

# Cours ORM

SIO2 - Antoine MATHAT

# Sommaire

- Les ORM
- Utilisation d'EF6
- DAO

Qu'est ce qu'un ORM ?

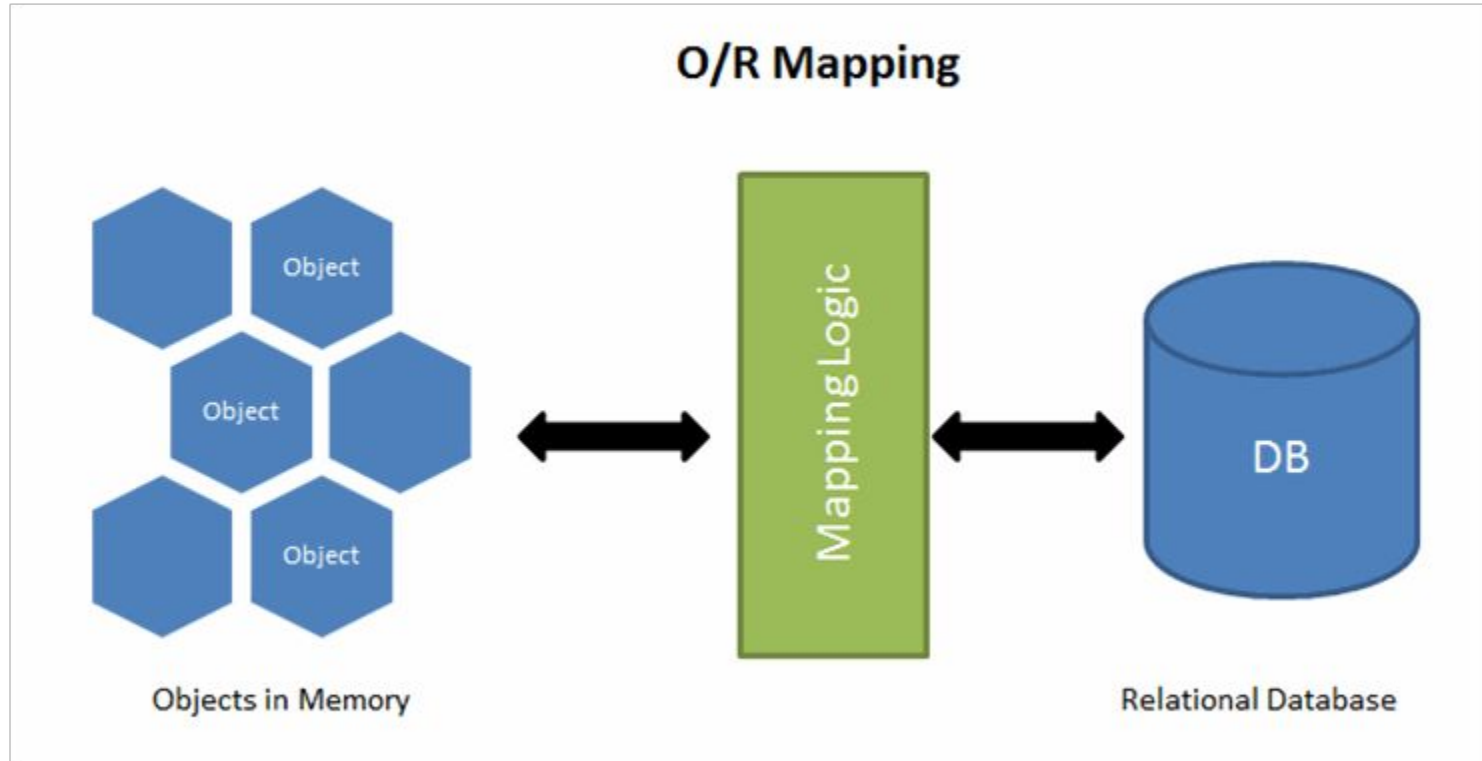
# Object Relation Mapping (ORM)

**ORM** est une technique qui permet de mapper un arbre de modèles en POO à une base de données relationnelle.

Un **ORM Framework** est une bibliothèque qui permet d'utiliser cette technique au sein d'un projet.

On obtient ainsi une **synchronicité** entre l'arbre de modèles et la base de données

# Object Relation Mapping (ORM)



# Object Relation Mapping (ORM)

- Un **modèle** est une classe qui stocke des données (Etudiant Voiture, Bien, Pret ...)
- Chaque **modèle** devient une **table**
- Les **attributs** du modèle deviennent les **champs** de la table
- **Tout est automatique**
  - La création des tables
  - Les requêtes basiques CRUD
  - Les associations entre les tables

# Object Relation Mapping (ORM)

L'utilisation d'un ORM Framework apporte aux développeurs :

- Un énorme **GAIN DE TEMPS**
- La production d'une base de données de **QUALITÉ**

# Entity Framework 6 (EF6)



L'ORM Framework de Microsoft



# Utilisation d'EF6

# Configuration d'EF6

1. Installation du paquet Entity Framework
2. Ajouter le Connecteur du SGBDR souhaité
3. Créer votre contexte et vos modèles
4. Démarrer votre base de données
5. Faites une première migration pour générer la base

Voir TP Initialisation ORM pour plus de détails.

# Les modèles (entité)

```
2 références
public class Prestataire
{
    0 références
    public int PrestataireId { get; set; }
    0 références
    public string RaisonSociale { get; set; }
    0 références
    public string Nom { get; set; }
    0 références
    public string Prenom { get; set; }
    0 références
    public string Telephone { get; set; }
    0 références
    public string Adresse { get; set; }
    0 références
    public List<Intervention> Interventions { get; set; }
}
```

Créer une liste de propriété correspondant aux attributs du modèle.

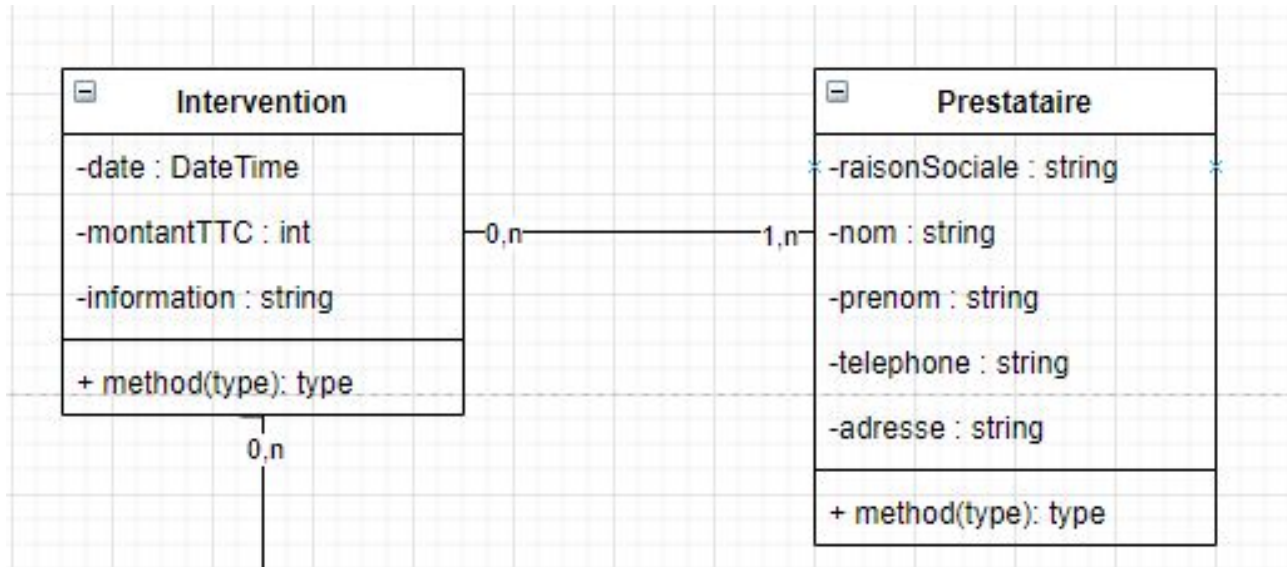
Attention : Il est important de préfixer l'id par le nom de la classe (pour les fk de l'héritage)

# Les modèles (entité)

```
9 références
public abstract class Bien
{
    0 références
    public int BienId { get; set; }
    2 références
    public string Nom { get; set; }
    1 référence
    public int Valeur { get; set; }
    1 référence
    public string Adresse { get; set; }
    1 référence
    public int Surface { get; set; }
    1 référence
    public List<Bail> Bails { get; set; }
    0 références
    public List<Intervention> Interventions { get; set; }
    0 références
    public Pret? Pret { get; set; }
}
```

Pour qu'une clé étrangère soit facultative, il faut autoriser la valeur nulle sur votre objet en utilisant "?"

# Association multiple entre deux modèles



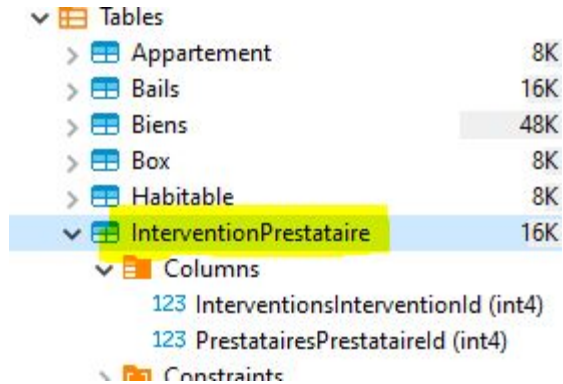
# Génération

```
3 références
public class Intervention
{
    0 références
    public int InterventionId { get; set; }
    0 références
    public DateTime Date { get; set; }
    0 références
    public double MontantHT { get; set; }
    0 références
    public string Informations { get; set; }
    0 références
    public List<Prestataire> Prestataires { get; set; }
    0 références
    public Bien Bien { get; set; }
}
```

```
2 références
public class Prestataire
{
    0 références
    public int PrestataireId { get; set; }
    0 références
    public string RaisonSociale { get; set; }
    0 références
    public string Nom { get; set; }
    0 références
    public string Prenom { get; set; }
    0 références
    public string Telephone { get; set; }
    0 références
    public string Adresse { get; set; }
    0 références
    public List<Intervention> Interventions { get; set; }
}
```

Pour une relation multiple bi-directionnelle, la table PrestataireIntervention est générée automatiquement

# Génération



▼	Tables	
>	Appartement	8K
>	Bails	16K
>	Biens	48K
>	Box	8K
>	Habitable	8K
▼	InterventionPrestataire	16K
▼	Columns	
	123 InterventionsInterventionId (int4)	
	123 PrestatairesPrestataireId (int4)	
▼	Constraints	

La table PrestataireIntervention.

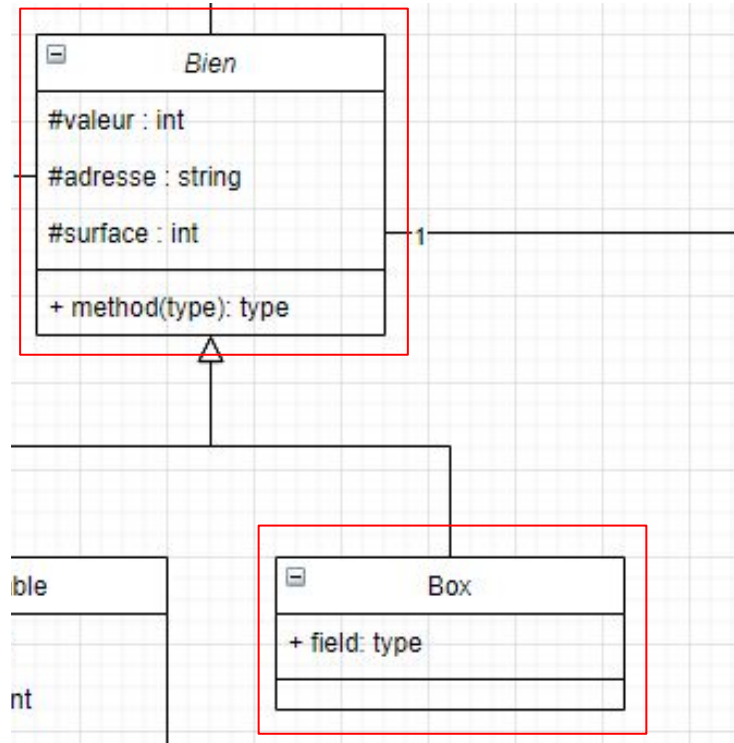
Les deux clés étrangères sont ajoutées automatiquement vers les tables concernées.

# Héritage

1. Bien nommer les Id des modèles
2. Ne pas linker les classes filles au contexte (pas de DbSet)
3. Surcharger la méthode OnCreating du contexte pour mapper les classes filles



# Héritage pour Box et Bien



# Héritage

## Classe mère

```
9 références
public abstract class Bien
{
    0 références
    public int BienId { get; set; }
    2 références
    public string Nom { get; set; }
    1 référence
    public int Valeur { get; set; }
    1 référence
    public string Adresse { get; set; }
    1 référence
    public int Surface { get; set; }
    1 référence
    public List<Bail> Bails { get; set; }
    0 références
    public List<Intervention> Interventions { get; set; }
    0 références
    public Pret Pret { get; set; }
```

## Classe fille

```
5 références
public class Box : Bien
{
    0 références
    public int BoxId { get; set; }
```

Bien nommer les Id pour la génération des clés étrangères !

# Héritage

```
15 références
public class GestImmoContext : DbContext
{
    0 références
    public DbSet<Bail> Bails { get; set; }
    1 référence
    public DbSet<Bien> Biens { get; set; }
    0 références
    public DbSet<Locataire> Locataires { get; set; }
    0 références
    public DbSet<Pret> Prets { get; set; }
    0 références
    public DbSet<Intervention> Interventions { get; set; }
    0 références
    public DbSet<Prestataire> Prestataires { get; set; }

    0 références
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        => optionsBuilder.UseNpgsql("Host=localhost;Database=gestimmo;Username=postgres;Password=root");
}
```

On ne map que la classe mère avec les DbSet, pas les filles.

# Héritage

```
0 références
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    => optionsBuilder.UseNpgsql("Host=localhost;Database=gestimmo;Username=postgres;Password=root");

0 références
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Habitable>().ToTable("Habitable");
    modelBuilder.Entity<Maison>().ToTable("Maison");
    modelBuilder.Entity<Box>().ToTable("Box");
    modelBuilder.Entity<Appartement>().ToTable("Appartement");
}
```

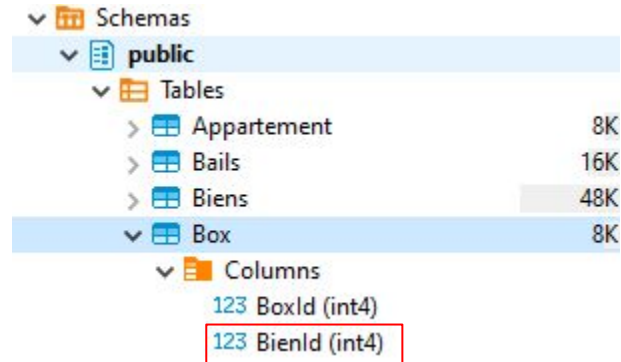
Toujours dans le context. On surcharge la méthode OnModelCreating pour demander la création des classes enfants.

# Héritage

## Classe Box

```
5 références
public class Box : Bien
{
    0 références
    public int BoxId { get; set; }
}
```

## Table générée Box



The screenshot shows a database schema tool interface. The 'Schemas' tree is expanded to 'public', and the 'Tables' tree is expanded to 'Box'. The 'Box' table is selected, and its columns are listed below. The 'BoxId' column is highlighted with a red box.

123	BoxId (int4)
123	BienId (int4)

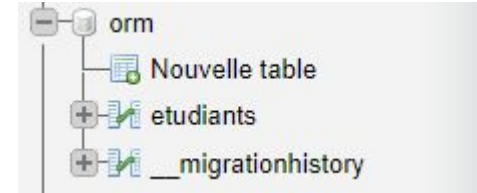
On peut constater que la référence est créée automatiquement. Nous n'avons pas eu besoin de créer l'attribut BienId dans la classe Box

# Utilisation d'EF6

Créer un contexte et lancer l'application pour générer la base de données.

```
EcoleContext ecoleContext = new EcoleContext();
```

Création du contexte



Base générée

# Data Access Object (DAO)

# Ajout d'une entité en base de données

```
EcoleContext ecoleContext = new EcoleContext();  
Etudiant e = new Etudiant { FirstMidName = "Carson", LastName = "Alexander", EnrollmentDate = DateTime.Parse("2005-09-01") };  
ecoleContext.Etudiants.Add(e);  
ecoleContext.SaveChanges();
```



# DAOBien

```
internal class DAOBien
{
    1 référence
    public List<Bien> getBienAvecPret()
    {
        List<Bien> biens;

        GestImmoContext context = GestImmoContext.GetInstance();
        var rows = from b in context.Biens
                    where b.Pret != null
                    select b;

        biens = rows.ToList();

        return biens;
    }
}
```

Les DAO permettent de préparer des requêtes plus compliquées, exemple de récupération des biens avec un prêt.