ADO.Net Entity Framework

Mme. Fatima-Ezzahra AIT BENNACER
f.aitbennacer@emsi.ma

2025 - 2026

Plan du cours

1. Introduction à ORM
2. Entity framework
3. L'architecture de Entity Framework
4. L'approche Database First
5. Cas pratique

Introduction à ORM

ORM: Mapping Objet-Relationnel

- Programme informatique jouant le rôle du traducteur entre le modèle **relationnel** et le modèle **objet**.
- Permet d'interroger et manipuler les données à partir d'une base de données à l'aide d'un paradigme orienté objet.

- Deux composants dans les ORM :
  - **Entités:** Instanciation d'une classe (étudiant, enseignant, cours, projet, etc).
  - **Gestionnaire d'entités:** à utiliser pour persister les entités dans la base de données.

Entity Framework

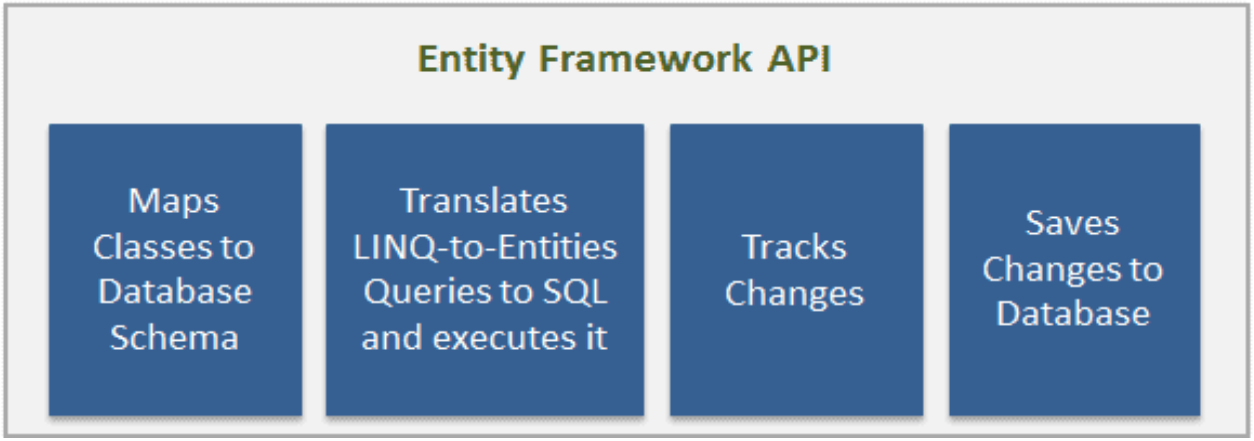
Entity Framework

- Entity Framework est un framework ORM open source pour les applications .NET prises en charge par **Microsoft** permettant de créer une couche d'accès aux données liées à une base de données relationnelle.
- Permettant aux développeurs de manipuler des données à l'aide d'objets de classes C# sans se concentrer sur les tables et colonnes d'une base de données (où ces données sont stockées).
- Permettant aux développeurs de créer et maintenir des applications orientées données avec moins de code, par rapport aux applications traditionnelles, grâce à **LinQ vers les entités.**

Le fonctionnement de Entity Framework

- **Dans le cas d'Entity Framework**
  - Entité (Une classe qui correspond à une table de base de données)
  - Le gestionnaire d'entités : *LinQ to Entities*

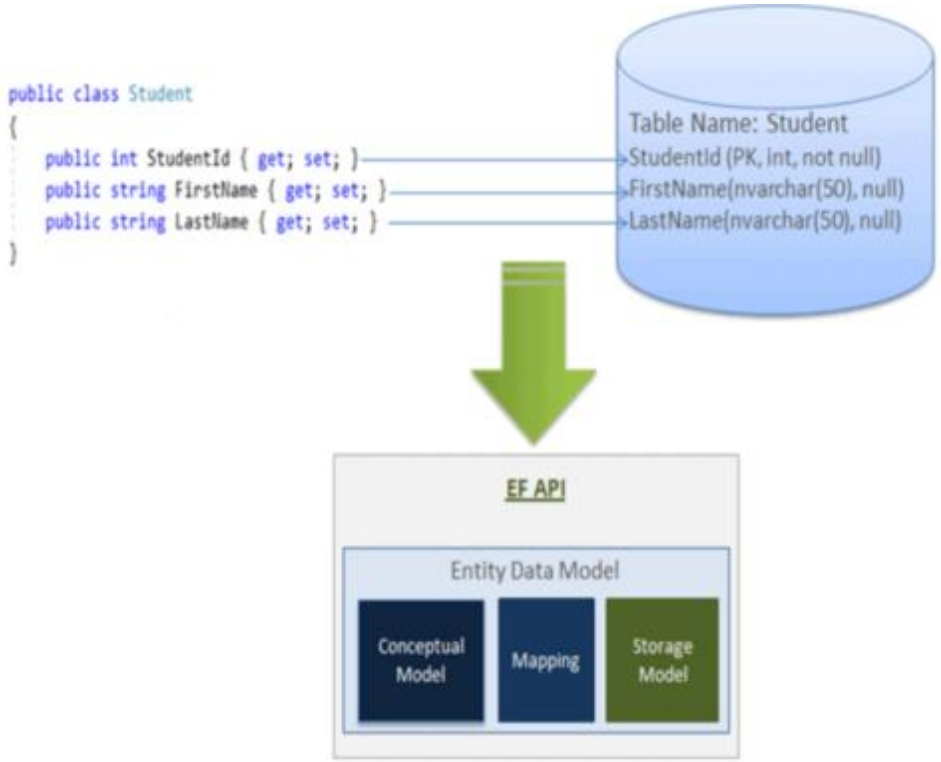
- **Fonctionnement d'Entity Framework**
  - L'API Entity Framework a la possibilité de:



Le fonctionnement de Entity Framework

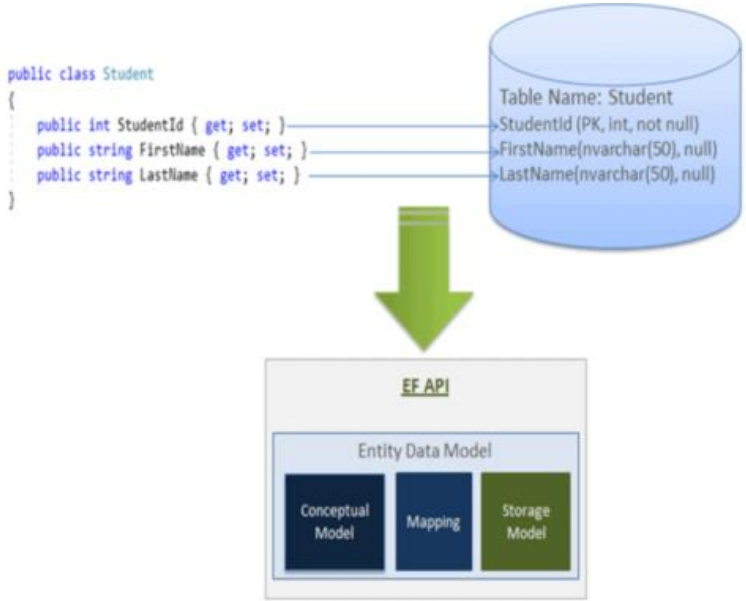
- **EDM: Entity Data Model**
  - EDM est une représentation en mémoire de l'ensemble des métadonnées

- **EDM consiste en trois parties:**
  - **Conceptuelle :** contient les classes du modèle et ses relations.
  - **Stockage :** contient le modèle physique de la base : tables, vues, procédures stockées, les relations et les clés.
  - **Mapping :** définit les mécanismes de passage du modèle conceptuel au stockage.



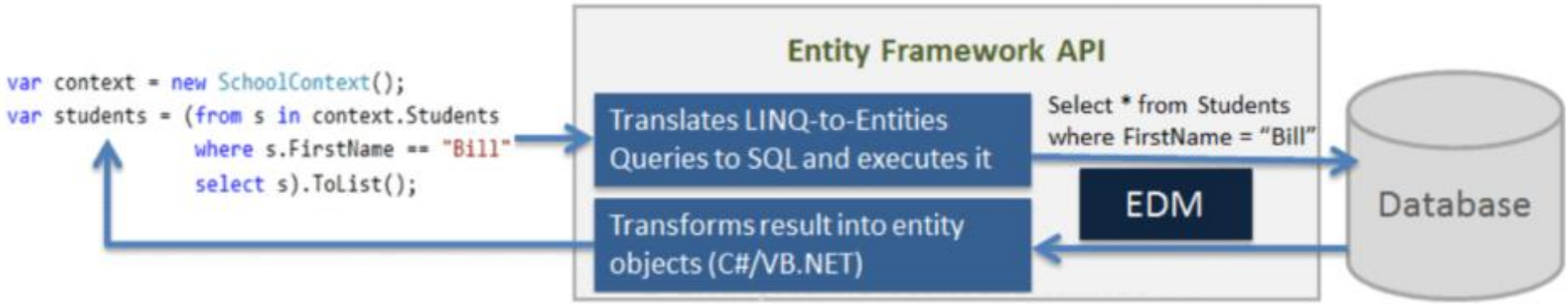
Maps Classes to Database scheme

- **A l'aide de EDM, EF peut:**
  - Effectuer des opérations **CRUD** (Create, Read, Update and Delete).
  - Créer des requêtes **SQL** à partir de requêtes **LINQ**, créer des commandes **INSERT, UPDATE et DELETE** et transformer le résultat de la base de données en objets d'entité.



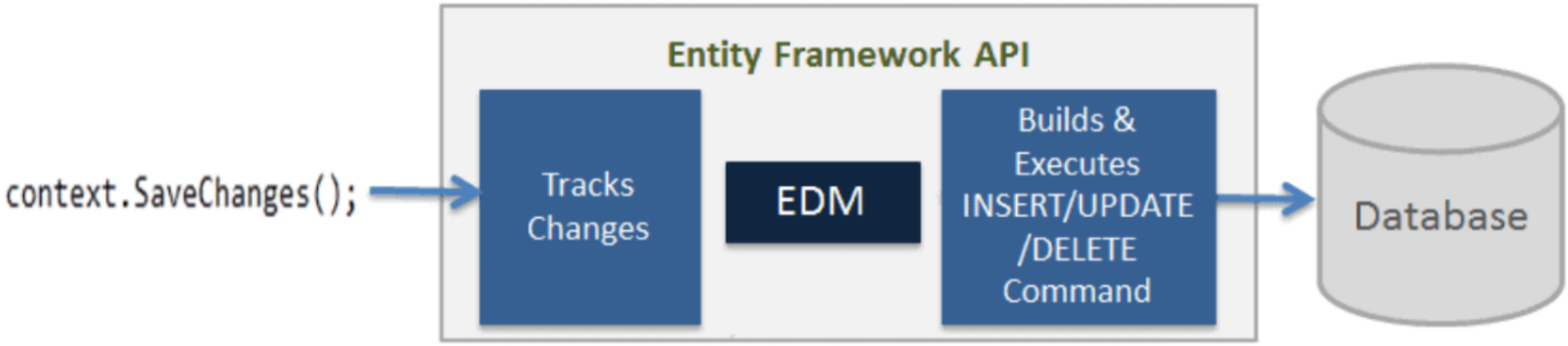
Translates LINQ-to-Entities Queries to SQL

- **À l'aide d'EDM, l'API EF:**
  - Traduit les requêtes LINQ-to-Entities en requêtes SQL pour les bases de données relationnelles.
    - LinQ to Entity : Un langage de requête pour un modèle orienté objet.
    - Il retourne les entités définies dans le modèle conceptuel.
  - Reconvertit les résultats (requêtes SQL) en objets d'entité.



Tracks and save changes to database

- Lorsque la méthode **SaveChanges()** est appelée, l'API EF déduit les commandes INSERT, UPDATE et DELETE en fonction de l'état des entités.
- Le **ChangeTrack** suit les changements survenus sur les entités



L'architecture de Entity Framework

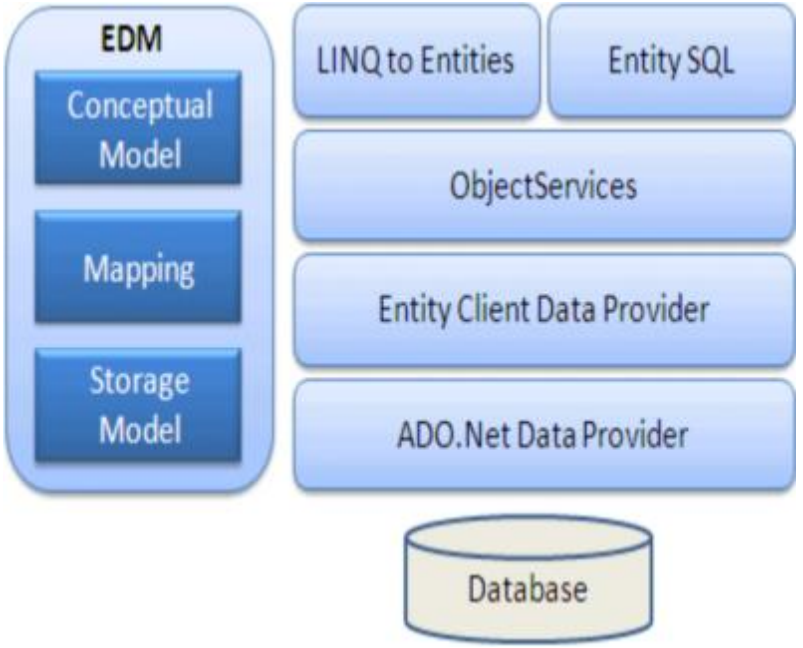
L'architecture de Entity Framework

EDM and LinQ to Entities

- **Object Service :** est le point d'entrée pour l'accès aux données d'une base de données. Il convertit les résultats SQL en objets C#

- **Entity Client Data Provider :** la tâche principale est de transformer une expression de LinQ à une requête SQL.

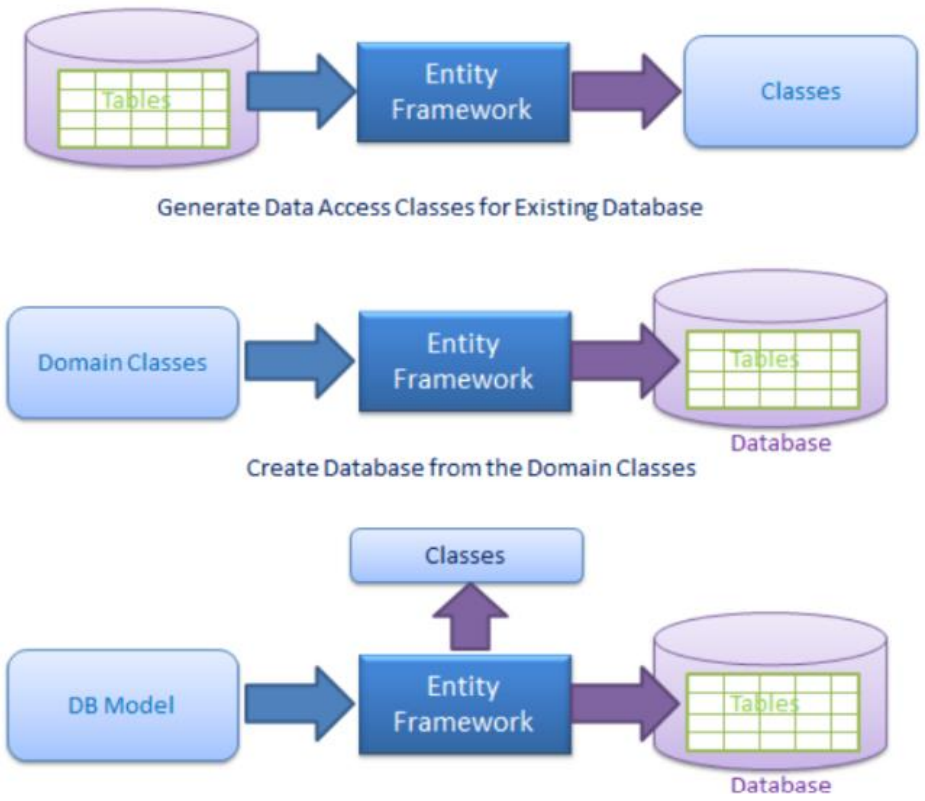
- **ADO.Net Data Provider :** Gère la communication technique avec la BD



Les différentes approches pour créer une base de données

Trois approches:

1. **Database First :** On crée la base de données (où on a une base de données qui existe déjà) et Entity Framework génère nos entités à partir de cette base de données.
2. **Code First :** On crée les entités puis et Entity Framework génère la base de données.
3. **Model First :** On crée notre modèle (de classe) et Entity Framework génère la base de données et les entités correspondantes.



Database First

Database First :

- Installer MS SQL (Microsoft Structured Query Language) et SSMS (SQL Server Management Studio).
- Créer une base de données (BD VENTE) sous SSMS

- CREATE DATABASE BD VENTE; use BD VENTE;
- CREATE TABLE PRODUITS (

```
        ID INT PRIMARY KEY IDENTITY(1,1),
        NOM VARCHAR(100) NOT NULL,
        DESCRIPTION VARCHAR(255),
        PRIX DECIMAL(10,2) NOT NULL,
        QUANTITE INT NOT NULL,
        DATE_AJOUT DATE DEFAULT GETDATE(),
        DISPONIBLE BIT DEFAULT 1
    );
```

Database First

Database First :

- Insérer des champs:

```
INSERT INTO PRODUITS (NOM, DESCRIPTION, PRIX, QUANTITE)
VALUES
('Ordinateur Portable', 'PC 15 pouces, i5, 8Go RAM', 799.99, 12), ('Souris Optique', 'Souris USB avec capteur optique', 19.99, 50);
```

	ID	NOM	DESCRIPTION	PRIX	QUANTITE	DATE_AJOUT	DISPONIBLE
1	1	Ordinateur Portable	PC 15 pouces, i5, 8Go RAM	799.99	12	2025-11-02	1
2	2	Souris Optique	Souris USB avec capteur optique	19.99	50	2025-11-02	1

Création Application Web : étapes à suivre

- Créer un nouveau projet sous Visual Studio **Fichier → Nouveau → Projet**.
- Sélectionner Application web ASP.NET Core MVC
- Remplir le champs Nom par: App_ACHAT.
- Valider et attendre la fin de création du projet.

Étapes à suivre

Installation des packages NuGet nécessaires (EF Core)

- **Méthode 1 – Interface graphique :** Outils > Gestionnaire de packages NuGet > Gérer les packages NuGet pour la solution
Chercher et installer ces packages :

    Microsoft.EntityFrameworkCore
    Microsoft.EntityFrameworkCore.SqlServer
    Microsoft.EntityFrameworkCore.Tools

- **Méthode 2 – Console :** Outils > Gestionnaire de packages NuGet > Console

    Install-Package Microsoft.EntityFrameworkCore
    Install-Package Microsoft.EntityFrameworkCore.SqlServer
    Install-Package Microsoft.EntityFrameworkCore.Tools

Connecter le serveur sql avec visual studio

ASP.Net

- Faire un clic sur Nouvelle connexion et saisir le nom du serveur (Faire un clic sur Object explorer dans SSMS).
- Sélectionner le nom de la base de données BD VENTE.

ASP.Net Core

1- Générer les modèles (Database First)

- Exécuter la commande suivante dans la console NuGet :

```
Scaffold-DbContext "Data Source=(localdb)\MSSQLLocalDB;Initial
Catalog=BD_VENTE;Integrated Security=True;TrustServerCertificate=True"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context
VenteContext
```

Cela crée : VenteContext.cs (classe DbContext) et Produit.cs (classe Model)

Connecter le serveur sql avec visual studio

2- Ajouter la chaîne de connexion dans appsettings.json :

```
"ConnectionStrings": {
  "VenteDb": "Data Source=(localdb)\\MSSQLLocalDB;Initial
Catalog=BD_VENTE;Integrated
Security=True;TrustServerCertificate=True"

}
```

3- Ajouter le DbContext dans Program.cs :

```
builder.Services.AddDbContext<VenteContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("VenteDb")));
```

Connecter le serveur sql avec visual studio

Le modèle BdVenteContext.cs:

- Une classe héritant de la classe DbContext est appelée classe de contexte (context class) dans le framework d'entité.
- BdVenteContext hérite de la **classe System.Data.Entity.DbContext**

DbContext

- **DbContext** est la classe principale responsable de l'interaction avec la base de données. Il est responsable de plusieurs activités:

  - **Querying:** convertit les requêtes LINQ-to-Entities en requêtes SQL et les envoie à la base de données.
  - **Change Tracking :** assure le suivi des modifications apportées aux entités après une requête à partir de la base de données.
  - **Persisting Data :** effectue les opérations d'insertion, de mise à jour et de suppression dans la base de données, en fonction des états de l'entité.
  - **Object Materialization :** convertit les données brutes de la base de données en objets d'entité.

DbSet

- La classe de contexte (BD VENTEContext) inclut l'ensemble d'entités de type DbSet<TEntity>.
- DbSet est considérée comme une propriété de la classe de contexte et liée à une table de la base de données.
- DBSet contient un ensemble de méthodes permettant d'effectuer les opérations CRUD nécessaires.

- DbContext correspond à votre base de données (ou à une collection de tables et de vues dans votre base de données).
- DbSet correspond à une table ou une vue dans votre base de données.

ProduitsController: les opérations CRUD

Lire (Read)

```
public async Task<IActionResult> Index()
{
    return View(await _context.Produits.ToListAsync());
}
```

- Récupère la liste de tous les produits depuis la base de données.
- Retourne la vue Index.cshtml avec les données.

ProduitsController: les opérations CRUD

Créer (Create) POST

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind(...)] Produit
produit)
```

- Récupère les données saisies dans le formulaire.
- Les ajoute à la base de données avec _context.Add(...).
- Enregistre avec SaveChangesAsync().
- Redirige vers Index() après création

ProduitsController: les opérations CRUD

Mettre à jour (Update): Edit(int id, Produit produit)

```
[HttpPost]
public async Task<IActionResult> Edit(int id, [Bind(...)] Produit
produit)
```

- Vérifie que l'id correspond bien à l'objet envoyé.
- Met à jour le produit dans la base avec _context.Update(...).
- Gère les erreurs de concurrence avec try...catch.

ProduitsController: les opérations CRUD

Supprimer (Delete): DeleteConfirmed(int id) (POST)

```
[HttpPost, ActionName("Delete")]
public async Task<IActionResult> DeleteConfirmed(int id)
```

- Supprime réellement le produit avec _context.Produits.Remove(...).
- Enregistre les changements.

TP N°5