



Entity Framework





Introduction





Présentation d'Entity Framework

Entity Framework (EF) est

- ◆ Un framework
- ◆ ORM (Object Relational Mapping)

Développé par Microsoft pour les applications .NET.





Présentation de ce qu'est un ORM

Il permet aux développeurs de travailler avec des données relationnelles en utilisant des objets CLR (Common Language Runtime) plutôt que des commandes SQL.

Cela signifie que les développeurs peuvent utiliser des classes et des objets pour interagir avec la base de données plutôt que de devoir écrire du code SQL manuellement.





Entity Framework / ADO.NET

ADO.NET est un framework qui permet aux développeurs d'accéder aux données relationnelles en utilisant des commandes SQL.

De plus, ADO.NET est un framework très bas niveau qui nécessite beaucoup de code pour effectuer des opérations simples.





Entity Framework / ADO.NET

Exemple de code ADO.NET :

```
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    SqlCommand command = new SqlCommand("SELECT * FROM Customers", connection);
    using (SqlDataReader reader = command.ExecuteReader())
    {
        while (reader.Read())
        {
            Console.WriteLine("{0} {1}", reader.GetInt32(0), reader.GetString(1));
        }
    }
}
```





Entity Framework / ADO.NET

Entity Framework permet :

- ◆ De simplifier l'accès aux données relationnelles
- ◆ De simplifier la gestion des relations entre les tables
- ◆ De simplifier la gestion des transactions





Entity Framework / ADO.NET

Exemple de code Entity Framework :

```
using (var context = new MyContext())
{
    var customers = context.Customers.ToList();
}
```





Les providers supportés par EF

Entity Framework supporte plusieurs types de bases de données via des providers spécifiques :

- ◆ SQL Server
- ◆ MySQL
- ◆ SQLite
- ◆ Oracle
- ◆ PostgreSQL





Qu'est-ce qu'un provider ?

Un provider est un composant qui permet à Entity Framework de communiquer avec une base de données spécifique.

Dans le cas de MySQL, Entity Framework utilise le provider `MySql.Data.EntityFramework`.

Note : Il existe plusieurs providers pour chaque type de base de données.





Creation d'un projet Entity Framework

Nous allons créer votre premier projet Entity Framework.

Il nous faut :

- ◆ Visual Studio code
- ◆ Nuget
- ◆ Le package EntityFramework
- ◆ Le package MySql.Data.EntityFramework





Creation d'un projet Entity Framework

Nous souhaitons créer un projet Entity Framework , notre projet sera nommé EF.

-> Nous allons créer un projet console.

```
dotnet new console -o EF
```





Creation d'un projet Entity Framework

Nous voulons ajouter les Connecteurs Entity Framework pour MySQL.

```
dotnet add package MySql.Data.EntityFrameworkCore
```





Présentation des principaux composants

Les principaux composants d'Entity Framework sont :

- ◆ DbContext
- ◆ DbSet
- ◆ Entités
- ◆ Relations





Présentation de DbContext

DbContext est l'une des principales classes d'Entity Framework qui est utilisée pour gérer la connexion à la base de données et les opérations CRUD (Create, Read, Update, Delete) sur les données.

Il est également responsable de la gestion des relations entre les tables de la base de données.

Entity Framework utilise DbContext pour créer des requêtes SQL sur la base de données.





Présentation de DbContext

```
public class PizzaContext : DbContext
{
    public DbSet<> Pizzas { get; set; }

    public PizzaContext() {}

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseMySQL("Server=localhost;Database=pizzadb;User=testuser;Password=password");
    }
}
```





Le constructeur de DbContext

Le constructeur de la classe DbContext est utilisé pour initialiser une nouvelle instance de la classe avec les options de configuration spécifiques.

Il existe plusieurs constructeurs disponibles pour la classe DbContext, chacun ayant des options de configuration différentes.





La méthode OnConfiguring

La méthode `OnConfiguring` est utilisée pour configurer les options de `DbContext`.

Cette méthode est appelée lorsque `DbContext` est initialisé pour la première fois.

Si vous utilisez le constructeur de `DbContext` sans paramètres, cette méthode est appelée pour configurer les options de `DbContext`.





La méthode OnConfiguring

Cette méthode est appelée pour configurer les options de DbContext, dans notre cas nous utilisons le provider MySQL.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    optionsBuilder.UseMySQL("Server=localhost;Database=ma_db;User=mon_user;Password=mon_password");
}
```





La méthode OnConfiguring

Si vous utilisez d'autres providers, vous devez utiliser la méthode correspondante.

```
optionsBuilder.UseSqlServer  
optionsBuilder.UseSqlite  
optionsBuilder.UseOracle  
optionsBuilder.UseNpgsql  
etc...
```





You avez un password vide ?

Si vous avez un password vide, vous pourrez utiliser la méthode suivante :

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'nouveau_mot_de_passe';
```

En changeant le mot de passe de l'utilisateur root, vous pourrez utiliser le mot de passe dans la chaîne de connexion.





Présentation de la chaîne de connexion

La chaîne de connexion est composée de plusieurs éléments :

- ◆ Server : l'adresse du serveur de base de données
- ◆ User id : l'identifiant de l'utilisateur
- ◆ Password : le mot de passe de l'utilisateur
- ◆ Database : le nom de la base de données





Présentation des Entités

Une entité est un objet qui est stocké dans la base de données, il est également appelé modèle de données.

Les entités sont des classes qui représentent les tables de la base de données.

Elles vont contenir les propriétés qui correspondent aux colonnes de la table.





Présentation des Entités

Nous allons pouvoir créer une classe *Customer* :

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
}
```





Les entités doivent être publiques ?

Oui, les entités doivent être publiques.

Elles doivent être publiques pour que Entity Framework puisse les utiliser.

Ces entités devront :

- ◆ Être publiques
- ◆ Avoir un constructeur par défaut
- ◆ Avoir des propriétés publiques





Présentation de DbSet

DbSet est une collection d'entités qui sont stockées dans la base de données.

DbSet représente une table dans la base de données.

DbSet devrait être défini dans la classe DbContext.

```
DbSet<Customer> Customers { get; set; }
```





Utilisation de DbSet

DbSet est une collection d'entités qui sont stockées dans la base de données.

```
public class PizzaContext : DbContext
{
    public DbSet<Customer> Customers { get; set; }

    public PizzaContext(){}
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseMySQL("Server=localhost;Database=pizzadb;User=testuser;Password=password");
    }
}
```





Interaction entre DbContext, DbSet et Entités

Les entités sont stockées dans des collections DbSet qui sont définies dans la classe DbContext.

Nous devons définir une propriété DbSet pour chaque entité que nous souhaitons gérer.

Ainsi, chaque entité sera stockée dans une collection DbSet.





Présentation des principales méthodes de DbContext (Add, Remove, SaveChanges, etc.)

- ◆ ADD
- ◆ REMOVE
- ◆ SAVECHANGES
- ◆ FIND
- ◆ ENTRY





Présentation de la méthode Add

La méthode Add est utilisée pour ajouter une nouvelle entité à la base de données.

Elle prend en paramètre une entité et retourne une référence à l'entité ajoutée.

Dès que l'entité est ajoutée, elle est placée dans l'état Added.





Présentation de la méthode Add

Nous l'utiliserons de cette manière :

```
var customer = new Customer  
{  
    Name = "Customer 1",  
    Address = "Address 1"  
};  
context.Customers.Add(customer);
```





Présentation de la méthode Remove

La méthode Remove est utilisée pour supprimer une entité de la base de données.

Elle prend en paramètre une entité et retourne une référence à l'entité supprimée.

```
context.Customers.Remove(customer);
```





Présentation de la méthode SaveChanges

La méthode `SaveChanges` est utilisée pour enregistrer les modifications apportées aux entités dans la base de données.

Elle retourne le nombre d'entités modifiées.

```
int changes = context.SaveChanges();
```





Présentation de la méthode Find

La méthode Find est utilisée pour rechercher une entité dans la base de données.

Elle prend en paramètre la clé primaire de l'entité et retourne une référence à l'entité trouvée.

```
Customer customer = context.Customers.Find(1);
```





Comment Entity Framework sait-il que la clé primaire est l'Id ?

Entity Framework sait que la clé primaire est l'Id car nous avons défini une propriété Id dans la classe Customer.

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
}
```





Présentation de la classe Repository

La classe Repository est une classe générique qui permet de gérer les entités.

Elle contient une propriété DbSet qui représente la collection d'entités.

Si elle est créée avec un template, elle peut gérer n'importe quelle entité.



```
public class Repository
{
    private DbContext _context;

    public Repository(DbContext context){
        _context = context;
    }

    public void Add(User user){
        _context.Users.Add(user);
    }

    public void Remove(User user){
        _context.Users.Remove(user);
    }

    public int SaveChanges(){
        return _context.SaveChanges();
    }

    public User Find(int id){
        return _context.Users.Find(id);
    }
}
```



Le principe de transaction

Une transaction est un ensemble d'instructions qui doivent être exécutées de manière atomique.

Si une instruction échoue, toutes les instructions doivent être annulées.

Elle a pour but d'éviter les données corrompues.





Présentation de la méthode Transaction

La méthode Transaction permet d'exécuter un ensemble d'instructions de manière atomique.

Elle prend en paramètre un bloc de code qui contient les instructions à exécuter.

Il devrait être utilisé dans un bloc using.





Présentation de la méthode Transaction

```
using (var transaction = context.Database.BeginTransaction())
{
    try
    {
        // Instructions
        transaction.Commit();
    }
    catch (Exception)
    {
        transaction.Rollback();
    }
}
```





Présentation de la méthode Commit

La méthode Commit permet de valider les modifications apportées à la base de données.

Elle est utilisée dans un bloc try.

Elle valide toutes les modifications apportées à la base de données depuis le début de la transaction.





Présentation de la méthode Rollback

La méthode Rollback permet d'annuler les modifications apportées à la base de données.

Elle est utilisée dans un bloc catch.

Elle annule toutes les modifications apportées à la base de données depuis le début de la transaction.





Les Lambda Expressions





Présentation des Lambda Expressions

Les lambda expressions sont des fonctions anonymes qui peuvent être utilisées pour créer des expressions.

Elles permettent :

- ◆ De créer des fonctions anonymes
- ◆ De créer des expressions
- ◆ De créer des délégués





Présentation des Lambda Expressions

Les lambda se composent de 3 parties :

- ◆ Les paramètres
- ◆ L'opérateur de flèche
- ◆ Le corps de la fonction

```
(x, y) => x + y
```





Présentation des Lambda Expressions

Les paramètres sont définis entre parenthèses.

Si nous avons qu'un seul paramètre, nous pouvons omettre les parenthèses.

```
x => x + 1
```





Création d'un projet DataBaseFirst





Création d'un projet DataBaseFirst

Pour créer un projet DataBaseFirst, nous devons utiliser les commandes suivantes :

- ◆ Dotnet add package
`Microsoft.EntityFrameworkCore.Design`
- ◆ Dotnet add package
`Microsoft.EntityFrameworkCore.SqlServer`
- ◆ Dotnet add package
`Microsoft.EntityFrameworkCore.Tools`





Création d'un projet DataBaseFirst

Après avoir installé les packages, nous pouvons créer le projet DataBaseFirst avec la commande suivante :

```
dotnet ef dbcontext scaffold "Server=localhost;Port=3306;Database=SolarSystem;User=root;Password=0000;"  
MySql.EntityFrameworkCore -o Models
```

Ensuite, nous devons créer un projet Console et ajouter le projet Models.



Les liaison entre les entités



La relations One-to-One





Représenter une relation entre deux entités

Si nous avons deux entités Customer et Order, nous pouvons représenter la relation entre les deux entités de cette manière :

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public Order Order { get; set; }
}
```





Représenter une relation entre deux entités

Nous pouvons représenter la relation entre les deux entités de cette manière :

```
public class Order
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public Customer Customer { get; set; }
}
```





Présentation de la relation One-to-One

La relation One-to-One est une relation entre deux entités où une entité est liée à une seule autre entité.

Nous pouvons représenter la relation One-to-One de cette manière dans la méthode OnModelCreating de la classe DbContext :

```
modelBuilder.Entity<Customer>()
    .HasOne(c => c.Order)
    .WithOne(o => o.Customer);
```





Présentation de la relation One-to-One

La méthode Entity prend en paramètre le type de l'entité Customer.

Elle retourne une référence à l'entité Customer.

La méthode HasOne prend en paramètre une expression lambda qui retourne la propriété qui représente la clé étrangère.

```
modelBuilder.Entity<Customer>()
    .HasOne(c => c.Order)
```





Présentation de la relation One-to-One

La méthode WithOne prend en paramètre une expression lambda qui retourne la propriété qui représente la clé primaire.

```
modelBuilder.Entity<Customer>()
    .HasOne(c => c.Order)
    .WithOne(o => o.Customer);
```





La relation One-to-Many





Présentation de la relation One-to-Many

La relation One-to-Many est une relation entre deux entités où une entité est liée à plusieurs autres entités.

Une entité Customer peut avoir plusieurs entités Order.

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public List<Order> Orders { get; set; }
}
```





Présentation de la relation One-to-Many

Nous pouvons représenter la relation One-to-Many de cette manière dans la méthode OnModelCreating de la classe DbContext :

```
modelBuilder.Entity<Customer>()
    .HasMany(c => c.Orders)
    .WithOne(o => o.Customer);
```





Présentation de la relation One-to-Many

La méthode Entity prend en paramètre le type de l'entité Customer.

Elle retourne une référence à l'entité Customer.

La méthode HasMany prend en paramètre une expression lambda qui retourne la propriété qui représente la clé étrangère.

```
modelBuilder.Entity<Customer>()
    .HasMany(c => c.Orders)
```





Présentation de la relation One-to-Many

La méthode WithOne prend en paramètre une expression lambda qui retourne la propriété qui représente la clé primaire.

```
modelBuilder.Entity<Customer>()
    .HasMany(c => c.Orders)
    .WithOne(o => o.Customer);
```





La relation Many-to-Many





Présentation de la relation Many-to-Many

La relation Many-to-Many est une relation entre deux entités où une entité est liée à plusieurs autres entités.

Une entité Customer peut avoir plusieurs entités Order.

```
public class Customer
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public List<Order> Orders { get; set; }
}
```





Présentation de la relation Many-to-Many

Nous pouvons représenter la relation Many-to-Many de cette manière dans la méthode OnModelCreating de la classe DbContext :

```
modelBuilder.Entity<Customer>()
    .HasMany(c => c.Orders)
    .WithMany(o => o.Customers);
```

