits en masse**

Pour supprimer des produits, on n’a besoin que de leurs identifiants. Créer un type table TypeTableId avec une seule colonne de type entier.

**Etape 7 : Création de la méthode de suppression en masse**

Au lieu de supprimer les produits, nous allons simplement les rendre obsolètes, c’est-à-dire passer leur champ Discontinued à 1.

Créer une méthode de contexte prenant en paramètre une liste d’identifiants de produits, et réalisant leur mise à jour en masse dans la base. Cette méthode utilisera le type table créé précédemment, et exécutera la requête au sein d’une transaction.

**Etape 8 : Modification de la requête de liste des produits**

Dans la méthode de contexte qui récupère la liste des produits, ajouter une clause where à la requête, de façon à ne récupérer que les produits qui ne sont pas obsolètes

**Etape 9 : Appel de la méthode de suppression**

Sur le clic du bouton Enregistrer, appeler la méthode de contexte de suppression des produits, en lui passant en paramètre la liste des produits supprimés.

Tester en supprimant plusieurs produits d’affilée, puis en cliquant sur Enregistrer. La liste des produits étant rechargée complètement à chaque clic sur ce bouton, si la suppression s’est bien passée, les produits supprimés ne doivent pas réapparaître.

# Northwind2 avec DataSet

Cet exercice utilise la base de données Northwind2 et stocke les données en mémoire dans des DataSet.

## Affichage de données

**Etape 1 : refactoring de code**

Nous allons faire en sorte de pouvoir utiliser plusieurs types de contextes différents de façon générique dans l’application. Pour cela :

* Créer une interface IDataContext qui reprend tous les membres publics de la classe Contexte. Les méthodes renvoyant des collections d’objets doivent utiliser des types IList<T> comme types de retour.
* Renommer la classe Contexte en Contexte1 dans tout le projet, et faire les modifications nécessaires pour qu’elle implémente IDataContext
* Dans la classe Northwind2App, ajouter une propriété statique DataContext de type IDataContext, sur le même principe que la propriété Instance. Cette propriété doit renvoyer une instance de Contexte1.
* Dans les pages de l’application, modifier les appels de méthodes du contexte, en utilisant la propriété précédente, et changer les variables de type List<T> en IList<T>.

Vérifier que l’application fonctionne toujours normalement.

Désormais, nous pouvons changer très facilement le type de contexte à instancier dans la propriété DataContext, sans avoir à modifier le reste du code.

**Etape 2 : Création d’un second type de contexte**

Créer une classe Contexte2 qui implémente IDataContext de façon implicite. Cette classe utilisera des DataSet pour dialoguer avec la base de données.

Ajouter les mêmes champs privés que Contexte1 pour la chaîne de connexion et l’id de la catégorie Others.

**Etape 3 : Récupération des pays et fournisseurs**

Implémenter les méthodes GetPaysFournisseurs et GetFournisseurs en récupérant les données dans des DataSet et en transformant leurs DataTable en List<T> grâce aux méthodes d’extension ToList<T> et ToSimpleList<T> fournies par la classe DataSetHelper dont voici le code :



**Etape 4 : Récupération des clients et commandes**

Implémenter la méthode GetClientsCommandes de la façon suivante :

* Créer une commande SQL regroupant deux requêtes, la première pour récupérer les clients (Id et nom de société), la seconde pour récupérer les commandes (Id, date de commande, date d’envoi, frais, nombre d’articles différents et montant total).
* Avant de récupérer le résultat dans un DataSet avec la méthode Fill :
  + Appeler la méthode FillSchema du DataAdapter, ce qui aura pour effet de créer deux DataTable, correspondant aux deux requêtes
  + Créer une relation entre ces deux tables de façon à retrouver facilement les commandes de chaque client. Pour cela, on utiliser un code du type :

dsClients.Relations.Add("OrdersOfCustomer",

               dsClients.Tables[0].Columns["Customerid"],

               dsClients.Tables[1].Columns["Customerid"]);

La méthode Add prend en paramètre le nom de la relation et les colonnes qui participent à la relation entre les deux tables.

* Remplir le DataSet en appelant sa méthode Fill
* Transférer le contenu du DataSet dans une liste d’entités Customer

Illustre l’utilisation d’un DataSet avec 2 DataTables en relation, et le parcours des lignes des DataTable

## Insertion, mise à jour et suppression de données

Cet exercice est la suite du précédent. Continuer à travailler dans le même projet.

L’objectif est ici d’implémenter la saisie de produits au moyen d’un DataSet. Les ajouts, modifications et suppressions seront mémorisés dans un DataSet, et enregistrés en base lorsque l’utilisateur le choisira.

**Etape 1 : Affichage des catégories**

Implémenter la méthode GetCatégories en utilisant un DataSet, et en transformant sa DataTable en List<T> grâce à la méthode d’extension ToList<T>.

**Etape 2 : Initialisation du DataSet des produits**

Dans le contexte, ajouter une méthode InitDatSetProduits qui permet de :

* Créer une commande pour la sélection des produits d’une catégorie passée en paramètre à la requête
* Instancier un DataSet et un SqlDataAdapter pour les produits, et les mémoriser dans des champs privés de la classe Contexte2
* Créer le schéma du DataSet (qui ne contiendra qu’une seule table)
* Générer les requêtes d’insertion, mise à jour et suppression de produits à partir de la requête de sélection, grâce à un SqlCommandBuilder

**Etape 3 : Enregistrement des modifications en base**

Dans l’interface IDataContext, et dans le contexte, ajouter également une méthode EnregistrerModifsProduits permettant d’enregistrer les modifications dans la base de données.

Dans cette méthode, créer une connexion, l’affecter aux commandes insert, update et delete du SqlDataAdapter créé précédemment, et appeler la méthode update du SqlDataAdapter.

Dans la classe Contexte1, on pourra implémenter cette méthode en renvoyant simplement la valeur 0.

**Etape 4 : Modification de la page des produits**

Dans le constructeur de la page des produits, si le contexte est de type Contexte2 :

* Ajouter une entrée de menu « 6 – Enregistrer les modifications », qui appelle une méthode interne EnregistrerModifsProduits, appelant elle-même la méthode du contexte et la méthode interne GérerErreurSql en cas d’erreur.
* Appeler la méthode InitDataSetProduits.

De cette façon, la page continuera de fonctionner quel que soit le type de contexte utilisé

**Etape 5 : Remplissage du DataSet**

Dans le contexte, implémenter la méthode GetProduits pour qu’elle remplisse le DataSet, et qu’elle retourne son contenu sous forme d’une liste de produits.

NB/ Pour rappel, la commande select a été créée dans la méthode InitDataSetProduits

**Etape 6 : Ajout ou modification d’un produit**

Implémenter la méthode AjouterModifierProduit de la façon suivante :

* Déclarer une ligne de donnée (DataRow) qui représentera le produit courant
  + Pour un ajout, il s’agira d’une nouvelle ligne crée en utilisant la méthode NewRow() de la DataTable
  + Pour une modification, il s’agira de la ligne à modifier, récupérée grâce à la méthode Find de la collection de lignes de la table.
* Affecter les champs de la ligne avec les valeurs des propriétés du produit passé en paramètre.  
  NB/ L'Id du produit est généré automatiquement par la base
* Appeler les méthodes BeginEdit et EndEdit de la ligne, respectivement avant et après l’affectation des valeurs de ses champs.
* Pour un ajout, ajouter la nouvelle ligne à la collection de lignes de la table

**Etape 7 : Suppression d’un produit**

Implémenter la méthode SupprimerProduit, en recherchant la ligne correspondant au produit dans la table (grâce à la méthode Find de la collection de lignes), puis en appelant la méthode Delete sur cette ligne, si elle a été trouvée.

**Etape 8 : Remplissage du DataSet avec modifications en cours**

Que se passe-t-il si on demande l’affichage de la liste des produits, alors que des lignes ont été modifiées ou supprimées dans le DataSet, et non enregistrées en base ?

Que faudrait-il faire pour gérer plus proprement ce cas ?

# Northwind2 avec Entity Framework

Cet exercice utilise la base de données Northwind2 et Entity Framework 6. Il illustre l’approche Code First avec base de données existante.

## Affichage de données

**Etape 1 : Ajout des références à EntityFramework**

Dans le menu contextuel du nœud « Références » du projet, sélectionner « Gérer les packages NuGet ».

Dans la fenêtre qui s’ouvre, cliquer sur l’onglet Parcourir, rechercher « Entity Framework », et l’installer. Ceci a pour effet de rajouter deux références au projet, ainsi que des entrées dans le fichier App.Config pour la gestion de la connexion (qui ne nous serons pas utile ici).

**Etape 2 : Création d’un troisième type de contexte**

Créer une classe Contexte3, qui dérive de DbContext, et qui implémente IDataContext de façon implicite. Cette classe utilisera EntityFramework pour dialoguer avec la base de données.

Créer un constructeur qui appelle le constructeur de base, en lui passant le nom de la connexion à la base en paramètre, sous la forme « name=… »

**Etape 3 : Récupération des pays et fournisseurs**

* Créer une entité Address contenant l’id et le pays.
* Sur l’entité Supplier, ajouter une propriété virtuelle Address de type Address, et une propriété AddressId de type Guid. Ceci permet de relier l’adresse au fournisseur. Faire en sorte qu’elles ne s’affichent pas à l’écran quand on affiche la liste des fournisseurs
* Dans la classe Contexte3, ajouter le code suivant :

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

   modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

}

Ceci permet de supprimer la convention qui fait que EF cherche des noms de tables au pluriel à partir des noms d’entités. Sans cela, EF chercherait les adresses dans une table nommée Addresses.

* Ajouter une propriété de type DbSet<Supplier>
* Implémenter les méthodes GetPaysFournisseurs et GetFournisseurs en utilisant des requêtes Linq to Entities sur le DbSet créé précédemment.

**Etape 4 : Récupération du nombre de produits**

Implémenter la méthode GetNbProduits en utilisant une requête sql exécutée via la méthode Database.SqlQuery

**Etape 5 : Récupération des clients et commandes**

Pour implémenter la méthode GetClientsCommandes, cette fois on ne récupérera pas le nombre d’articles ni le montant total de chaque commande.

* Ajouter une propriété CustomerId dans l’entité Orders (non affichable), de façon à ce que EF puisse faire automatiquement le lien entre les clients et leurs commandes.
* Ajouter l’attribut suivant sur l’entité Order, de façon à ce que EF puisse mapper automatiquement cette entité à la table Orders :

[Table("Orders")]

public class Order

* Sur les propriétés du nombre d’articles et du montant total de la commande, ajouter l’attribut [NotMapped] de façon à indiquer à EF de ne pas essayer de les mapper avec des champs de la base.
* Implémenter la méthode GetClientsCommandes, en utilisant un DbSet<Customer> déclaré en propriété de Contexte3 et une requête Linq to Entities optimisée pour la récupération de données sans suivi des modifications.

## Insertion, mise à jour et suppression de données

Cet exercice est la suite du précédent. Continuer à travailler dans le même projet.

L’objectif est ici d’implémenter la saisie de produits avec Entity Framework. Les ajouts, modifications et suppressions seront suivis par un DbSet, et enregistrés en base lorsque l’utilisateur le choisira.

**Etape 1 : Récupération des catégories de produits**

Implémenter la méthode GetCategories en utilisant un DbSet.

**Etape 2 : Récupération des produits**

Implémenter la méthode GetProduits à l’aide d’un DbSet, qui assurera le suivi des modifications.

**Etape 3 : Enregistrement des modifications en base**

* Dans le contexte, implémenter la méthode EnregistrerModifsProduits en appelant simplement SaveChange sur le contexte.
* Dans PageProduit :
  + Faire en sorte que le menu « 6 – enregistrer » s’affiche bien
  + Dans la méthode EnregistrerModifsProduits, intercepter l’erreur susceptible d’être remontée par EF si on tente de supprimer un produit référencé par une commande, et afficher le même message que dans l’exercice précédent  
    NB/ L’exception Sql à gérer est fournie par la propriété InnerException de l’exception levée par EF.

**Etape 4 : Ajout ou modification d’un produit**

Implémenter la méthode AjouterModifierProduit de la façon suivante :

* Pour un ajout, ajouter le produit au DbSet en utilisant sa méthode Add
* Pour une modification, rechercher le produit à modifier dans le DbSet à partir de son Id, puis affecter ses propriétés avec celles du produit passé en paramètre  
  NB/ L'Id du produit est généré automatiquement par la base

**Etape 5 : Suppression d’un produit**

Implémenter la méthode SupprimerProduit, en recherchant le produit correspondant dans le DbSet, puis en appelant la méthode Remove, si le produit a été trouvé.

Tester la suppression d’un produit référencé par une commande suivie d’un enregistrement, et vérifier que le message d’erreur défini précédemment s’affiche bien.

# Tests unitaires

L’objectif est de tester unitairement les méthodes des classes de contexte créées dans les exercices précédents

**Etape 1 : création du projet de tests**

Dans la même solution que les exercices précédents, ajouter un projet de tests unitaires nommé TestsNorthwind2

Dans ce projet, ajouter une référence au projet Nortwind2

**Etape 2 : création d’une classe de tests**

Générer une classe de tests pour la classe Contexte1 dans le projet créé précédemment. Utiliser pour cela le menu contextuel sur le nom de la classe et à la boîte de dialogue de création de tests unitaires. Renommer la classe de tests « TestsDataContext »

Ajouter une méthode d’initialisation dans laquelle on créera une instancie de Contexte1 qui sera utilisée pour l’ensemble des tests

**Etape 3 : implémentation des méthodes de tests**

Compléter les méthodes de tests avec les tests suivants :

|  |  |
| --- | --- |
| **Méthode de test** | **Description du test** |
| TestGetPaysFournisseurs | Vérifier qu’on récupère 16 pays et que le dernier est les USA |
| TestGetFournisseurs | Vérifier que les fournisseurs du Japon sont ceux d’id 6 et 4 |
| TestGetNbProduits | Vérifier que le Royaume Uni propose 7 produits |
| TestGetClientsCommandes | Vérifier qu’on récupère 91 clients et que le client RANCH a 5 commandes |
| TestGetCatégories | Vérifier qu’on récupère 8 catégories de produits et que la dernière est nommée « Seafood » |
| TestGetProduits | Vérifier qu’il y a 12 produits dans la catégorie Seafood et que le 7ème est le N° 40 |
| TestAjouterModifierProduit | Ajouter un nouveau produit dans la catégorie Cheeses et vérifier qu’il y a désormais 11 produits dans cette catégorie |
| TestSupprimerProduit | Supprimer le produit créé précédemment et vérifier qu’il y a de nouveau 10 produits dans la catégorie |
| TestAjouterProduitCategorie | Ajouter un nouveau produit sans catégorie, de façon à ce que la catégorie Others soit créée. Vérifier qu’il y a désormais 9 catégories |
| TestEnregistrerModifsProduits | Vérifier que la méthode d’enregistrement renvoie simplement 0, car elle est inutile dans ce DataContext |

**Etape 4 : exécution des tests**

Lancer la suite complète de tests, et le cas échéant, corriger ceux qui sont en erreur.

**Etape 5 : Tests de la classe Contexte2**

* Modifier la méthode d’initialisation pour créer une instancie de Contexte2 à la place de Contexte1.
* Modifier la dernière méthode de tests pour tester véritablement l’enregistrement
* Exécuter de nouveau les tests