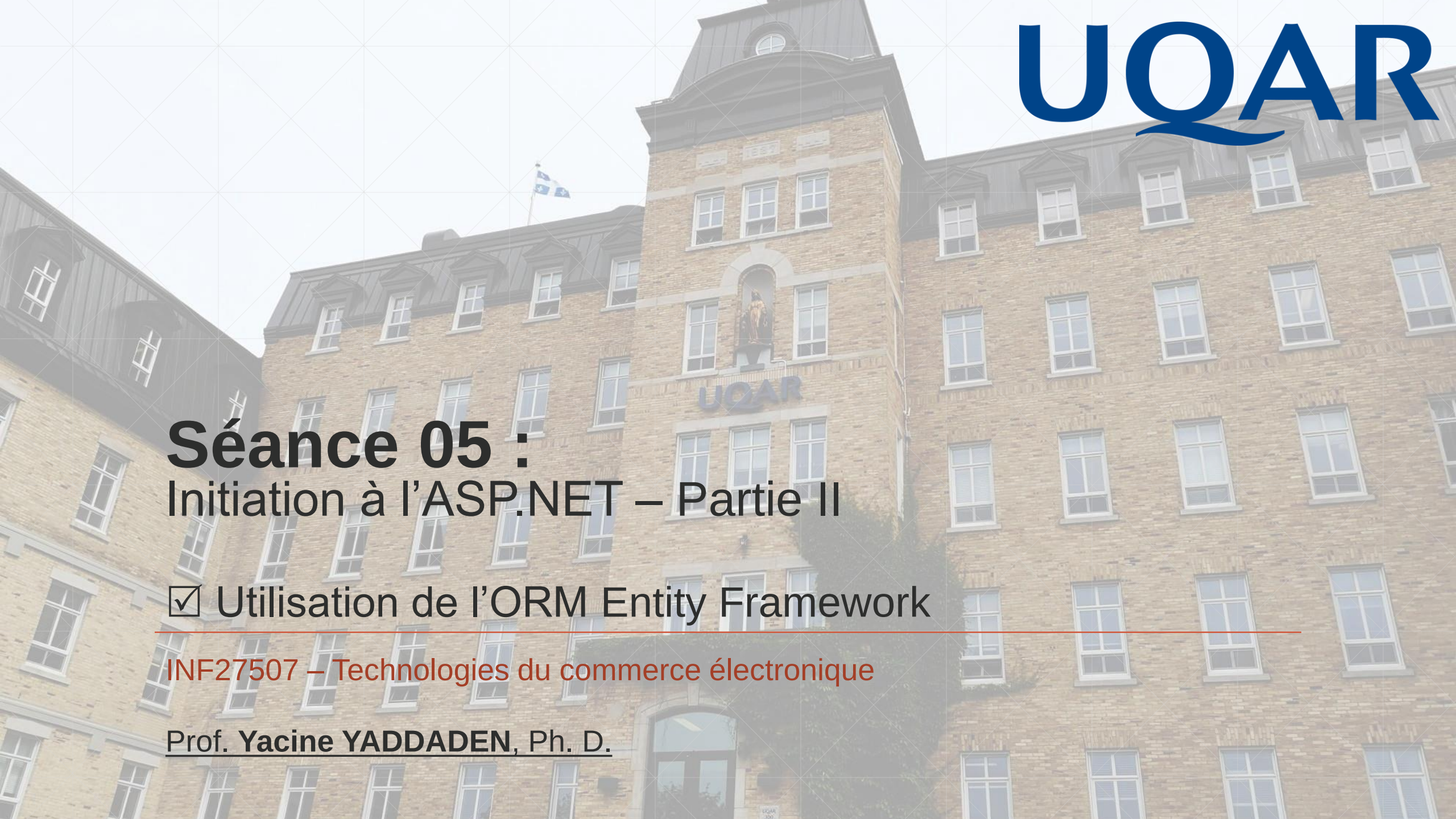


Séance 05 :
Initiation à l'ASP.NET – Partie II

☑ Utilisation de l'ORM Entity Framework

INF27507 – Technologies du commerce électronique

Prof. **Yacine YADDADEN**, Ph. D.

Plan

1. Introduction et Contexte
2. Persistance de données et **ORM**
3. *Paradigme* **MVC**
4. Les fondamentaux de **Microsoft Entity Framework**
5. Utilisation avec **MySQL Server**
6. Mise en application
7. Questions et discussion

Introduction et Contexte

- **Contexte**
  - Dans les programmes, applications ou logiciels, il est question de *données*,
  - Ces dernières doivent être *efficacement gérées* et *facilement accessibles*.
- **Problématique**
  - Habituellement, les données sont *stockés en mémoire volatile*,
  - Une fois le programme ou application *fermé*, les données sont *perdue*,
  - Comment garder ses données de manière *permanente* ?
- **Solution**
  - Faire appel à la **persistance** des données qui se présente sous *différentes formes*.

Persistance de données

- **Définition :** *« c'est l'ensemble des mécanismes mis en œuvre afin de permettre la sauvegarde des données de façon permanente. »*,
- **Objectif**
  - Conserver les données même après fermeture de l'application ou du programme.
- **Les différentes formes :**
  - Il est possible d'utiliser de simples fichiers *sans structure*, mais
    - Ça engendre des problèmes et de difficultés d'accès et d'utilisation des données.
  - Il est également possible d'utiliser des fichiers *structurés* ou *formattés*
    - Exemples : JSON, XML, CSV, …
    - Il y a amélioration, mais il y a aussi des *limitations* liées à la *quantité de données*.

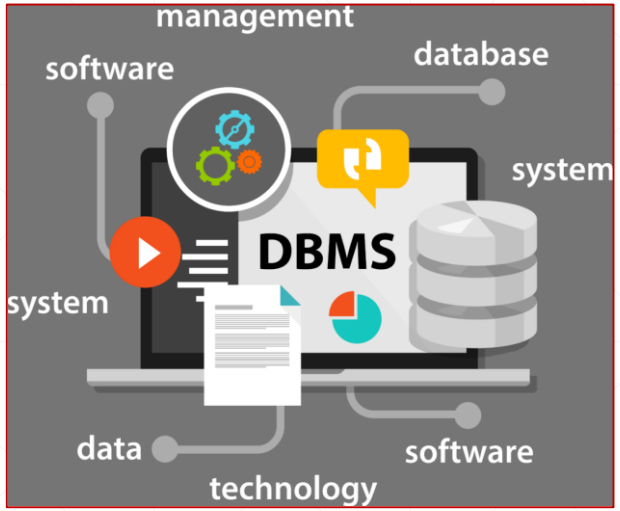
Limitations des fichiers

- ✗ Il est nécessaire de lire tout le fichier afin de récupérer une certaines donnée,
- ✗ Il n'est pas possible de lancer des requêtes personnalisées,
- ✗ La gestion et l'accès aux données n'est ni sécurisé ni optimisé,
- ✗ Risque accrue d'erreur et de perte de données,
- ✗ Temps de réponse important dans le cas de grande masse de données,
- ✗ …

✓ La solution consiste à utiliser un **SGBD**

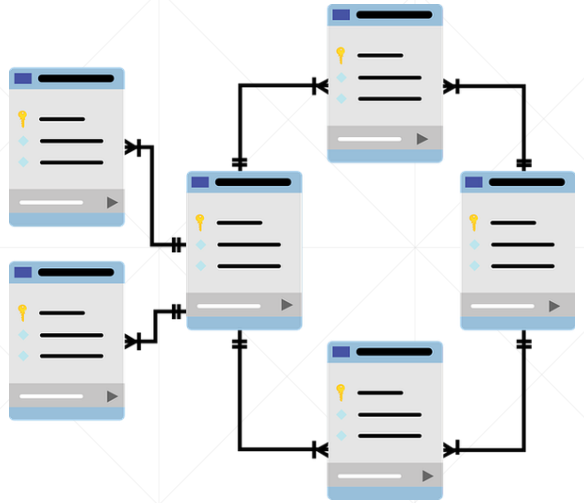
SGBD ou DBMS

- **Définition :** *« **S**ystème de **G**estion de **B**ases de **D**onnées ou **D**ata**B**ase **M**anagement **S**ystem est programme ou logiciel permettant le stockage, la manipulation et la gestion des données. »,*
- Il y a deux types de **SGBD** basés sur :
  - **SQL** : ***S**tructured **Q**uery **L**anguage*
    - *Ou **L**angage de **R**equête **S**tructurée*
  - **NoSQL** : ***No** **S**tructured **Q**uery **L**anguage*
    - *Ou **L**angage de **R**equête **No**n **S**tructurée*
- **SQL** est plus ancien *(représentation sous forme de tables)*,
- **NoSQL** est plus flexible *(Big Data)*.



Bases de données *relationnelles*

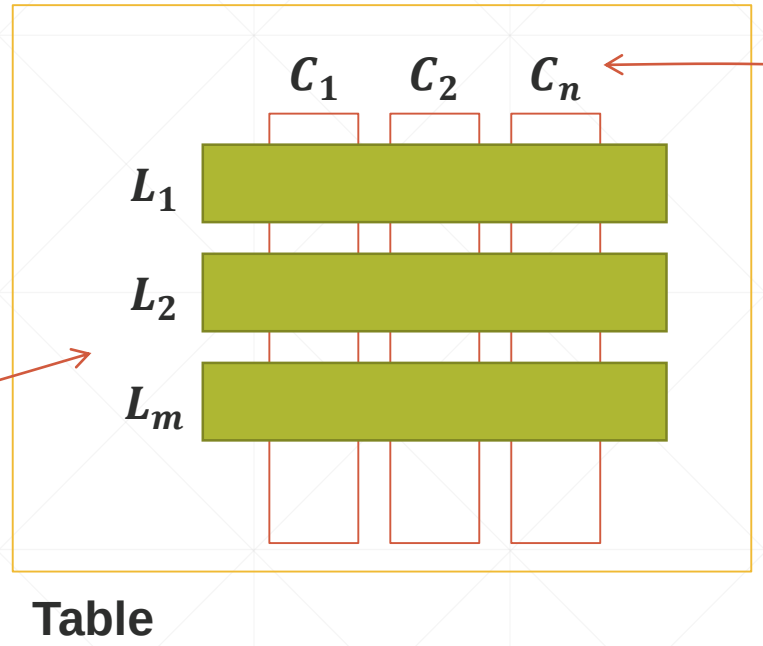
- C’est un type de base de données qui est représenté sous forme de *tableaux à deux dimensions* : **relations** ou **tables**,
- Les *colonnes* représentent les *champs* ou *attributs*,
- Les *lignes* représentent les *données* ou *enregistrements*,
- Les lignes sont *indépendantes* les unes des autres,
- Ce modèle a été introduit par **Edgar Frank Codd** en **1970**,
- Il utilise le *langage* **SQL** pour les *requêtes*.



- Il est possible de créer des *liaisons*, *associations* ou *jointures* entre les tables.

Bases de données *relationnelles*

*Les données représentant les personnes : **Albert**, **Romain**, **Julie**, …*



Table

*Les différents champs ou attributs, exemple : **id**, **nom**, **prénom**, …*

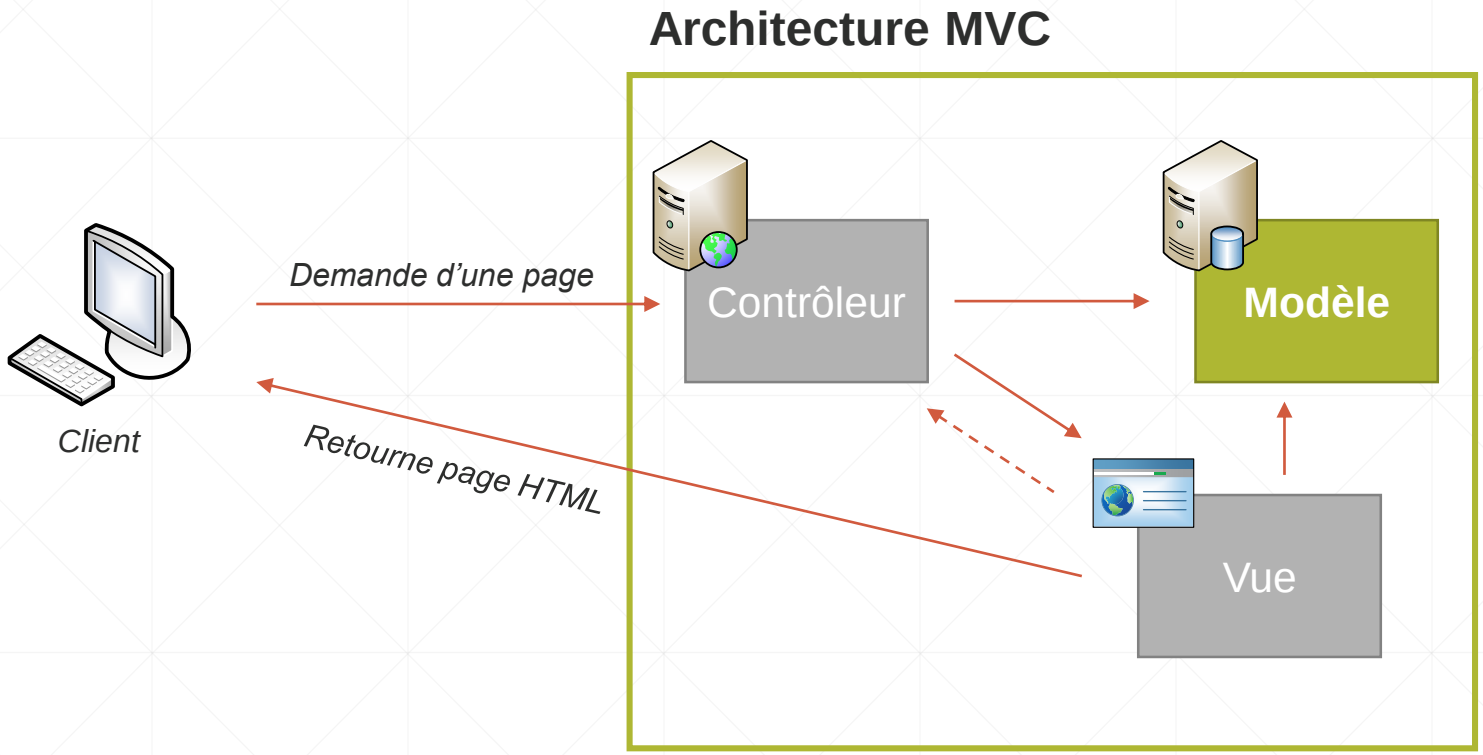
Qu’est-ce qu’un ORM ?

- **Définition :** *« ou **O**bject-**R**elational **M**apping c’est programme informatique permettant de créer une abstraction ou interface entre le programme applicatif bénéficiant de la persistance des données et la base de données utilisée. »*,
- **Objectif :** *rendre la manipulation des données plus aisée, moins sujette à des erreurs, efficace et optimisée.*
- Il y a deux *formes* ou deux *aspects* entre lesquels l’**ORM** va faire l’*intermédiaire* :
  - Partie Application sous forme de *base de données orientée objet,*
  - Partie **SGBD** sous forme de *base de données relationnelle.*
- Selon le langage utilisé, différent **ORM** peuvent être utilisés :
  - Exemples : **Python** *(SQLAlchemy)*, **Java** *(Hibernate)*, **C#** *(Entity Framework)*.

Le paradigme MVC

- **Définition :** *« c'est un motif d'architecture logicielle ou design pattern destiné à la conception d'interface graphique lancé en 1978. Il est souvent utilisé pour la conception de sites et applications Web. »*,
- **Objectif :**
  - *Permettre une séparation entres les données et le rendu graphique,*
  - *Meilleure organisation et maintenabilité du code.*
- Il est composé de trois élément principaux :
  1. Le **modèle** *(Model)* : *responsable de représentation et de l'accès aux données,*
  2. La **vue** *(View)* : *responsable du rendu graphique ou partie apparence,*
  3. Le **contrôleur** *(Controller)* : *permet d'ordonnancer les différentes actions.*

Aperçu du MVC



Plan

1. Introduction et Contexte
2. *Paradigme* **MVC**
3. Persistance de données et **ORM**
4. Les fondamentaux de **Microsoft Entity Framework**
   a. Qu'est-ce que **Microsoft Entity Framework** ?
   b. Mise en place de la persistance
   c. Mise en place de relations
   d. Les opérations **CRUD**
   e. Notion de données liées
5. Mise en application
6. Questions et discussion

Qu’est-ce que Microsoft Entity Framework ?

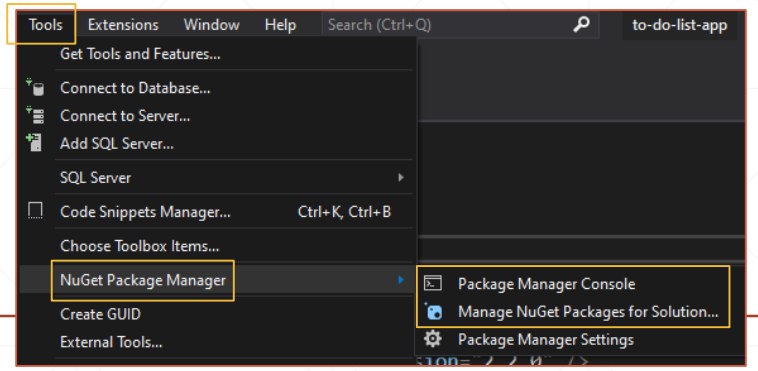
- **Définition :** *« C’est l’**ORM** destiné à **ADO.NET** proposé et maintenu par Microsoft. Il est écrit en **C#** et il est utilisé par la technologie **Microsoft .NET** pour l’aspect persistance des données. Il a été proposé en **2008** en Open Source. »*,
- Qu’est-ce qu’**ADO.NET** ? *Ou **A**ctiveX **D**ata **O**bject est bibliothèque logicielle de Microsoft fournissant une interface d’accès aux données sous Windows.*
- Microsoft **E**ntity **F**ramework (**EF**) **Core**[1] :
  - *C’est une version allégée et extensible d’Entity Framework proposé en **2016**,*
  - *Il est **multiplateforme** et tire profit de la force du langage **LINQ**.*
- Qu’est-ce que **LINQ** ? *Ou **L**anguage-**IN**tegrated **Q**uery est un composant du Framework .NET ajoutant la capacité d’interrogation sur les données. **EF Core** utilise **LINQ-to-Entity** ou **ELINQ**.*

[1] Lien : https://github.com/dotnet/efcore

Mise en place de la persistance – Étape 01

Afin de pouvoir configurer la *persistance de données*, il faut suivre les étapes :

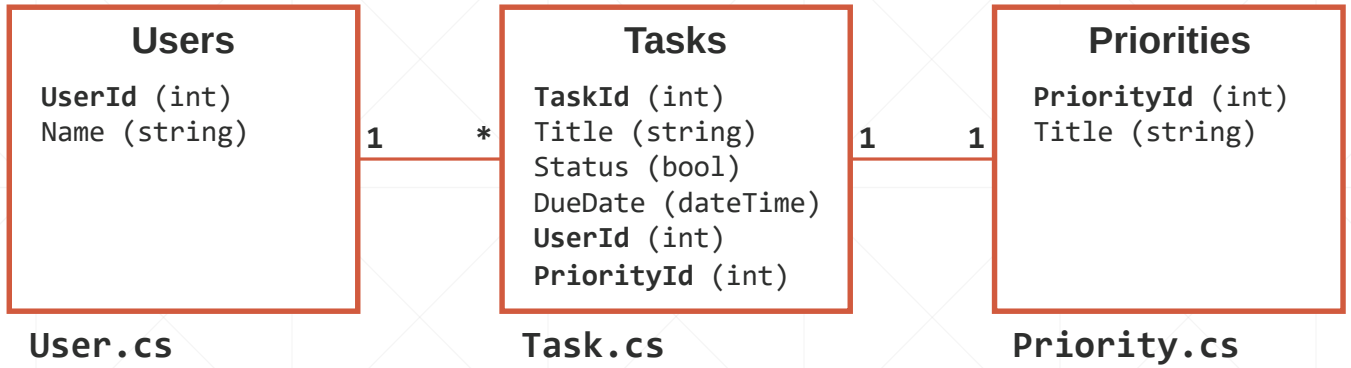
1. **Installation des paquets nécessaires :**
   - **EF Core** n'est pas installé par défaut sur **Microsoft Visual Studio**, Il faut passer soit par :
   - Console de **gestion de paquets NuGet** :
     - **Aller :** *`Tools > NuGet package Manager > package Manager Console`*
   - Interface graphique pour la **gestion de paquets NuGet** :
     - **Aller :** *`Tools > NuGet package Manager > Manage NuGet Packages for Solution...`*
   - *Il faut installer les paquets suivants :*
     - *`Install-Package` **`Microsoft.EntityFrameworkCore.SqlServer`***
     - *`Install-Package` **`Microsoft.EntityFrameworkCore.Tools`***
     - *`Install-Package` **`Microsoft.EntityFrameworkCore.Design`***



Mise en place de la persistance – Étape 02

2. Création des *entités* du modèle à utiliser :

- Tout au long de ce cours, on se basera sur le diagramme suivant *(Task Manager)* :



- Pour chacune des *tables* ou *entités*, il faut créer une *classe* à part.

Mise en place de la persistance – Étape 03

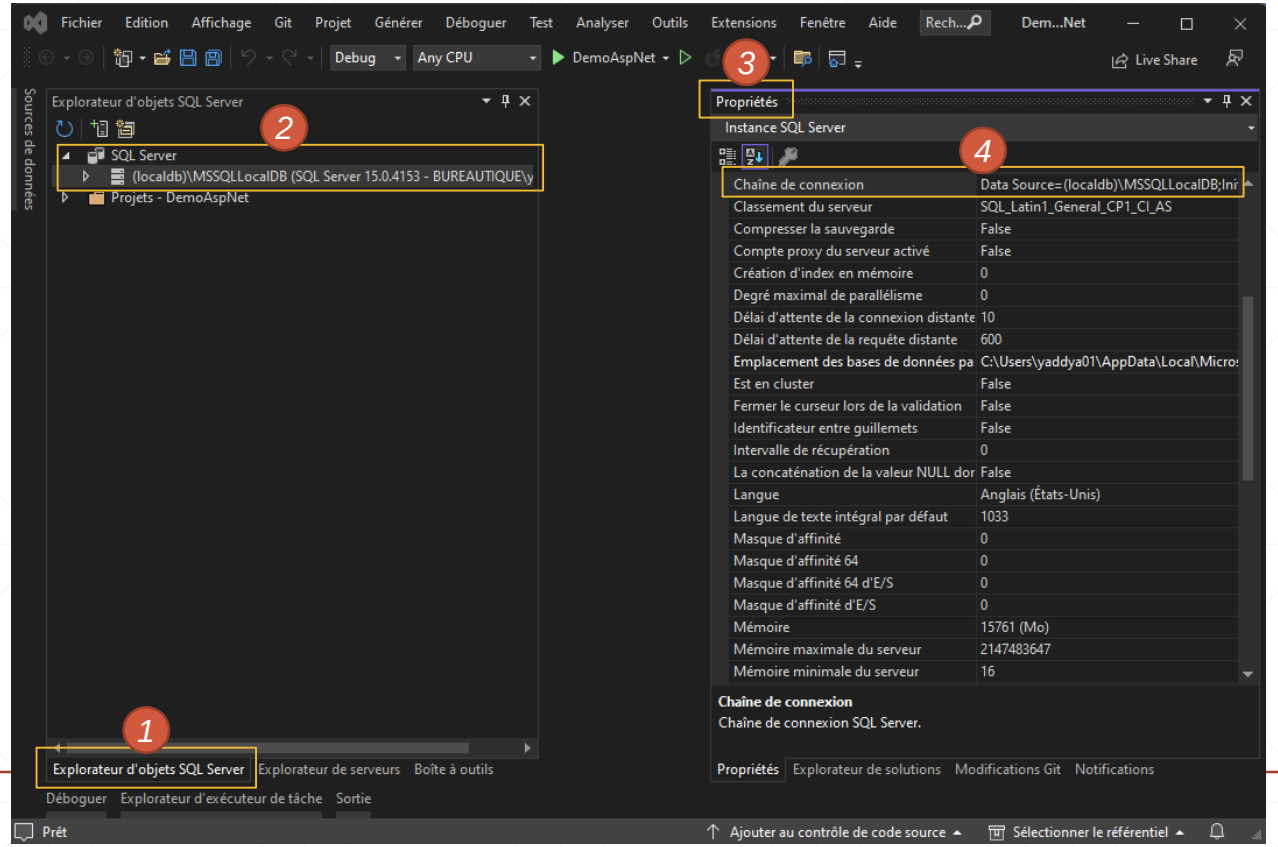
3. **Création du fichier de configuration :**
   - **Aller :** *Add > Class...*
   - Ajouter le fichier : **ApplicationName***Context.cs*
4. **Mettre les configurations nécessaires :**
   - Déclarer les différentes *tables* à créer,
   - Configurer la *connexion* avec la base de données,
   - Inclure les *initialisations* au besoin.

- Dans la dispositive qui suit le code à utiliser dans **ApplicationName***Context.cs* :

Mise en place de la persistance – Étape 03

Afin de récupérer la **chaîne de connexion** pour la base de données :

1. *Explorateur d'objet SQL Server*,
2. *SQL Server > Propriétés*,
3. *Chaîne de connexion*.



Mise en place de la persistance – Étape 03

```
class TaskManagerDbContext : DbContext
{
    public DbSet<Models.User> Users { get; set; }
    public DbSet<Models.Task> Tasks { get; set; }
    public DbSet<Models.Priority> Priorities { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder dbContextOptionsBuilder)
    {
        string connection_string = "Data Source=(localdb)\\MSSQLLocalDB;Initial
          Catalog=master;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServer
          Certificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False";
        string database_name = "TaskManagerDB";
        dbContextOptionsBuilder.UseSqlServer($"{connection_string};Database={database_name};");
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Models.Priority>().HasData(
            new Models.Priority() { PriorityId = 1, Title = "High" },
            new Models.Priority() { PriorityId = 2, Title = "Medium" },
            new Models.Priority() { PriorityId = 3, Title = "Low" }
        );
    }
}
```

Les tables

Chaîne de connexion à base de données

Nom de la base de données

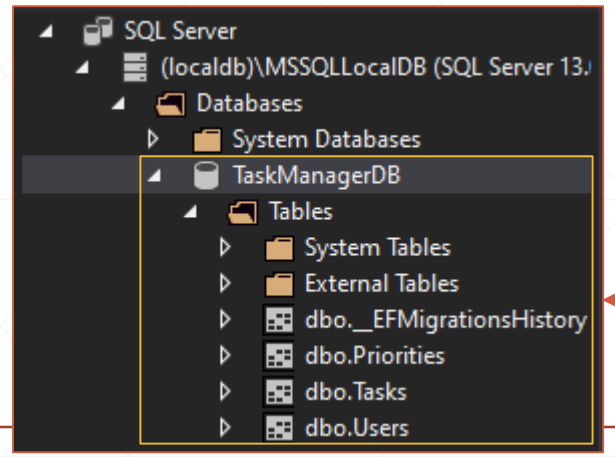
Initialisation de la table ***Priorities***

```
using Microsoft.EntityFrameworkCore;
```

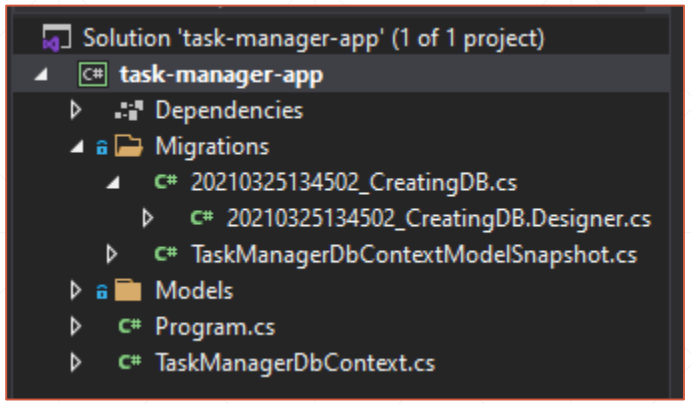
Dépendance

Mise en place de la persistance – Étape 04

5. Ensuite, il faut générer les fichiers de migration avec :
   - **Commande : `add-migration` *`given_name_to_the_migration`***
   - Cette commande est exécutée dans la **console de gestion de paquet NuGet**.
6. Afin d'appliquer les fichier de migration générer, il faut exécuter :
   - **Commande : `update-database -verbose`**



Création de la base de données avec les tables



Mise en place de relations

- Il y a principalement trois types de relations :
  - **Une-vers-Une** ou *One-to-One*,
  - **Une-vers-Plusieurs** ou *One-to-Many*,
  - Plusieurs-vers-Plusieurs ou *Many-to-Many*,
- On verra les deux premières, car c'est celles qu'on utilisera pour le reste du cours :
  - Les relations peuvent être établies au niveau du ***modèle***,
  - Une les modifications au niveau des classes effectuées, il faut :
    - Il faut générer les *fichier des migrations*,
    - Les *appliquer*.

Mise en place de relations – *One-to-One*

Ci-dessous le code utilisé pour lier la table **`Tasks`** à **`Priorities`** :

Clé étrangère

```
class Task
{
    public int TaskId { get; set; }
    public string Title { get; set; }
    public bool Status { get; set; }
    public DateTime DueDate { get; set; }

    public int PriorityId { get; set; }
    public Priority Priority { get; set; }
}
```

Task.cs

Clé primaire

```
class Priority
{
    public int PriorityId { get; set; }
    public string Title { get; set; }
}
```

Priority.cs

Mise en place de relations – *One-to-Many*

Ci-dessous le code utilisé pour lier la table **`Tasks`** à **`Users`** :

```cs
class User
{
    public User()
    {
        Tasks = new List<Task>();
    }

    public int UserId { get; set; }
    public string Name { get; set; }

    public ICollection<Task> Tasks { get; set; }
}
```

Instance de la collection

Clé primaire

`User.cs`

```cs
class Task
{
    public int TaskId { get; set; }
    public string Title { get; set; }
    public bool Status { get; set; }
    public DateTime DueDate { get; set; }

    public int UserId { get; set; }
    public User User { get; set; }
}
```

Clé étrangère

`Task.cs`

Les opérations CRUD – *Fondamentaux*

Il est possible de réaliser *quatre différentes opérations* de bases sur une base de données afin de manipuler des données à travers une *requête* :

- **Ajouter** d'une nouvelle ligne,
- **Lecture** des données,
- **Mise à jour** des champs,
- **Suppression** de lignes de données.

Les opérations CRUD – *Ajouter*

- Afin d'ajouter un nouvel utilisateur, on utilisera le code suivant :

Contexte →
Appliquer →

```
static void AddUser(Models.User user)
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();
    taskManagerDbContext.Users.Add(user);
    taskManagerDbContext.SaveChanges();
}
```

- Il est également possible d'ajouter une liste d'utilisateurs avec :

```
static void AddUsers(List<Models.User> users)
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();
    taskManagerDbContext.Users.AddRange(users);
    taskManagerDbContext.SaveChanges();
}
```

Les opérations CRUD – *Lecture*

- Afin de lire l'ensemble des utilisateurs dans la base de données :

```
static void ShowUsers()
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();
    List<Models.User> users = taskManagerDbContext.Users.ToList();
}
```

- Afin de lire un utilisateur en particulier en utilisation sa *clé primaire* :

```
static void ShowUser(int userId)
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();
    Models.User user = taskManagerDbContext.Users.Find(userId);
}
```

- Il est également possible d'effectuer un *filtrage* :

```
Models.User user = taskManagerDbContext.Users.Where(u => u.UserId == userId).First();
```

Méthode

Opération sur chaque utilisateur

Juste le premier

Les opérations CRUD – *Mise à jour **et** Suppression*

- Afin de mettre à jour un utilisateur en particulier, il faut utiliser :

```
static void UpdateUser(int userId, string name)
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();
    Models.User user = taskManagerDbContext.Users.Find(userId);
    user.Name = name;
    taskManagerDbContext.SaveChanges();
}
```

- Pour supprimer un utilisateur en particulier, il faut utiliser :

```
static void DeleteUser(int userId)
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();
    Models.User user = taskManagerDbContext.Users.Find(userId);
    taskManagerDbContext.Users.Remove(user);
    taskManagerDbContext.SaveChanges();
}
```

Notions de données liées dans ORM

- Afin d'ajouter une nouvelle tâche à un utilisateur, il faut utiliser :

```
static void AddTaskToUser(int userId, Models.Task task)
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();

    Models.User user = taskManagerDbContext.Users.Include(u => u.Tasks).Where(u => u.UserId == userId).First();

    user.Tasks.Add(task);
    taskManagerDbContext.SaveChanges();
}
```

- Avec **task** :

```
new Models.Task()
{
    Title = "Faire le travail pratique",
    Status = false,
    DueDate = new DateTime(2021, 3, 30),
    Priority = taskManagerDbContext.Priorities.Find(1)
};
```

Notions de données liées dans ORM

Si on veut afficher *toutes les informations des tâches* d'un *utilisateur* :

```
static void ShowTasksOfUser(int userId)
{
    TaskManagerDbContext taskManagerDbContext = new TaskManagerDbContext();
    Models.User user = taskManagerDbContext.Users.Include(u => u.Tasks)
                                                 .ThenInclude(t => t.Priority)
                                                 .Where(u => u.UserId == userId).First();

    foreach (Models.Task task in user.Tasks)
    {
        Console.Write($"{task.TaskId} - {task.Title} - {task.Status} - {task.DueDate.ToShortDateString()} - ");
        Console.WriteLine($"{{task.User.Name} - {task.Priority.Title}");
    }
}
```

Inclure toutes les informations des tâches

Inclure toutes les informations des priorités

Utilisation avec MySQL Server

- Par défaut, il est recommandée d'utiliser **SQL Server** qui est installé par défaut avec **Microsoft Visual Studio Community**,
- Cependant, s'il y a des problème lors de l'utilisation ou bien que l'on souhaite utiliser un serveur de gestion de base de données externe, on peut se tourner vers **MySQL Server**,
- Afin de pouvoir l'utiliser, il faudra installer les éléments suivants :
  - **MySQL Community Server :** https://dev.mysql.com/downloads/mysql/
  - **MySQL Workbench :** https://www.mysql.com/fr/products/workbench/
- Il y a aussi des alternative comme **MariaDB** ou bien **PostgreSQL**.

Utilisation avec MySQL Server

- Une fois installée, il faudra procéder :
  - La création d'une base de donnée,
  - Ainsi qu'un utilisateur ayant les privilèges suffisants.
- Au niveau de la solution Microsoft Visual Studio, il faudra ajouter le paquet suivant :
  - *Install-Package* ***Pomelo.EntityFrameworkCore.MySql***
  - Il supporte **MySQL** et **MariaDB**.
- Au niveau de la configuration, il faudra juste faire une modification au niveau :
  - Du fichier **ApplicationName***Context.cs*

Ancienne version
Nouvelle Version

```
string ConnectionString = "server=localhost;port=3306;database=<database>;user=<user>;password=<password>;";
```

  - Mettre la configuration suivante : `dbContextOptionsBuilder.UseMySql(ConnectionString);`

```
dbContextOptionsBuilder.UseMySql(ConnectionString, ServerVersion.AutoDetect(ConnectionString));
```

Mise en application

En vous basant sur l'exercice de la recherche de livres, il vous est demandé :

1. Configurer la persistance des données avec :
   - **Microsoft Entity Framework Core**,
   - **MySQL Community Server**.
2. Permettre à l'utilisateur :
   - Ajouter un nouveau livre,
   - Supprimer un livre existant.
3. De permettre la création de profils utilisateurs,
4. Chaque utilisateur aura à gérer ses propres livres.

Questions & Discussion

Bibliographie

1. Smith, J. P. (2018). *Entity Framework core in action*. Manning Publications.
2. **Sites Web :**
   - https://docs.microsoft.com/fr-fr/ef/
   - https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx