Cours .Net
Entity Framework V1.0

Lemettre Arnaud
Arnaud.lemettre@gmail.com

EPITA
ÉCOLE D'INGÉNIEURS EN INFORMATIQUE

Microsoft® .NET

Sommaire

- Introduction
- Principe théorique
- Mise en pratique
- Utilisation avancée
- Questions

Introduction

Qu'est ce qu'entity Framework ?

- C'est un ORM (object-relational mapping)

Introduction

Un ORM ?

technique de programmation informatique qui crée l'illusion d'une base de données orientée objet à partir d'une base de données relationnelle en définissant des correspondances.

Exemple :

Java : Hibernate
.Net : Nhibernate, Entity Framework

Pseudo ORM : Linq To SQL

Introduction

- Pourquoi utiliser EF ?
  - Eviter d'écrire du SQL dans le code
  - Permettre un contrôle à la compilation
  - Fortement typé
  - Abstraction de la base de données (SQL Server supporté par Microsoft, MySQL, Oracle par les autres éditeurs.)

Introduction

	Abstraction	Requêtes	Résultats
ADO.NET 2.0	Faible : Schéma relationnel	Chaînes de caractères	DataReaders
ADO.NET Entity Framework	Haut niveau: schéma conceptuel	LINQ	Fortement typés

PRINCIPES THÉORIQUES

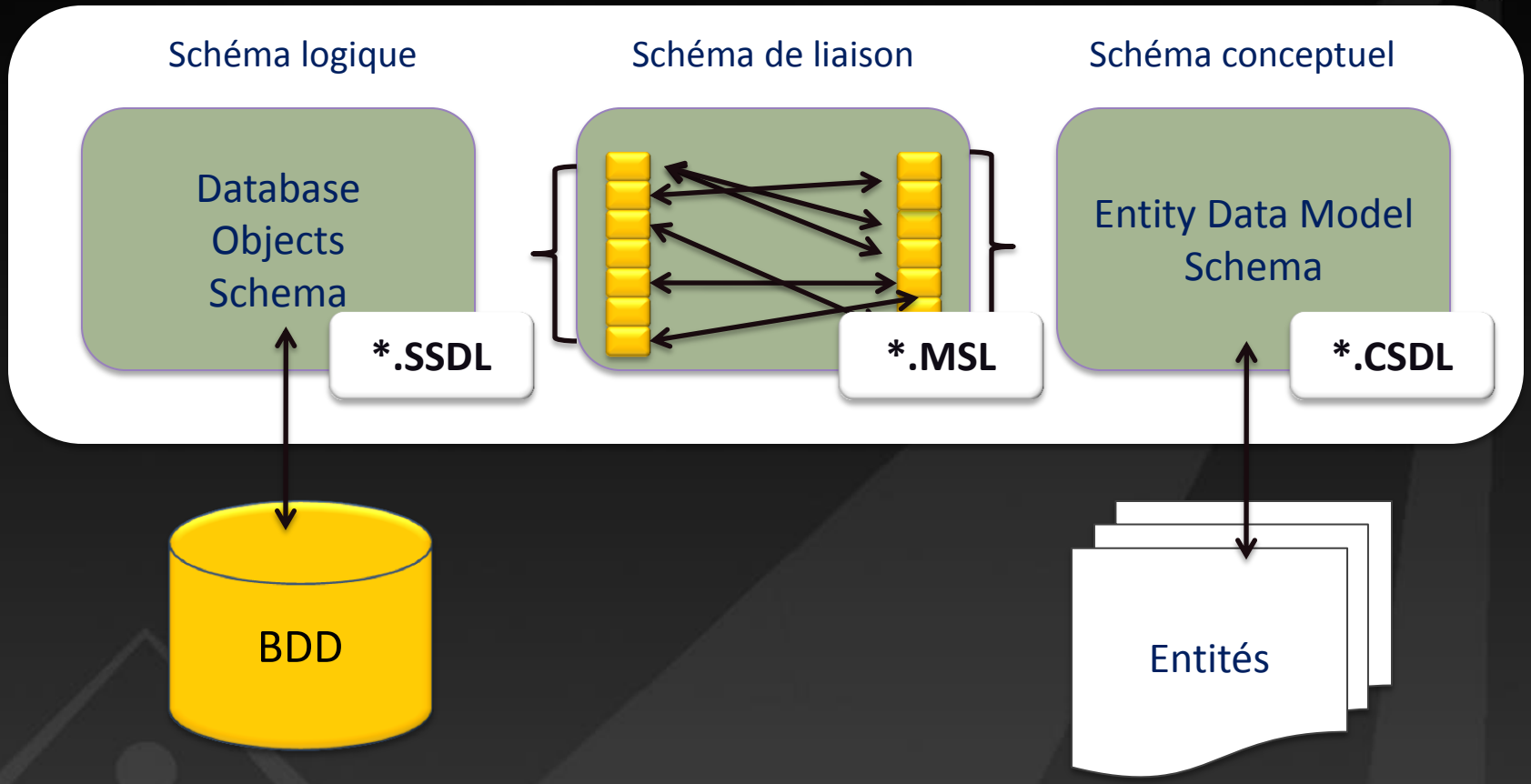
Principes théoriques

- Indépendance du schéma conceptuel (entités ou objets) du schéma logique de la base de données
- Fournit une couche d'abstraction permettant d'accéder aux données via un model (Entity Data Model)

Principes théoriques

- Un model c'est :
  - Un schéma conceptuel : définition des entités et des fonctions utilisées. Ces éléments sont définis dans un fichier XML appelé CSDL (Conceptual Schema Definition Language).
  - Un schéma logique : définition des tables, vues et procédures stockées déclarées dans la base de données. Toutes ces informations sont regroupées dans un fichier XML appelé SSDL (Store Schema Definition Language).
  - Un Schéma de liaison : Il définit les liaisons entre le schéma conceptuel et le schéma logique. Il associe entre autres les entités déclarées dans le fichier CSDL aux tables définies dans le fichier SSDL. Ce mapping est inscrit dans le fichier XML MSL (Mapping Specification Language).

PRINCIPES THÉORIQUES



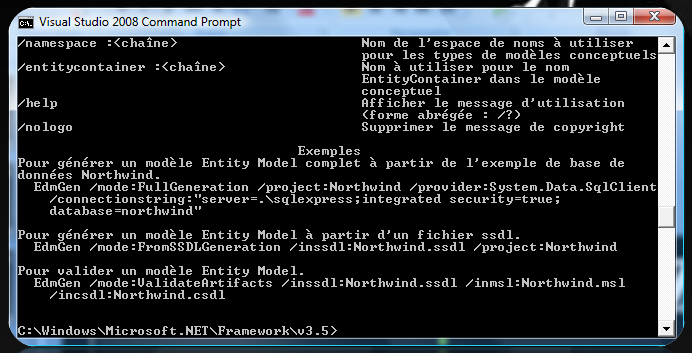
Principes théoriques

- Les avantages de ce schéma :
  - Modification partie logique
  - Modification partie conception

Indépendantes l'une de l'autre

  - Juste besoin de mettre à jour le fichier de liaisons

Principes théoriques



- Comment faire un model ?
  - 2 solutions :
    - EdmGen.exe => outil en ligne de commande
    - Assistant de visual studio => outil graphique

L'assistant de VS est moins complet que la ligne de commande, mais offre un gain de temps pour commencer, à terme l'outil sera capable de faire de même que l'outil en ligne de commande.

documentation

Principes théoriques

- Une entité :
  - Ensemble de propriétés et de relations d'associations avec les autres entités
  - Elles sont présentes dans le schéma conceptuel
  - Générée grâce à EdmGen.exe
  - Elle représente les classes qui seront manipulées dans le code. Ces classes peuvent être étendues avec des Partial

Principes théoriques

- L'accès aux données :
  - Linq To Entities
  - Entity SQL
  - Générateur de requêtes

Principes théoriques

- Linq To Entities :
  - Vérification des types à la compilation
  - Auto complétion
  - Proche de la syntaxe du Linq To Sql
  - Une requête en linq est ensuite traduite en Esql

Principes théoriques

- Entity SQL :
  - Requête sous forme de chaîne de caractères, donc pas de vérification
  - Offre la possibilité de construire les requêtes à la volée
  - Enrichie le TSQL, en rajoutant le support du model dans les requêtes
  - Permet l'utilisation de oftype

Principes théoriques

- Générateur de requêtes :
  - Mélange de Linq et de Esql

documentation

Principes théoriques

- Comment se fait l'accès aux données ?
  - Par l'intermédiaire du contexte, c'est une classe générée automatiquement par EdmGen.exe.
    On y trouve toutes les entités pour y effectuer les requêtes. Et l'ensemble des méthodes CRUD.

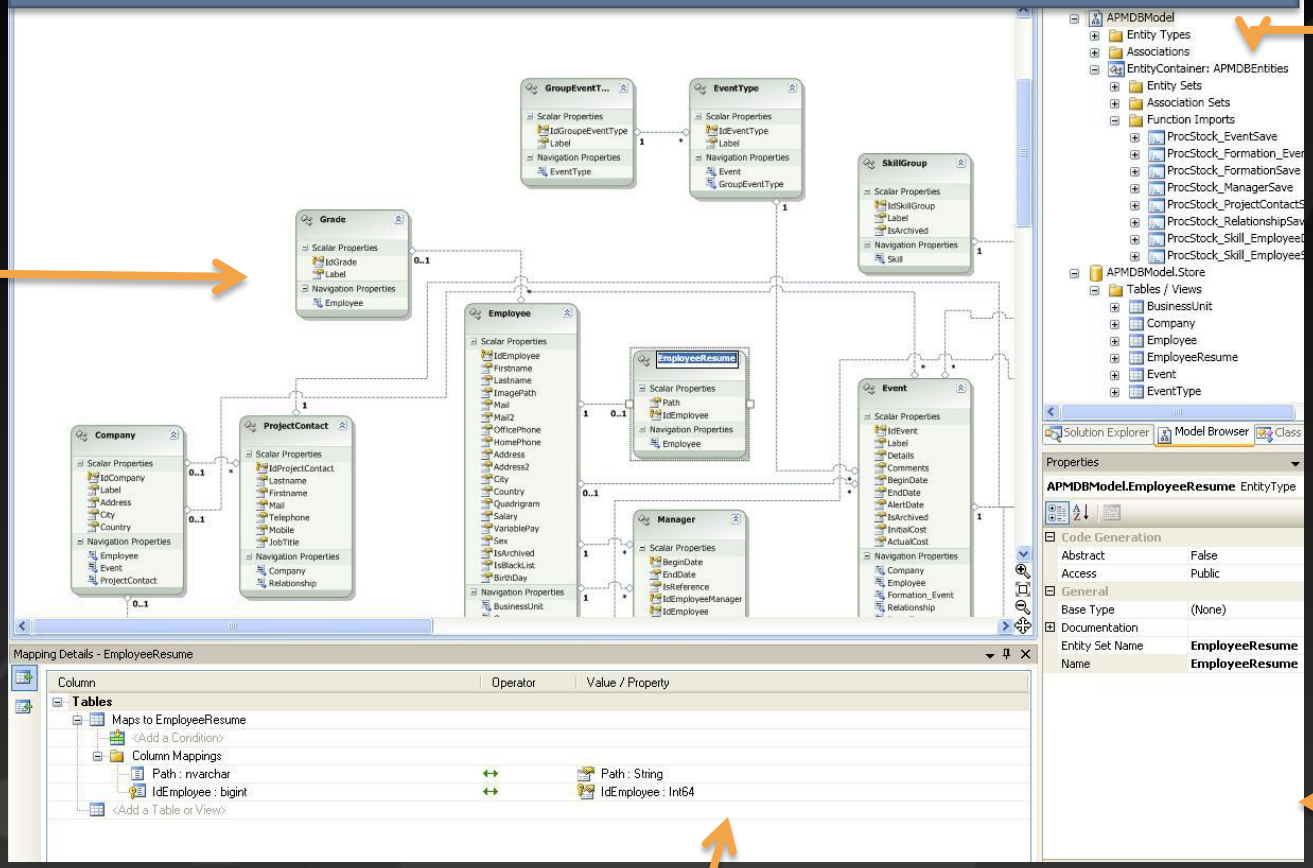
    De plus, le service de tracking de modifications est inclus dans cet objet, ainsi que la gestion de l'accès concurrentiel. C'est à partir de ce contexte que nous allons effectuer les différents requêtages.

Mise en pratique

INTERFACE D'EF

Tous les éléments du schéma conceptuel et logique du modèle

Vue générale représentant les relations et les entités

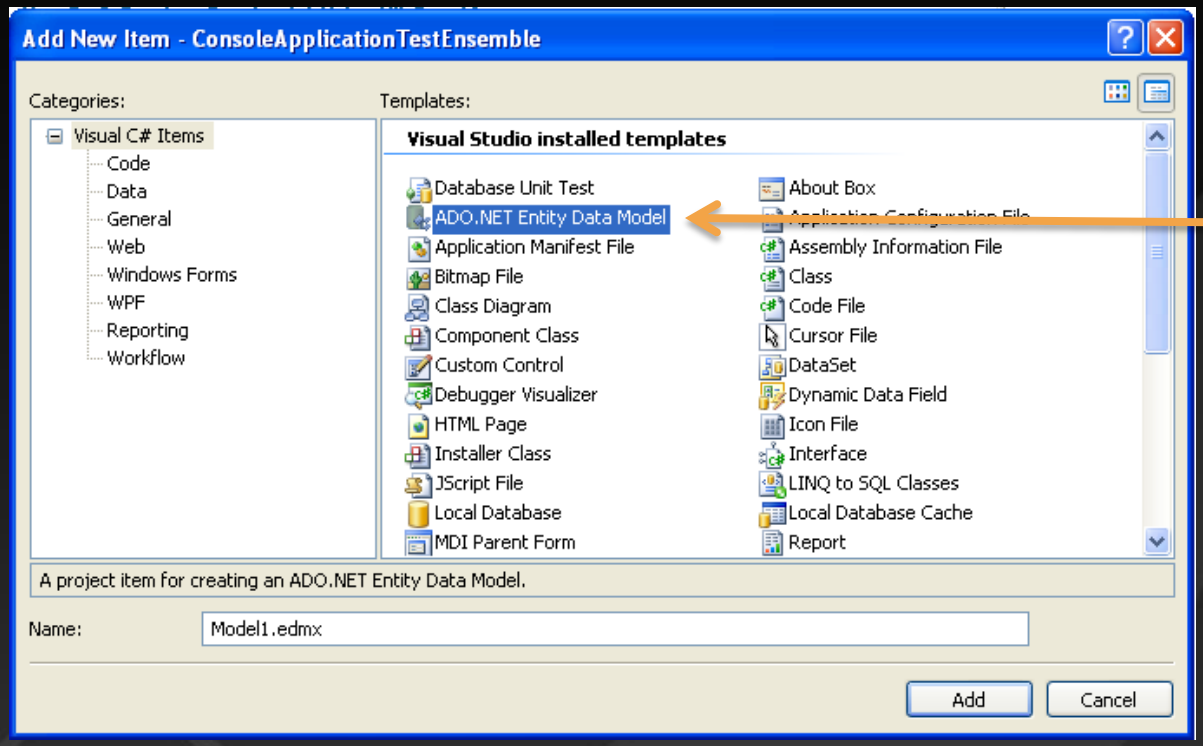


Fenêtre des propriétés des objets. Elle permet, par exemple, de définir l'abstraction de certaines classes

Permet d'avoir les détails du mapping, de rajouter des conditions, les procédures stockées …

Création d'un model

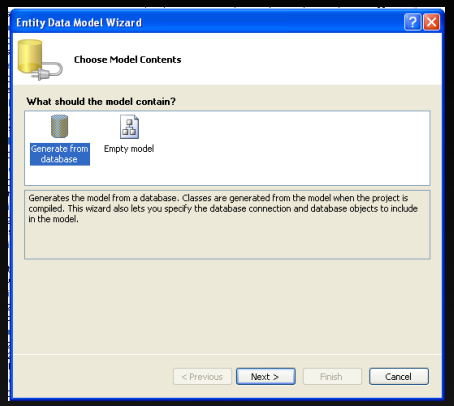
Sur un projet -> add -> New Item



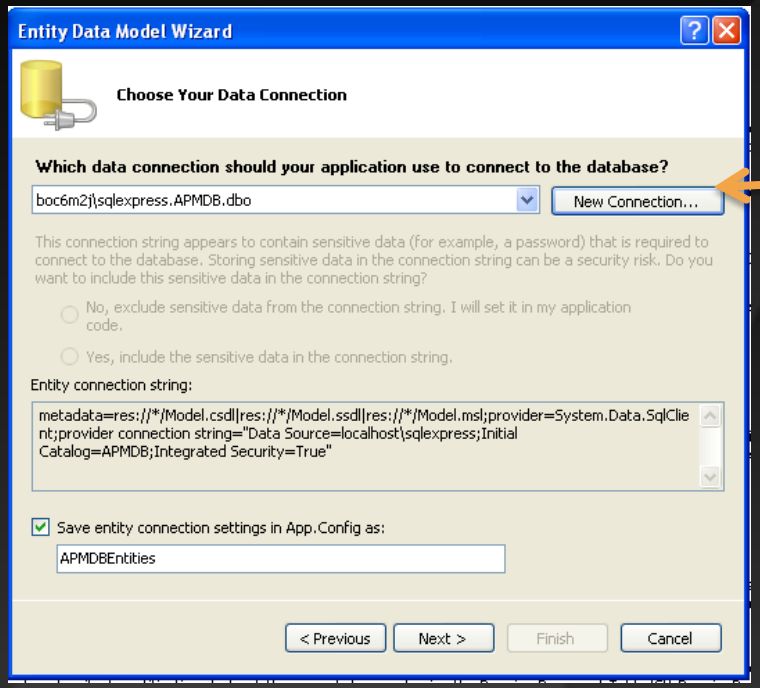
Puis il suffit de sélectionner :

ADO.NET Entity Data Model

Création d'un model

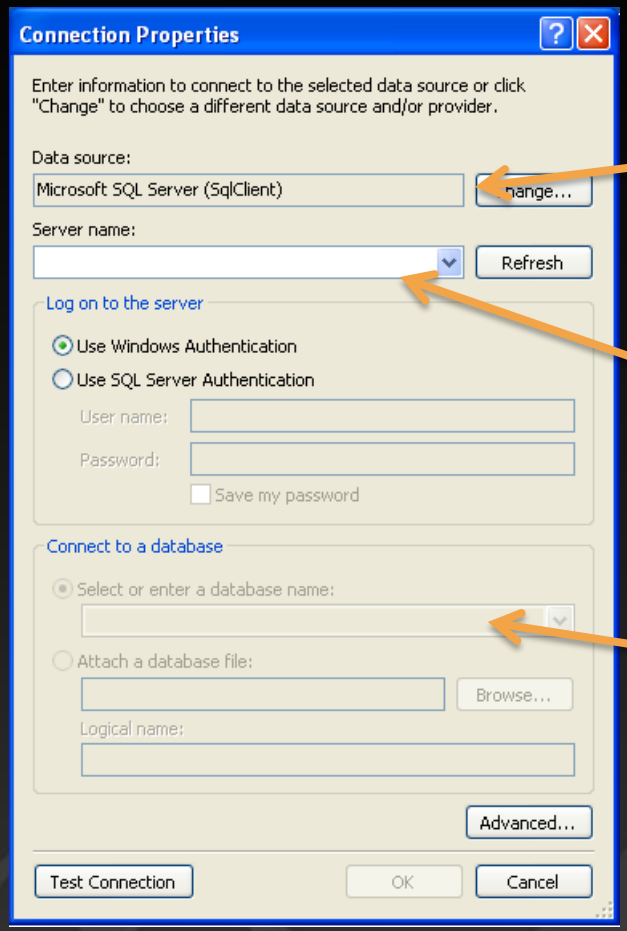


Sélection de la base de données. Si la chaîne n'est pas déjà présente il faudra donc la créer



Nouvelle connexion

Création d'un model

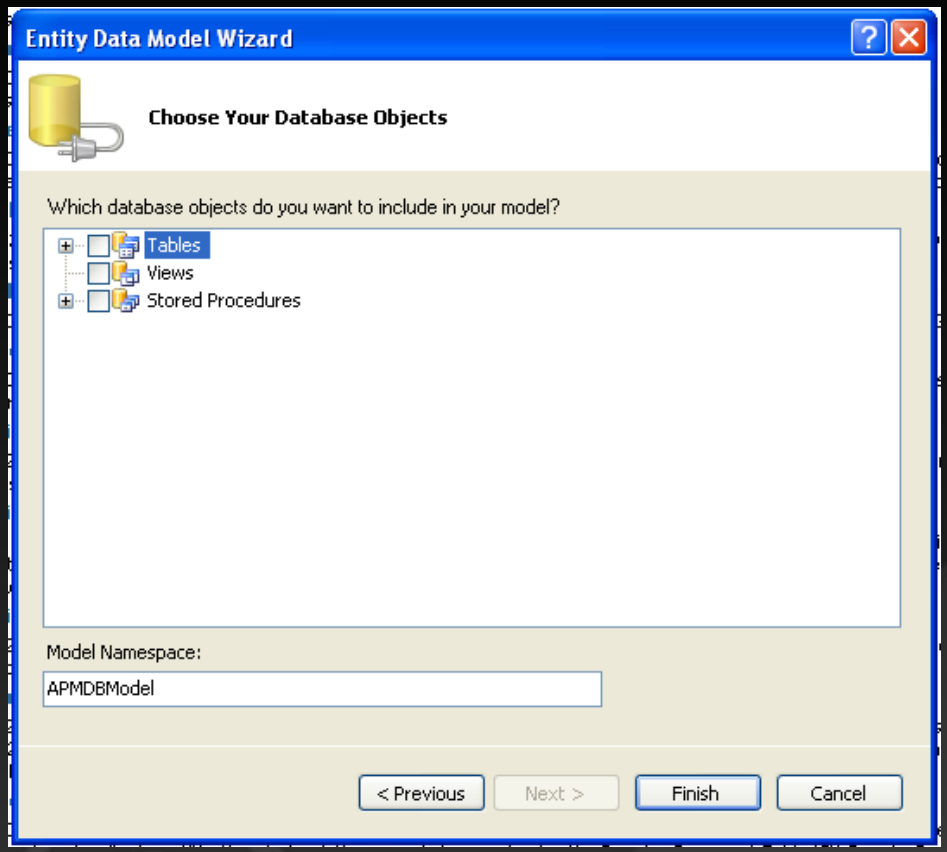


Adapter le connecteur en fonction de la base (Ms SQL, MySQL)

Nom du serveur, si c'est du Ms SQL en général on obtient ceci : Localhost\sqlexpress

Puis sélectionner la bonne base de données

Création d'un model



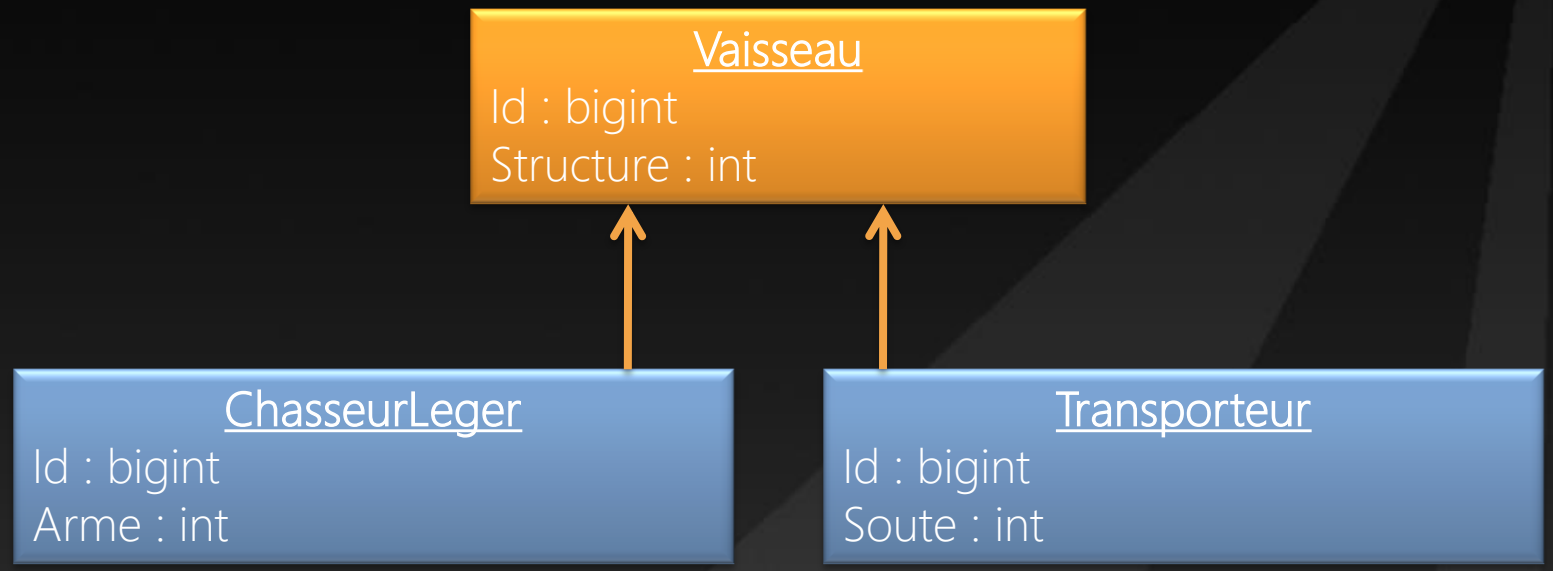
Permet de sélectionner ce que l'on veut mettre dans le model.

Quand on clique sur le bouton finish, EdmGen.exe génère le model et l'ajoute au projet.

Création d'un model

Il faut bien réussir son modèle du premier coup. C'est-à-dire, il ne faut pas supprimer une table car on peut pas la remettre.

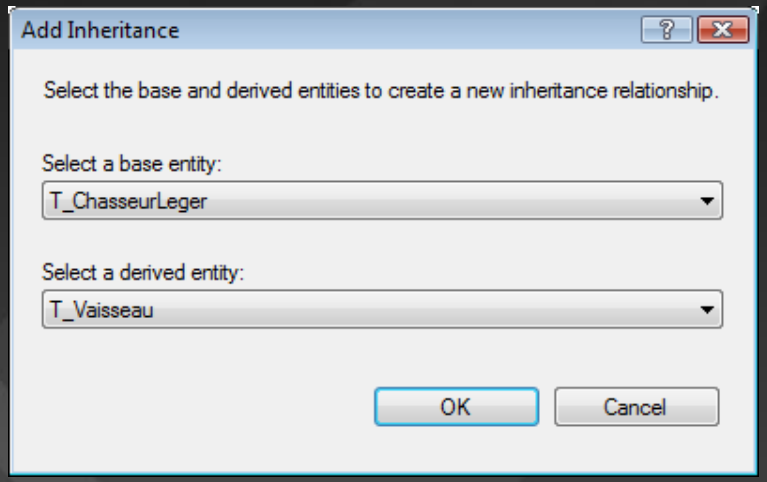
Relation d'héritage



Relation D'héritage

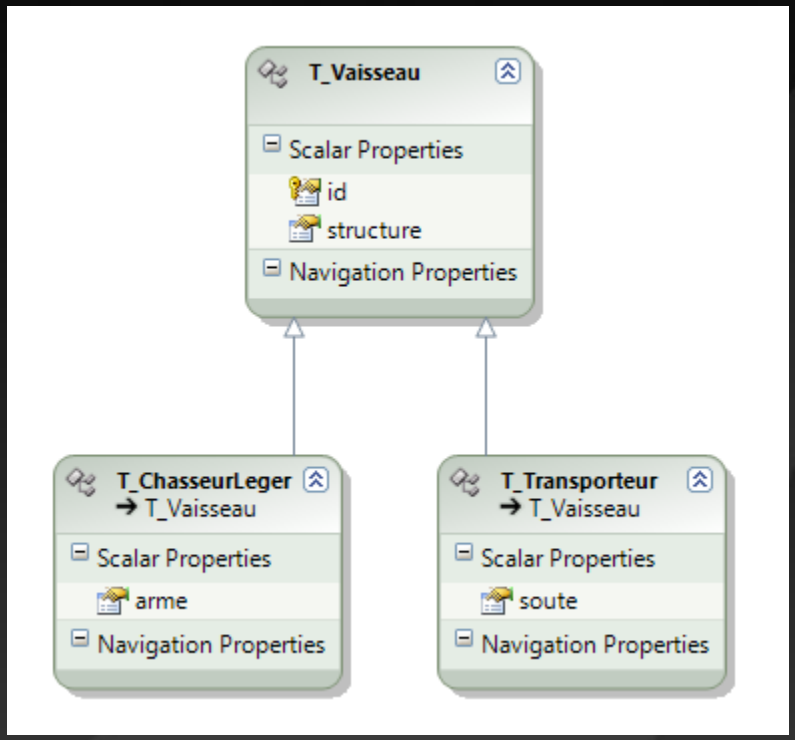
Pour réaliser cette notion d'héritage sur la partie objet :

- Supprimer toutes les clés d'identités des classes filles ici ChasseurLeger et Transporteur
- Puis il faudra indiquer le lien d'héritage pour cela 2 solutions :
  - Soit sur la classe fille, on fait un clic droit puis properties et on sélectionne base type à Vaisseau
  - Soit clic droit sur la table puis add -> Inheritance
  - Base type le nom de la classe fille, derived type : vaisseau



Relation d'héritage

On obtient donc ceci :



Relation D'héritage

Utilisation avec C#:

Permet de faire un select, en effet les tables de dérivation n'apparaissent pas directement pour faire une requête

```
using (ogamelikeEntities bdd = new ogamelikeEntities())
{
    var res = from chasseur in bdd.T_Vaisseau.OfType<T_ChasseurLeger>()
                  select chasseur;
}
```

Relation D'héritage

Utilisation avec C#:

Cependant pour créer une entrée, il y a bien l'objet dérivé. On complète l'objet puis il ne faut pas oublier de sauver les modifications.

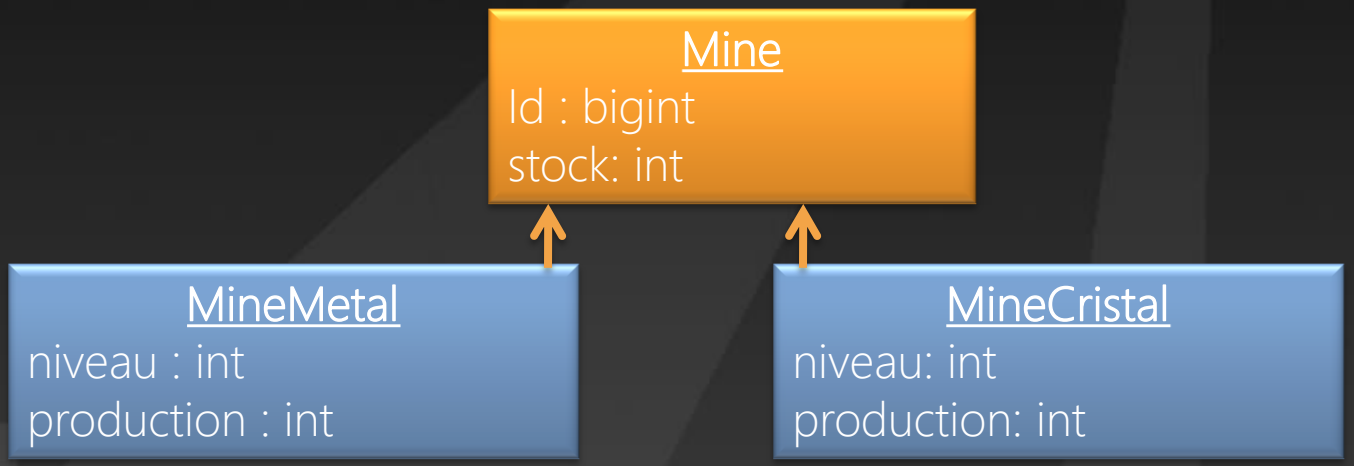
```
using (ogamelikeEntities bdd = new ogamelikeEntities())
{
    T_ChasseurLeger X350 = new T_ChasseurLeger()
    {
        structure = 500,
        arme = 10
    };

    bdd.AddToT_Vaisseau(X350);
    bdd.SaveChanges();
}
```

UNE TABLE, PLUSIEURS ENTITÉS

Mine
Id
Type
Stock
Production
Niveau

Le but est d'avoir plusieurs objets selon le type de mine, par exemple un objet :
-Mine de métal
-Mine de cristal

Et obtenir une représentation :



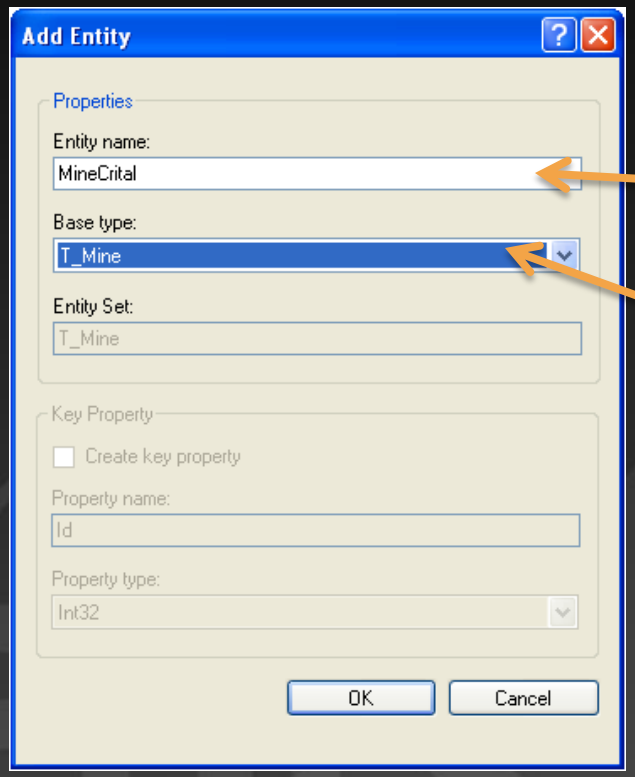
Une table, plusieurs entités

Pour effectuer cette réalisation, la première chose est de mettre en abstract la classe mine

- Properties sur l'entité puis abstract = true

Ensuite , il faut créer les nouvelles entités par exemple MineCristal, MineMetal

- - Pour cela clic droit sur le model add -> entity



Nom de la nouvelle entité

Classe abstraite

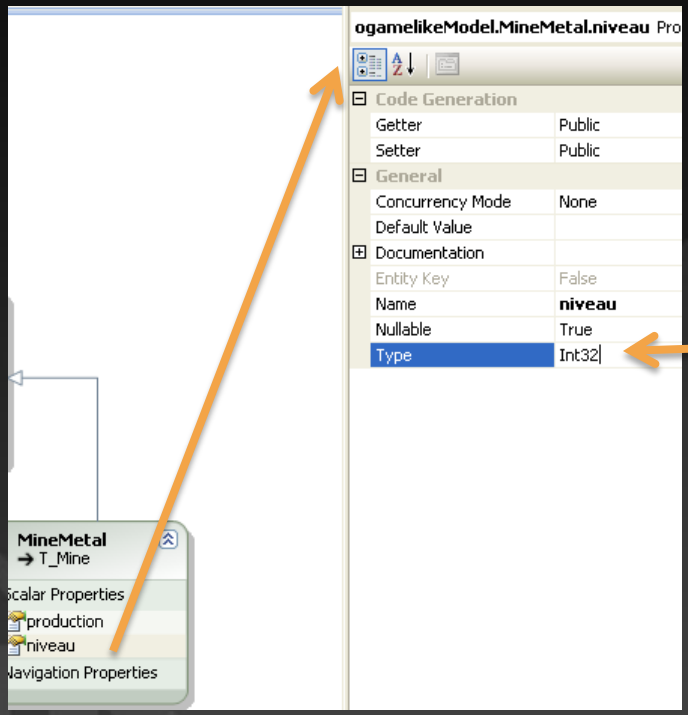
Une table, plusieurs entités

- Maintenant il faut supprimer les propriétés de la classe abstraite que l'on veut mettre dans les nouvelles entités. Mais également la propriété qui nous servira pour faire la différence entre les 2 types, à savoir la propriété Type.

Si la propriété, de séparation n'est pas supprimée cela provoquera des erreurs lors de l'exécution. Penser bien à vérifier dans la liste d'erreur que votre model est bon.

Une table, plusieurs entités

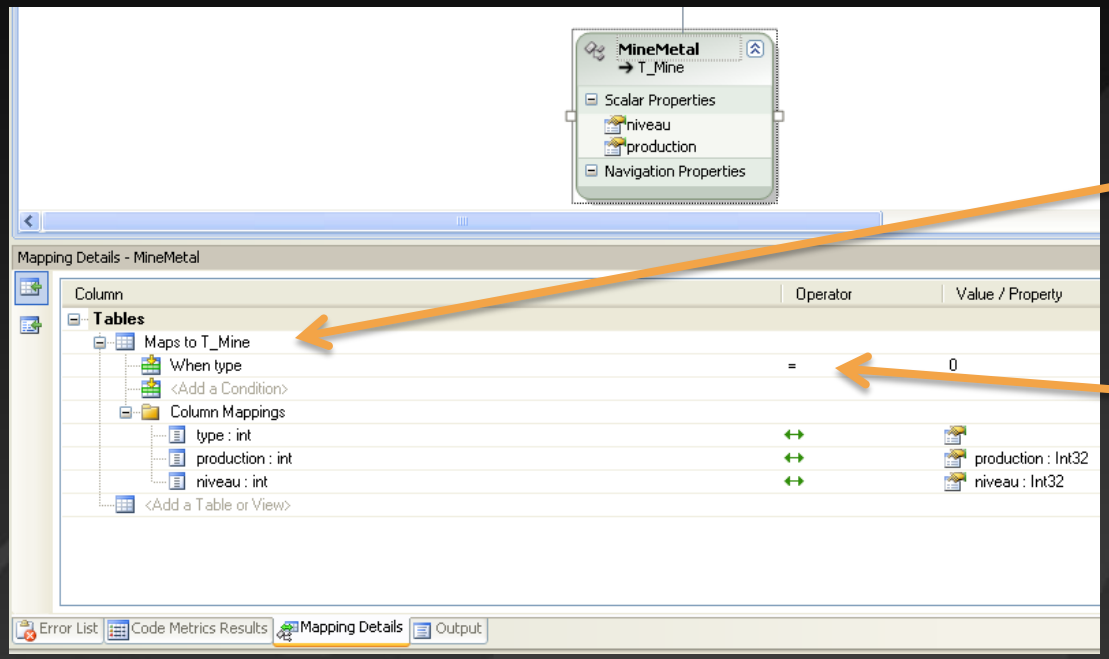
- Il faut ajouter les propriétés aux 2 nouvelles entités que l'on vient de créer, donc clic droit sur l'entité puis add -> property scalar
- A ce niveau vous pouvez la nommer. Attention, il faut ensuite aller dans les propriétés de cette property pour sélectionner le bon type !



Type de la propriété

Une table, plusieurs entités

- Il faut maintenant compléter le mapping, pour cela il faut sélectionner la bonne table et au besoin mapper les propriétés avec les champs de la table



Sélection de la table

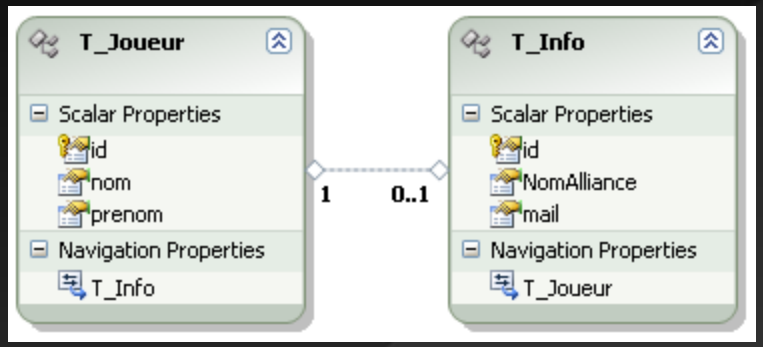
Ajout de la contrainte pour choisir la bonne table

Une table, plusieurs entités

- Les requêtes s'effectuent de la même manière

Une entité, plusieurs Tables

- Pour des raisons de simplicité, on peut vouloir toutes les informations sur une seule entité, alors qu'en base le fait d'avoir 2 tables est tout à fait justifié.
- Ce cas est uniquement possible si nous avons une relation de type 0..1->1

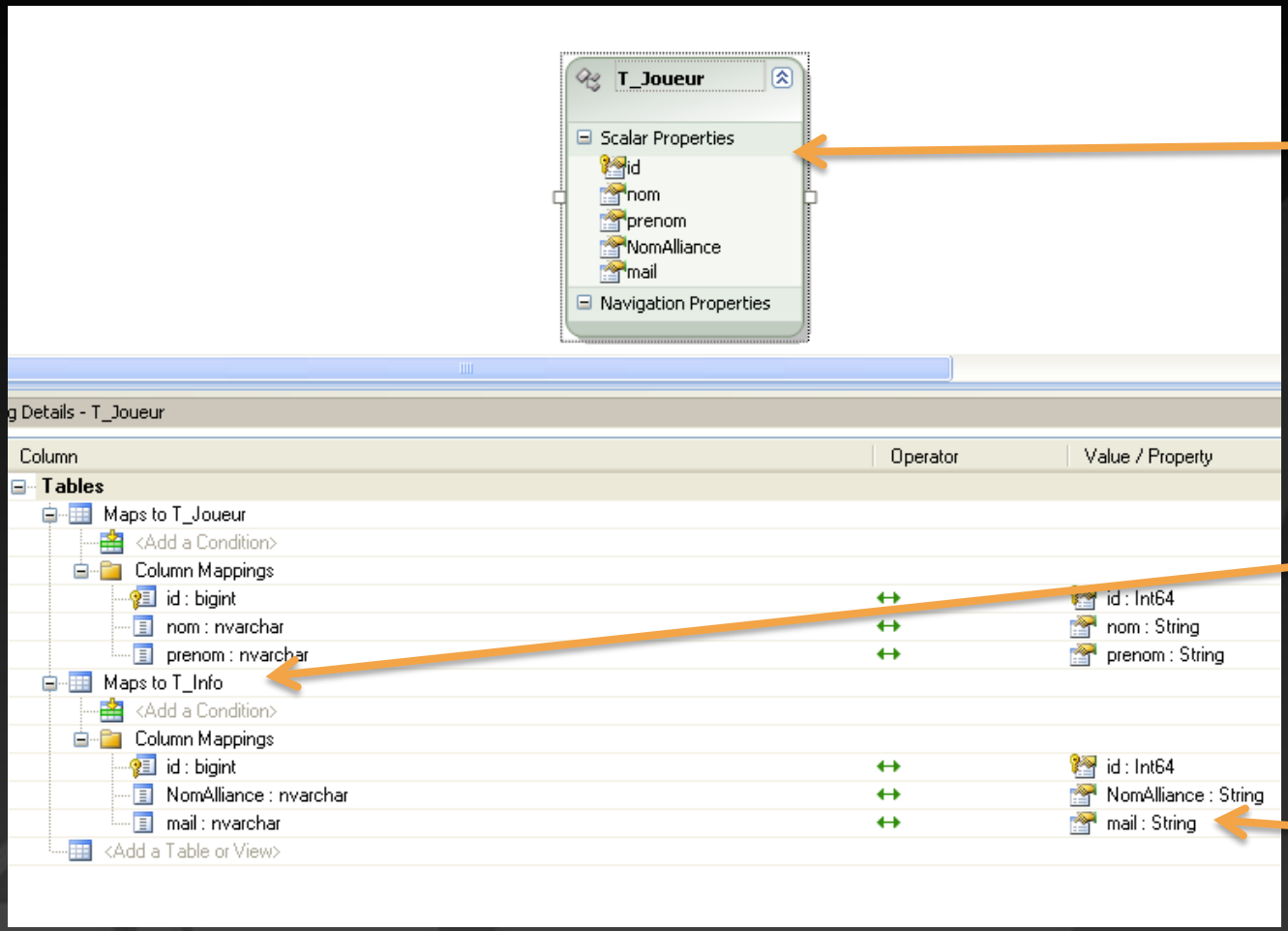


- Nous allons fusionner ces deux entités

Une entité, plusieurs Tables

- Pour cela copier d'abord les propriétés de l'entité T_Info pour les mettre dans T_Joueur.
- Puis supprimer l'entité T_Info
- Il faut maintenant rajouter le mapping sur l'entité T_Joueur. Pour cela passer sur le mapping détaillé puis cliquer sur <Add a table or View> et sélectionner T_Info. Automatiquement il va mapper les propriétés non utilisées et qui correspondent au bon type. Sur une grosse entité, il se peut que vous deviez le faire à la main pour que cela corresponde parfaitement à vos besoins.

Une entité, plusieurs Tables



L'entité joueur fusionnée

Le rajout du mapping sur T_info

Les propriétés de l'entité

Une entité, plusieurs Tables

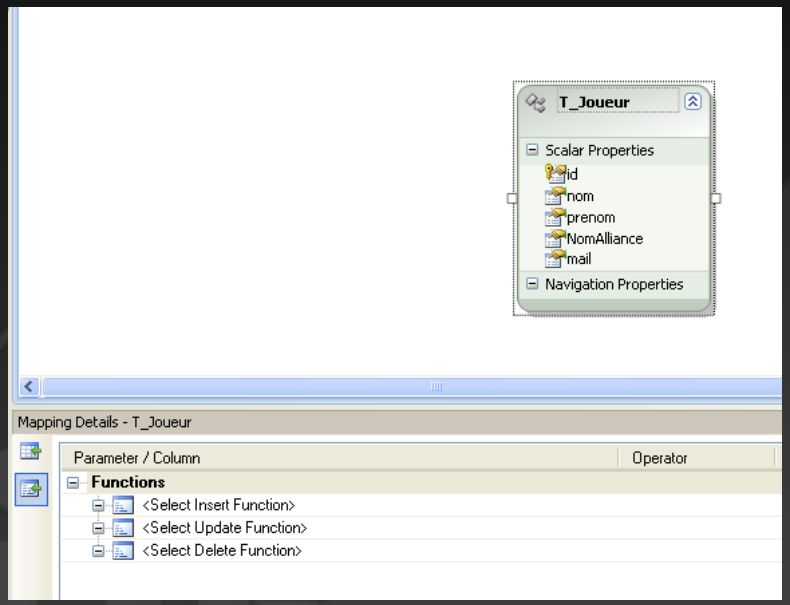
- Au niveau du code, la table T_info a disparu et vous ne pouvez pas requêter dessus.
- Pour remplir cette table, dès qu'une information concernant la table est remplie dans l'entité T_Joueur, alors cela modifie les entrées de la table en base T_Info.

Procédures Stockées

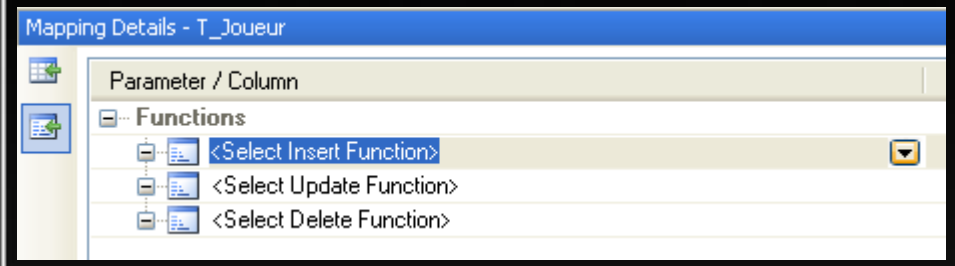
- Entity Framework est également capable de « gérer les procédures stockées ». Cependant dans certaines conditions, il faut faire attention.

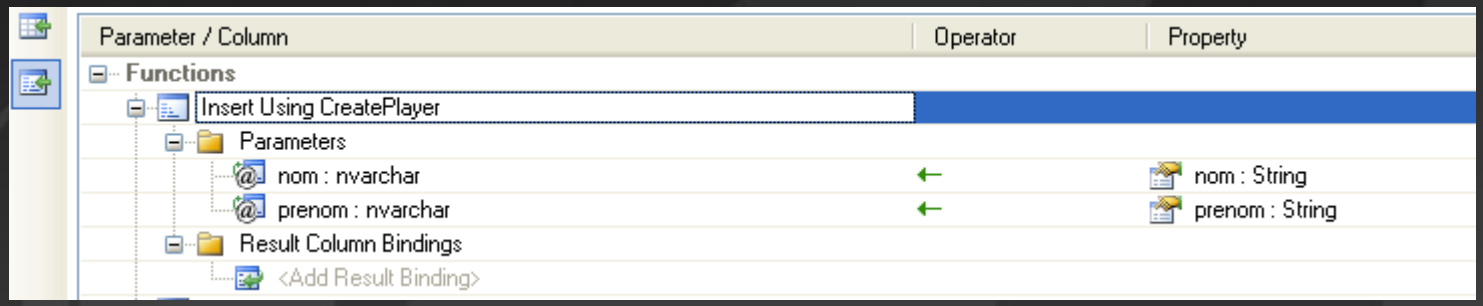
Procédures Stockées

- Les CUD de l'entity framework peuvent être overridés. Cependant si on décide de les remplacer, il faudra en faire de même pour toutes les autres.



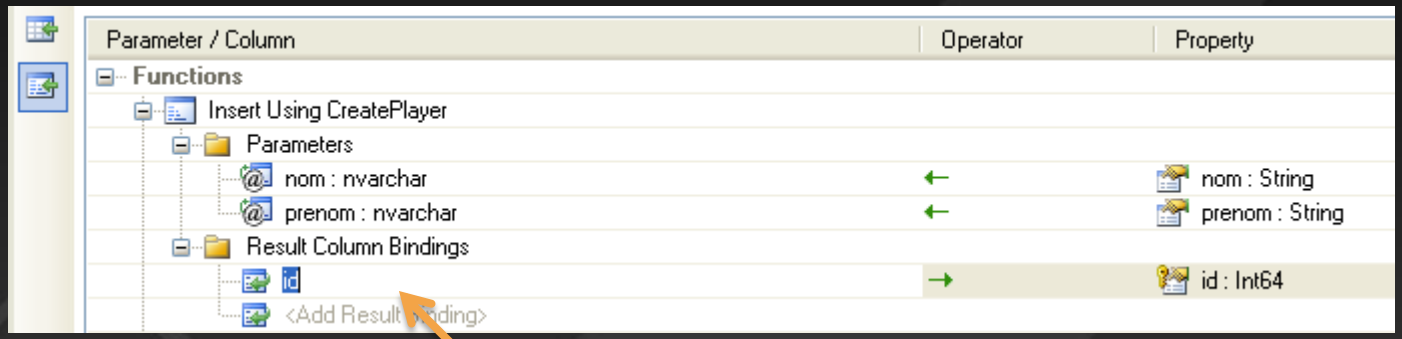
PROCÉDURES STOCKÉES





Procédures Stockées

- Mais ce n'est pas fini ! Car pour une procédure d'insertion c'est bien de pouvoir récupérer l'id qui a été généré !



Il faut taper id dans la colonne de binding et le designe associe directement avec l'id de l'entité

Procédures Stockées

Au niveau du T SQL cela se traduit par :

```
CREATE PROCEDURE [dbo].[CreatePlayer]
    -- Add the parameters for the stored procedure here
     @nom nvarchar(50),
   @prenom nvarchar(50)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
INSERT INTO [dbo].[T_Joueur]
           (
            [nom]

           ,[prenom]   )
     VALUES
            (@nom ,
             @prenom   )

select id
 from [dbo].[T_Joueur]
 where id  = scope_identity()
END
GO
```

Procédures stockées

Ajout d'une entité :

```
using (Entities bdd =new Entities())
{
    T_Joueur player = new T_Joueur()
    {
        mail = "player@epita.fr",
        nom = "lemettre",
        NomAlliance = "CardShark Influence",
        prenom = "Arnaud"
    };
    bdd.AddToT_Joueur(player);
    bdd.SaveChanges();
}
```

Mise à jour d'une entité :

```
using (Entities bdd = new Entities())
{

    T_Joueur player =
    bdd.T_Joueur.Where(x => x.id ==
    1).FirstOrDefault();
    if (player != null)
    {
        player.mail =
    "lemett_a@epita.fr";
    }
    bdd.SaveChanges();
}
```

Suppression d'une entité :

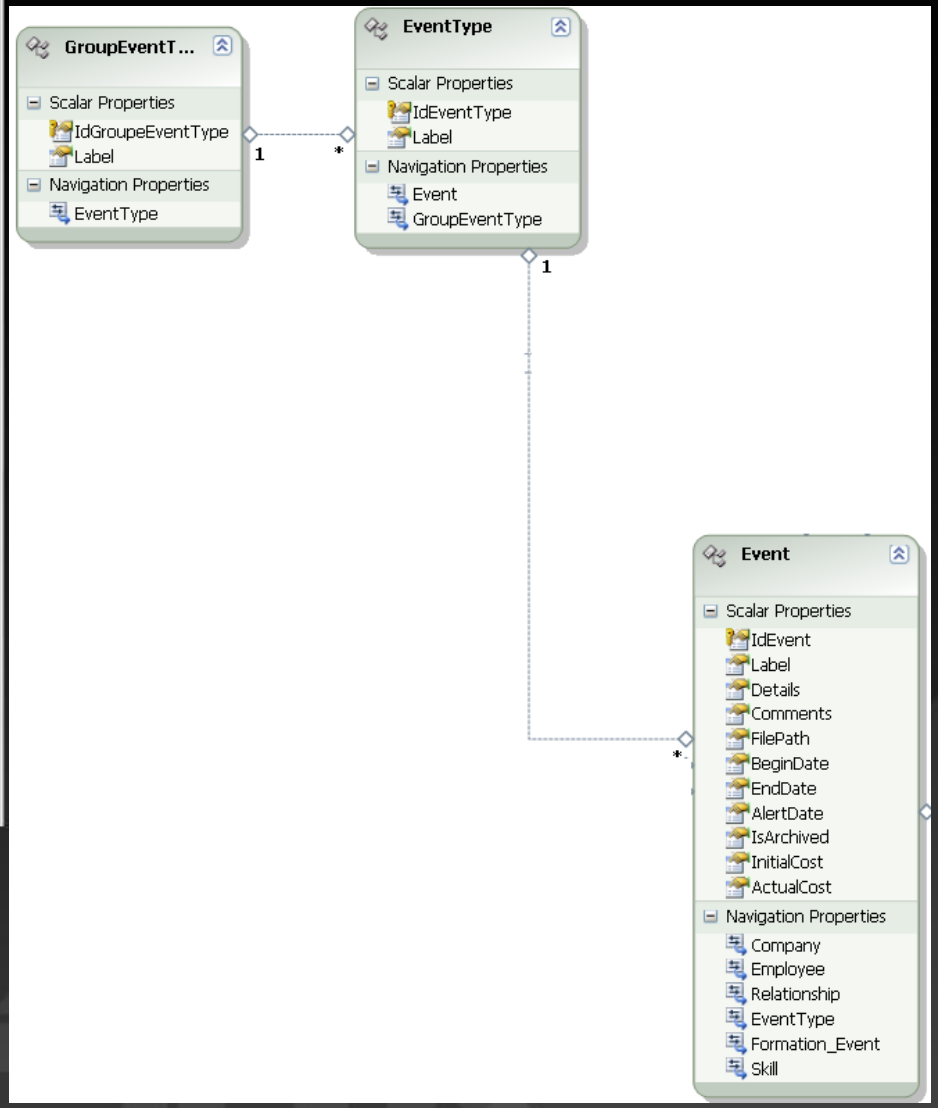
```
using (Entities bdd = new Entities())
{
    bdd.DeleteObject(bdd.T_Joueur.Where(x => x.id == 1).FirstOrDefault());
    bdd.SaveChanges();
}
```


Procédures Stockées

- Cependant, le fait d'utiliser ces fonctions de mises à jours peuvent poser de gros problèmes lors de schéma de base un peu particulier :

  exemple :

Procédures Stockées



Si on insère un Event en lui fixant un EventType alors l'entity framework ne va pas comprendre qu'il faut juste Fixer la valeur de l'id et non pas créer un nouvel EventType

Le moyen de contourner ce problème est de créer sa propre fonction d'insertion et une procédure stockée à part.

L'autre moyen mais qui ne fonctionne pas tout le temps est de récupérer l'objet avant et de l'affecter dans l'objet Event. Cependant lors d'une mise à jour cela ne fonctionnera pas.

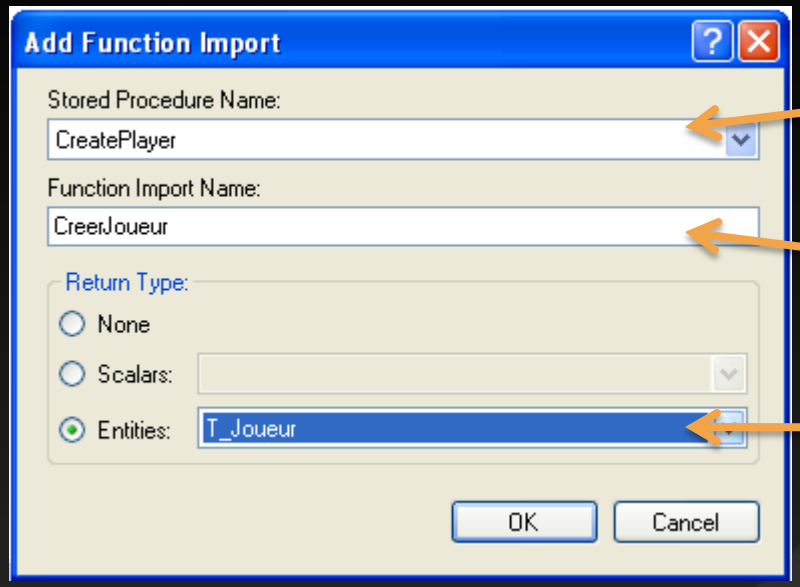
Procédures Stockées

- Bien entendu, les procédures stockées peuvent être d'une autre nature que des CRUD, « l'entity framework peut également le gérer ». Pour les utiliser, il faut importer les fonctions. Puis nommer la fonction pour l'utiliser et enfin spécifier le type de retour

Attention cependant seules les procédures qui retournent une entité fonctionnent. Si vous choisissez de ne rien retourner / ou autre chose, vous ne pourrez pas l'utiliser dans Entity Framework

Procédures Stockées

- Pour cela clic droit sur le model -> Add -> Function import …

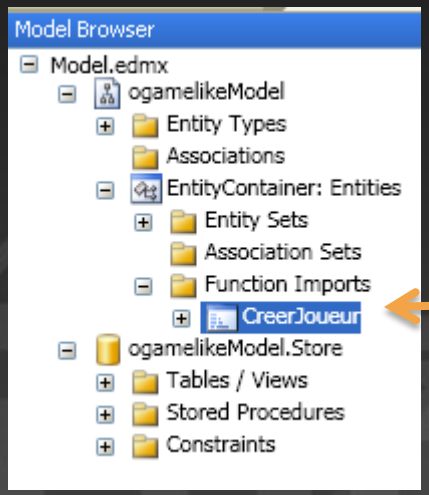


Nom de la procédure dans la base de données

Nom de la procédure dans le code

Type de retour

Attention le type de retour doit toujours être une entité sinon cela ne fonctionne pas



Apparition de la fonction dans l'explorateur

Procédures Stockées

Exemple de la requête :

```
CREATE PROCEDURE [dbo].[CreatePlayer]
    -- Add the parameters for the stored procedure here
     @nom nvarchar(50),
   @prenom nvarchar(50)
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
INSERT INTO [dbo].[T_Joueur]
           (
            [nom]

           ,[prenom]   )
     VALUES
            (@nom ,
             @prenom  )
 -- obligation de faire ca pour retourner l'entité est
 -- que entity framework l'accepte comme une proc stock "valide"
select *
 from [dbo].[T_Joueur]
 where id  = scope_identity()
END
```

Procédures Stockées

Utilisation dans un programme C#:

```
ogamelikeEntities bdd = new ogamelikeEntities();
//utilisation sans type de retour
bdd.CreerJoueur("l", "Spike");
//utilisation avec type de retour
ObjectResult<T_Joueur> res = bdd.CreerJoueur("lemettre", "Arnaud");
//Affichera l'id du joueur lemettre
Console.WriteLine(res.First().id);
```

Procédures Stockées

Grâce à certaines fonctionnalités, on peut faire fonctionner les procédures stockées pour qu'elles retournent autre chose que des entités.

```
CREATE PROCEDURE dbo.InscriptionForDateRange
    @StartDate DATETIME,
    @EndDate DATETIME
AS
SET NOCOUNT ON;
SELECT Count(T_Joueur.DateInscription) AS NbJoueur
FROM T_Joueur
WHERE DateInscription BETWEEN @StartDate AND @EndDate
```

Il suffit après d'importer comme une procédure stockée normale.

Procédures Stockées

Dans le code on peut utiliser la fonction de cette manière :

```
int result = 0;
using (EntityConnection conn = new EntityConnection("Name = ogamelikeEntities1"))
{
    EntityCommand cmd = conn.CreateCommand();
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.CommandText = "ogamelikeEntities1.InscriptionForDateRange";
    cmd.Parameters.AddWithValue("StartDate", new DateTime(2009, 1, 1));
    cmd.Parameters.AddWithValue("EndDate", DateTime.Now);
    conn.Open();

    result = Convert.ToInt32(cmd.ExecuteScalar());
}
```

Accès aux données

- Les requêtes de linq to entities supportent les mêmes opérateurs que linq to SQL

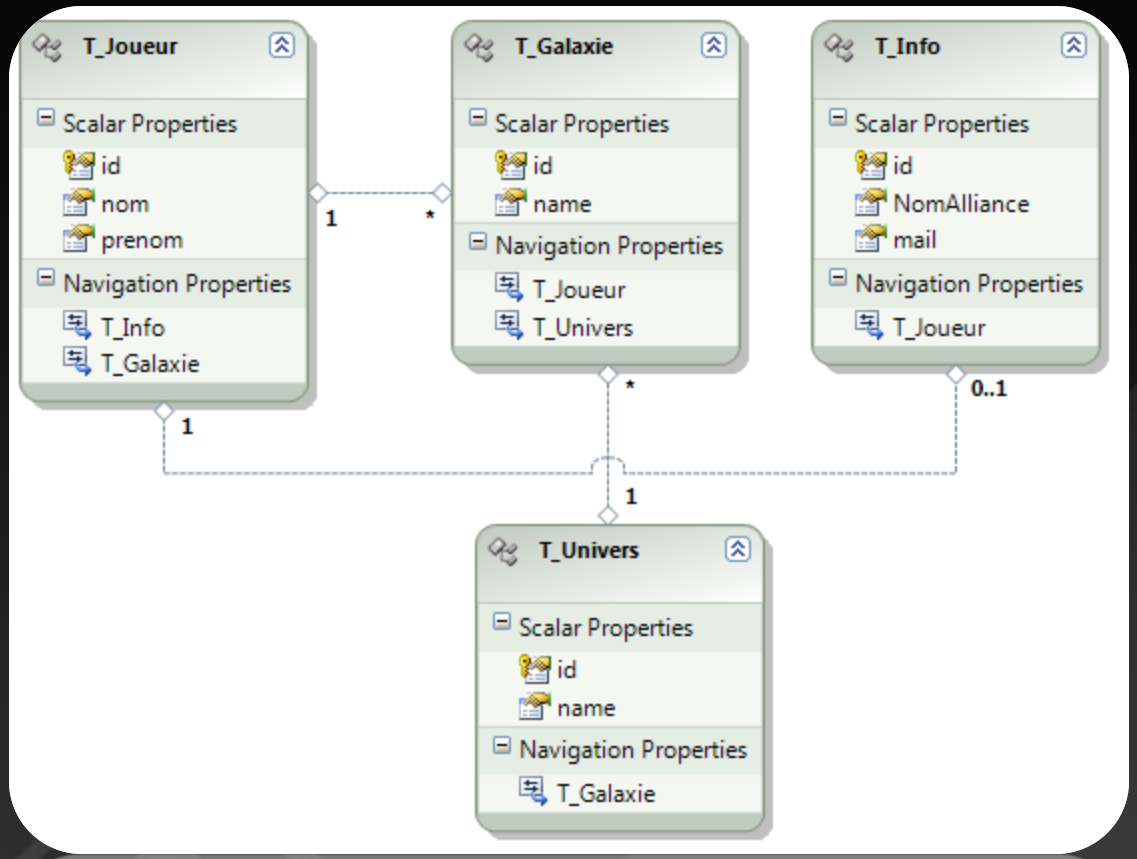
Select, SelectMany
Where
GroupJoin, Join
All, Any
Distinct
Except
Intersect
Union
OrderBy, OrderByDescending, ThenBy, ThenByDescending
GroupBy
Average
Count, LongCount
Max, Min, Sum
OfType, Cast
First, FirstOrDefault
Skip, Take

Accès aux données

- Si vous effectuez une requête et que vous parcourez le résultat vous pouvez avoir une null exception.
- Pour cela il faut rajouter un include au moment de la requête.
- Ou alors effectuer un load sur l'entité, en se servant de isloaded.
- Hélas, dans tous les cas cela ne passera pas il faudra utiliser le nomtableReference

- Ce cas se produit surtout si vous n'utilisez pas le lazy loading

Accès aux données



Soit le schéma suivant.

Si on veut afficher le nom de tous les joueurs,
Ainsi que toutes les galaxies dans lesquelles ils sont présents, puis les univers.

Exemples

```
using (ogamelikeEntities1 bdd = new ogamelikeEntities1())
{
    List<T_Joueur> listJoueur = bdd.T_Joueur.Include("T_Galaxie").ToList();
    foreach (T_Joueur item in listJoueur)
    {
        Console.WriteLine(item.nom);
        if (!item.T_Galaxie.IsLoaded)
        {
            item.T_Galaxie.Load();
        }
        foreach (T_Galaxie galaxie in item.T_Galaxie)
        {
            Console.WriteLine(galaxie.name);
            if (!galaxie.T_UniversReference.IsLoaded)
            {
                galaxie.T_UniversReference.Load();
            }
            Console.WriteLine(galaxie.T_Univers.name);
        }
    }
}
```

Inutile si on a déjà mis un include

Permet de charger l'entité

Remarques

La fonction « include », comme on le voit ici, peut comporter toute l'arborescence .
Ceci peut éviter les load dans le code mais c'est beaucoup plus lourd en requête.

```
using (ogamelikeEntities1 bdd = new ogamelikeEntities1())
{
    List<T_Joueur> listJoueur =
    bdd.T_Joueur.Include("T_Galaxie.T_Univers").ToList();

    foreach (T_Joueur item in listJoueur)
    {
        Console.WriteLine(item.nom);

        foreach (T_Galaxie galaxie in item.T_Galaxie)
        {
            Console.WriteLine(galaxie.name);
            Console.WriteLine(galaxie.T_Univers.name);
        }
    }
}
```

A n'utiliser que dans des Conditions bien particulières

UTILISATION PLUS AVANCÉE

Rajout de fonctionnalité

- Comme dit au début du cours, les entités utilisent le mot clé Partial. Cela permet de rajouter des fonctionnalités à une classe déjà existante. Il faut bien faire attention à avoir le même namespace et nom de classe.

RAJOUT DE FONCTIONNALITÉ

- Exemple :
  - Surcharge de la méthode toString()
  - Implémentation des événements sur le changement d'une propriété

Rajout de fonctionnalité

```
public partial class T_Joueur
{
    /// <summary>
    /// affiche le nom une fois que celui
    /// ci a été changé
    /// </summary>
    partial void OnnomChanged()
    {
        Console.WriteLine(nom);
    }

    /// <summary>
    /// surchage de la méthode tostring
    /// </summary>
    /// <returns>le nom d'un joueur</returns>
    public override string ToString()
    {
        StringBuilder representation = new StringBuilder();
        representation.Append(nom);
        representation.Append(":");
        representation.Append(prenom);
        return representation.ToString();
    }
}
```

Rajout de fonctionnalité

- De la même façon on peut étendre la classe context, pour implémenter des réponses à des événements.

Communication

- Les objets entities peuvent être transportés via WCF. Cette fonction sera vue dans les prochains cours.

- Cependant, sauf dans des cas particuliers, il faut éviter d'y recourir. Car lors du transfert des classes les contraintes sont gardées et donc cela peut poser des problèmes si tout n'est pas bien rempli. Les exceptions arrivent avant l'appel à la méthode.

Communication

- Les entités sont également sérialisables.

Communication

```
using (ogamelikeEntities bdd = new ogamelikeEntities())
{
    T_Joueur joueur = bdd.T_Joueur.Where(x => x.id == 1).FirstOrDefault();
    if (joueur != null)
    {
        XmlSerializer serialiseur = new XmlSerializer(typeof(T_Joueur));
        TextWriter textWriter = new StreamWriter("joueur.xml");
        serialiseur.Serialize(textWriter, joueur);
        textWriter.Close();
    }
}
```


Communication

```
XmlSerializer serialiseur = new XmlSerializer(typeof(T_Joueur));

TextReader textReader = new StreamReader("joueur.xml");
T_Joueur client = (T_Joueur)serialiseur.Deserialize(textReader);
textReader.Close();
```


Concurrence

- L'entity framework implémente un système de gestion de concurrence des données. Cependant il n'est pas actif par défaut.
- De plus la concurrence s'effectue uniquement sur une propriété et pas sur une table.

Entity Framework V4.0

Entity Framework V4.0

- Comble la majorité des faiblesses de la version 1.0
  - Meilleur support des procédures stockées
  - Génération de la base à partir du modèle
  - Amélioration de la génération du SQL
  - Prise en charge des types complexes
  - Génération de modèles (t4)

Entity Framework V4.1

Entity Framework V4.1

- Téléchargement :
  - http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=8363
- Nouveauté :
  - Code First
    - Utilisation de code first : http://msdn.microsoft.com/fr-fr/magazine/hh126815.aspx

Entity Framework V4.3

Entity Framework V4.3-4.5

- Disponible au travers du système de package Nuget
  - Amélioration de Code First (gestion de la migration …)

QUESTIONS ?