# Entity framework Core

Entity Framework Core 8.0 (EF Core 8.0) est **un framework open-source** de Microsoft pour **simplifier l'accès** aux bases de données dans les applications .NET.

Il offre une manière plus simple de travailler avec les données en permettant aux développeurs de **manipuler des objets et des classes** plutôt que d'écrire des requêtes SQL directes, simplifiant ainsi le développement d'applications.

EF Core 8.0 agit comme un **ORM** (Object-Relational Mapping), facilitant le **mapping des objets du code aux tables** de la base de données et simplifiant les opérations de lecture, écriture et mise à jour des données.

<https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>

## Deux manières de l’utiliser

Entity Framework Core peut être utilisé de **deux manières** principales : "Database-First" et "Code-First". Voici une explication succincte des deux approches :

### Code-First **:**

Dans cette approche, les développeurs définissent **d'abord le modèle de données** dans leur code, **puis la base de données est générée** à partir de ce modèle. Les développeurs définissent des classes et des relations dans le code, puis **utilisent les migrations** pour **créer** ou mettre à jour **la base de données** en fonction de ces modèles.

* + Avantages **:** Offre une flexibilité totale dans la conception du modèle, permet aux développeurs de **travailler principalement dans le code** sans se soucier de la structure de la base de données.

### Database-First **(DB-First)** :

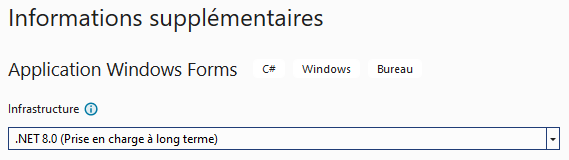
Dans cette approche, la **base de données existe déjà**, et le modèle de données est généré à partir de la structure de la base de données existante. Le développeur **génère des classes** de modèle à partir de la base de données existante à l'aide d'outils tels que la commande **Scaffold-DbContext** dans EF Core ou une extension comme EF Core Power tools permettant d’utiliser une interface graphique.

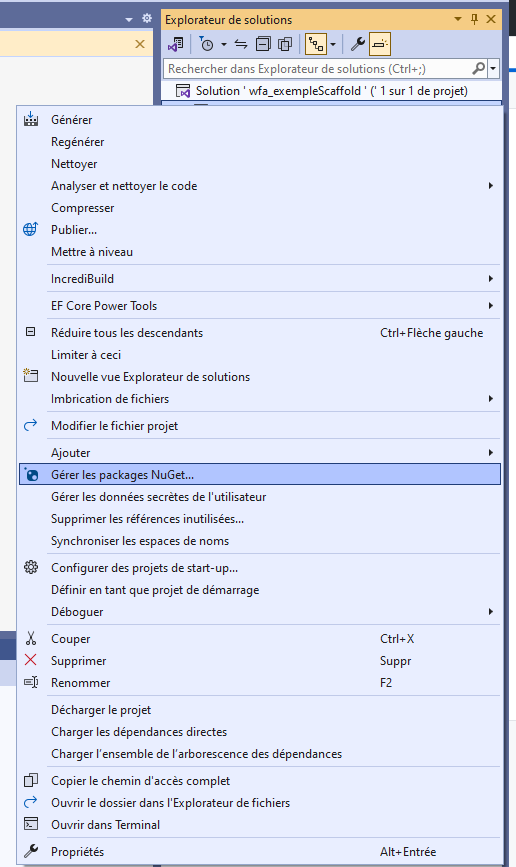
* + Avantages **:** utile, lorsque la **base de données est déjà conçue**, permet de créer **rapidement** un modèle à partir d'une base de données existante.

Nous utiliserons la méthode DB-first, et l’extension EF Core Power tools.

## Comment générer les classes de notre base de données dans Visual Studio

1. Dans Visual studio, choisissez : 
2. Puis assurez-vous d’utiliser le **Framework cible .Net 8.0**





1. Pour utiliser Entity Framework Core 8.0,

il nous faut **ajouter ces packages NuGet** (bouton droit sur le projet ) :

**Microsoft.EntityFrameworkCore**

Puis un package pour note base de données :

**Microsoft.EntityFrameworkCore.SqlServer**

Puis des outils pour la création de classe à partir de la BD :

**Microsoft.EntityFrameworkCore.Tools**

**Microsoft.EntityFrameworkCore.Design**

### Méthodes scaffold (structure)

Ouvrir la console de gestionnaire de package (menu Affichage, autre fenêtre, console de gestionnaire de package). Celle-ci nous permet de **faire exécuter des lignes de commande**.

Commande pour générer les entités et le contexte à partir de votre base de données existante ( à mettre sur 1 ligne) :

Scaffold-DbContext "Server=bd-tinfo;Database=ltScolaire;

Trusted\_Connection=True;

TrustServerCertificate=true"

Microsoft.EntityFrameworkCore.SqlServer

-OutputDir Models

Cette commande créera le context et les classes dans le répertoire Models, incluant les tables et les vues, **mais pas les procédures stockées**. Ce qui est un **désavantage**, nous devrons programmer ces classes nous-mêmes.

Plusieurs options sont possible avec la commande Scaffold-DbContext, voir <https://learn.microsoft.com/en-us/ef/core/cli/powershell>

Noter que la chaine de connexion est dans le OnConfiguring du context. Il serait mieux de la placer dans app.config et d’y faire référence. Ici le nom de la connectionString sera **maBD**.

<connectionStrings>  
 <add name ="maBD"  
 connectionString = "Server=bd-tinfo; Database=lt\_scolaire2;

Trusted\_Connection=True;

TrustServerCertificate=true"/>  
</connectionStrings>

Et on modifiera le OnConfiguring du context comme ceci :

optionsBuilder.UseSqlServer(

ConfigurationManager.ConnectionStrings["maBD"].ConnectionString);

Lorsqu’on fait des modifications, il est possible de seulement modifier certains objets, spécifiez les objets à mettre à jour et utiliser l’option -Force

### Méthode avec EF Power Tools

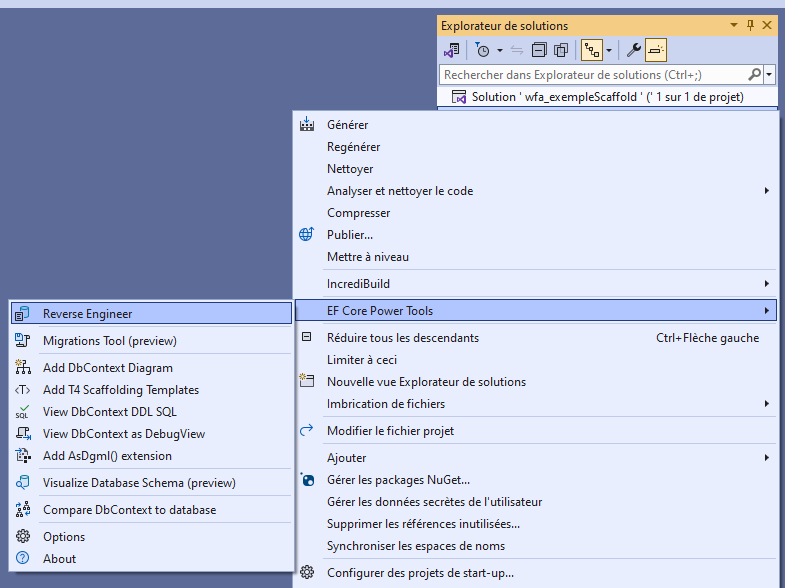
Extension à ajouter à Visual Studio.

EF Power Tools extension propose **une interface graphique** permettant de générer le model à partir du code (code-first) et/ou à partir de la base de données(db-first). Il offre des fonctionnalités visuelles pour explorer le modèle de données généré à partir de la base de données (création de diagramme).

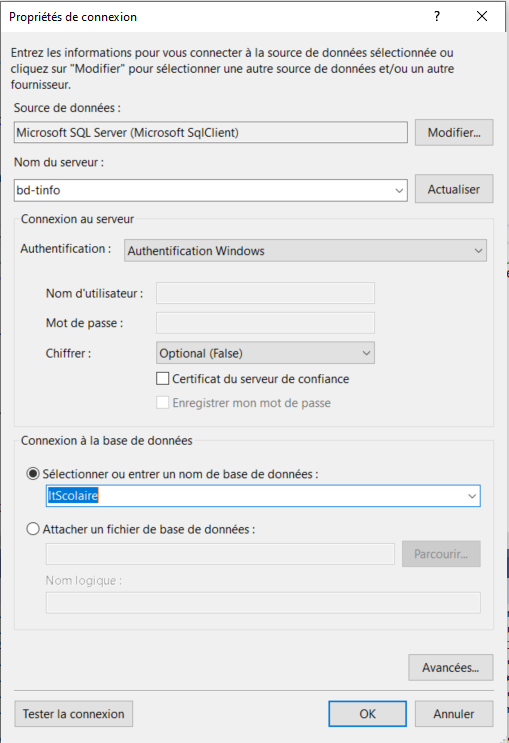
**Avantages** : Il génère le code pour prendre en charge les procédures stockées (pas fait avec la méthode scaffold).

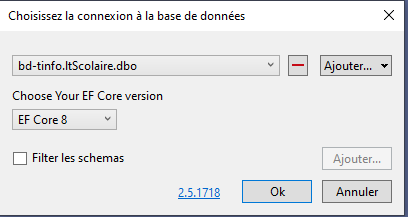
**Inconvénient** : Il écrase tous les éléments. Si de la programmation est ajoutée, elle sera écrasée.

Bouton droit sur votre projet, EF CoreTools, reverse Engineer.

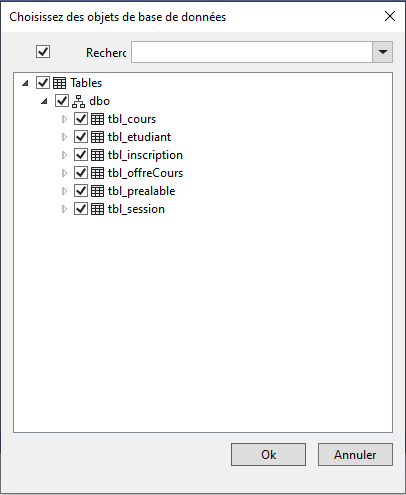


Choix de votre serveur et de votre base de données : N’oubliez pas de cocher chiffrer : optionnal

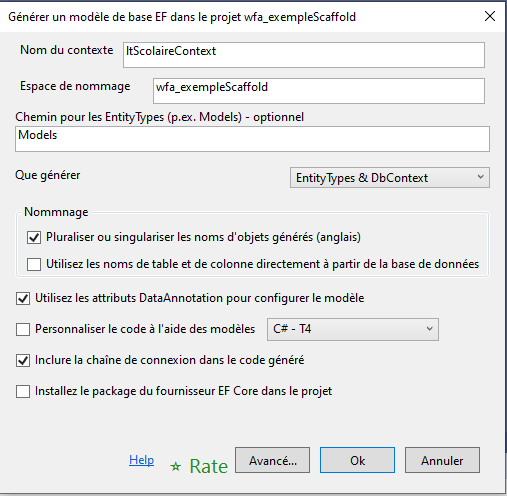




Choix de vos objets (tables, vues, procédure stockée)



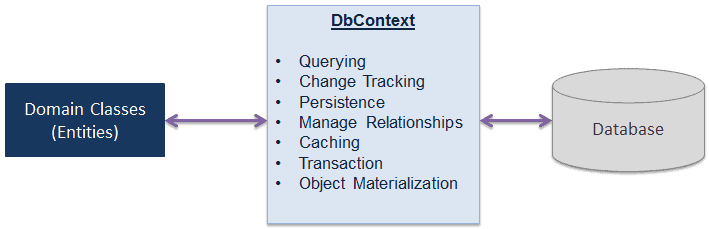
Choix de spécifications :



Comment obtenir les valeurs d’une table avec EF :

La classe DbContext fait partie intégrante d’entityFrameWork Core. C’est cette classe qui fait le lien entre notre BD et nos classes de domaine de notre application (nos tables, nos vues, nos procédures stockées…).

Une instance de DBContext représente une session avec la base de données. On pourra l’utiliser pour faire des requêtes, des ajouts/modification/destruction et des enregistrements à la base de données.

[[1]](#footnote-1)

using (var context = new ltScolaireContext ()) { …}

Avec **using**, on s’assure de faire un **dispose** du context à la fin du bloc. Selon notre besoin, on pourra aussi utiliser le context pendant l’exécution complète de notre Windows form (donc ne pas le fermer tout de suite).

Pour obtenir le contenu d’une table, il suffit d’utiliser **notreContext.notreTable** . Cette opération extrait les données. En BD1, on utilisait une connexion, une commande (select), un executeReader de la commande et le résultat était placé dans un SQLDataReader. Toutes ces actions sont exécutées de façon transparente.

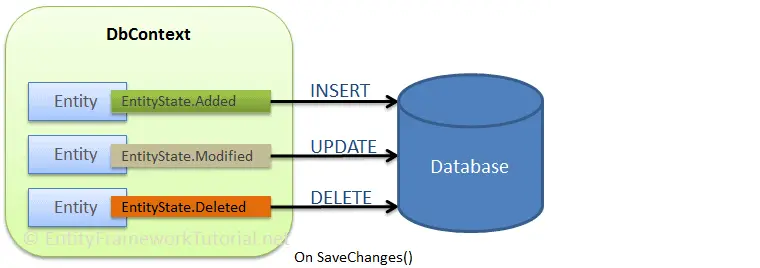
Exemple pour utiliser les données de la table cours dans le using précédent :

**context**.TblCours;

On pourra l’associer à un DataSource ou faire .ToList() sur le résultat pour le transformer en list.

# Principe du context

Référence : <https://www.entityframeworktutorial.net/efcore/saving-data-in-connected-scenario-in-ef-core.aspx>



Grâce au DbContext Entity Framework Core peut garder l’entityState de chaque ligne de celui-ci. Ainsi lors de l’appel de l’insctruction SaveChanges, il effectuera les opérations d’ajout, de modification ou de destruction nécessaires en utilisant l’entityState des lignes.

EntityState peut être unchanged, added, modified ou deleted.

Le saveChange a donc une action sur ce qui est dans le Dbcontext selon leur entityState.

## Voir les instructions SQL générés

Il est possible de voir les instructions sql générer avec le saveChanges. Il faut ajouter dans votre context.cs, dans la méthode OnConfiguring :

optionsBuilder.UseSqlServer("Data Source=bd-tinfo;Initial Catalog=ltScolaire;Integrated Security=True;Encrypt=False").LogTo(Console.Write, LogLevel.Information);

LogLevel.Information permet de ne voir que les instructions SQL, pas toutes les informations.

Il faut aussi ajouter : using Microsoft.Extensions.Logging; dans votre context.cs

Et s’assurer dans le menu **Projet** de Visual studio, **propriété** de votre projet, **général**, **type de sortie** : application console. C’est dans la console que vous verrez les instructions sql générées par le SaveChanges.

## Ajout

La méthode ADD du dbContext permet d’ajouter une entité (classe correspondant à une table) dans le context.

En utilisant ADD, l’entityState devient added. Vous pouvez utiliser l’instruction suivante pour le constater :

using (var context = new ltScolaireContext ())

{

var cours = new TblCour();

cours.NomCours = "objets connectés";

Création d’un cours

cours.NoCours = "4206PMBA";

ajout du cours dans le context (entityState added)

context.TblCours.Add(cours);

ajout du cours dans la BD

context.SaveChanges();

}

Vous pouvez faire afficher son entityState après et/ou avant le ADD avec cette instruction :

MessageBox.Show(context.**Entry**(cours).**State**.ToString());

Il suffit de faire SaveChanges pour que l’ajout se fasse dans la Base de données.

## Modification

Pour modifier un élément existant, il faut qu’il soit dans le context, puis nous devons y apporter une modification qui changera son entityState à modified.

Pour mettre l’entity dans le context, plusieurs options s’offrent à nous dont la méthode Find ou encore la méthode attach.

La méthode Find permet de rechercher à partir de la **clé primaire** de la table. Cette recherche placera la ligne trouvée dans le context. Il nous suffira de modifier la ligne, ce qui changera son entityState à modified.

SaveChanges enregistrera les modifications à la BD.

using (var context = new ltScolaireContext())

{

Met le cours dans le context en le recherchant

string noCours = "4204B2BA";

var cours = context.Find<TblCour>(noCours);

Modification du cours

entityState modified)

cours.NomCours = "Base de données II";

cours.Pond = "2-3-2";

context.SaveChanges();

Modification du cours dans la BD

}

Si dans notre application, nous avons déjà fait un find et avons fermé notre context, l’entity n’est plus dans le context (entityState est detached) mais peut encore être dans notre variable entity (ici cours) si elle est globale. Il nous est possible, au lieu de rechercher à nouveau, de prendre cet entity et de l’ajouter au context avec Attach.

using (var context = new ltScolaireContext())

Met le cours existant dans le context

{

context.TblCours.Attach(cours);

Modification du cours (entityState modified)

cours.NomCours = "Base de données 2";

cours.Pond = "2-3-3";

Modification du cours dans la BD

context.SaveChanges();

}

## Destruction

Pour détruire, la méthode Remove permet de mettre le entityState à removed, et le saveChange fera la destruction dans la base de données.

Il faut préalablement mettre l’entité dans le context. Nous pouvons utiliser les mêmes méthodes de recherche qu’avec la modification, soit find ou Attach ( si les données ont déjà été recherchées).

using (var context = new ltScolaireContext())

{

string noCours = "420101BA";

var cours = context.Find<TblCour>(noCours);

context.TblCours.Remove(cours);

context.SaveChanges();

}

# Utilisation d’un combobox

### Procédure

* Le comboBox peut être lié à une source de données (DataReader, DataSet, BindingSource, list…). Il permet d’afficher un contenu et de retourner une valeur différente sur sélection (par exemple la clé).
* Les propriétés sont :
  + **DisplayMember** : détermine la colonne affichée.
  + **ValueMember** : détermine la valeur retournée par ComboBox.SelectedValue
  + **Datasource** : est lié aux données. CombobBox.SelectedItem me retourne l’objet du datasource

Dans cet exemple, nous lierons le ComboBox à une liste contenant les données de notre table.

using (var context = new ltScolaireContext ())

{

cmbNomCours.**DisplayMember** = "nom\_cours";

cmbNomCours.**ValueMember** = "no\_cours";

cmbNomCours.**DataSource** = **context**.tbl\_cours.**ToList**(); }

On pourrait faire un **tri** sur le nom de cours, comme ceci :

cmbNomCours.DataSource = context.tbl\_cours.**OrderBy**(x=>x.**nom\_cours**).**ToList**();

**x=>x.nom\_cours** est appelé une **lambda expression**.

x est le paramètre (représente la table tbl\_cours).

=>indique une lamda Expression

x.nom\_cours spécifie le champ pour faire le tri.

En savoir plus : <https://www.tutorialsteacher.com/linq/linq-lambda-expression>

# Comment sélectionner une ligne de données

Il suffit d’utiliser la méthode FIND sur notre table (comme l’utilisation d’un where sur la clé):

<https://www.entityframeworktutorial.net/entityframework6/dbset.aspx>

using (var context = new ltScolaireContext ())

{

tbl\_cours **cours = context**.tbl\_cours.**Find**(1);

}

Celle-ci cherche le no\_cours 1. No\_cours étant la clé de la table.

On pourrait également utiliser une instruction SQL pour faire la recherche : (mettre : using System.Data.SqlClient; dans le haut de votre formulaire)

using (var context = new ltScolaireContext ())

{

tbl\_courscours = context.tbl\_cours.**SqlQuery** ("select \* from tbl\_cours where no\_cours = @no\_cours",

new **SqlParameter** ("@no\_cours",cmbNomCours.**SelectedValue**.ToString())).**FirstOrDefault**();

}

on pourrait également faire un where sur la table avec une méthode LINQ :

using (var context = new ltScolaireContext ())

{

string no\_cours = cmbNomCours.SelectedValue.ToString();

cours = context.tbl\_cours.**Where**(c => c.no\_cours == no\_cours).**FirstOrDefault**();

}

Par la suite, nous pouvons utiliser les colonnes de la ligne extraite, par exemple pour le placer dans un textBox :

nocoursTextBox.Text = **cours**.**no\_cours**;

1. Référence : Entityframeworktutorial.net [↑](#footnote-ref-1)