



ADO.NET

TP ADO.NET Expert Entity Framework

Objectifs :

Eviter la plomberie

Dans TP précédent on avait des classes métier et des classes de service (Employe et EmployeServices). L'idée est de ne plus avoir à écrire ces classes

On doit rester agnostique à la BD

Langage typé qui permet d'écrire des requêtes sur des objets et collections (Linq)

On pourra travailler selon 2 modes :

Database first : on a la BD et on va écrire le code autour

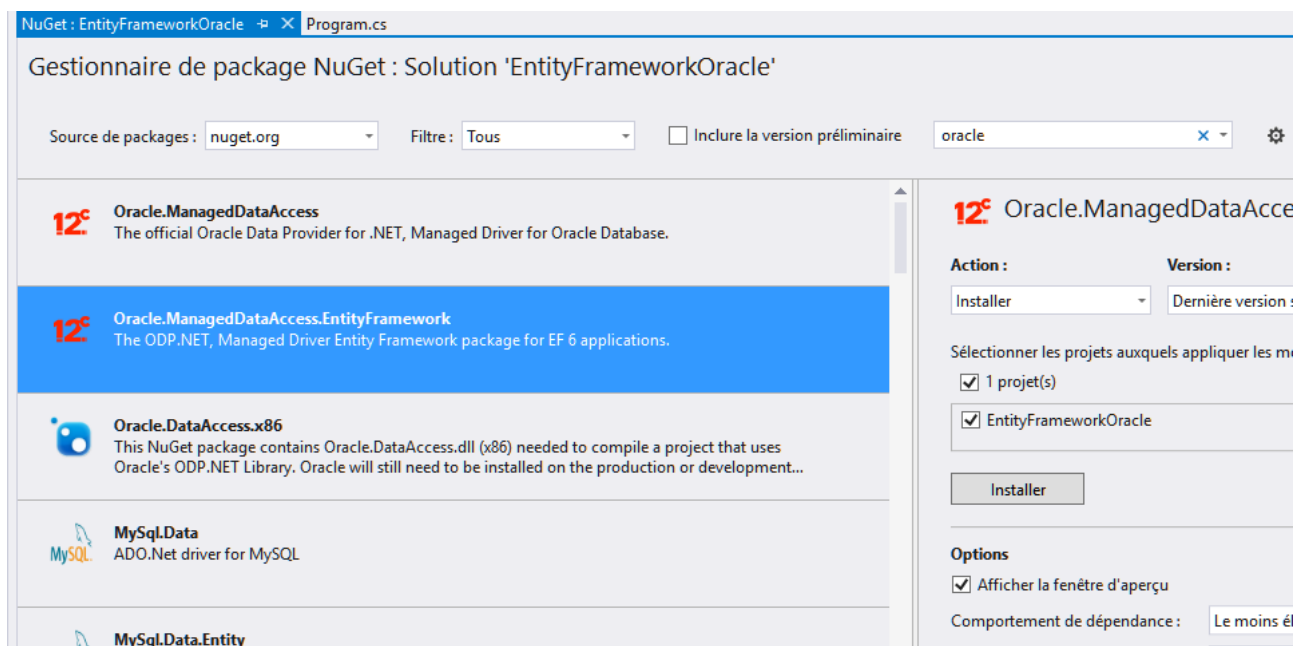
Code first : On a le code et on veut le mapper à une BD.

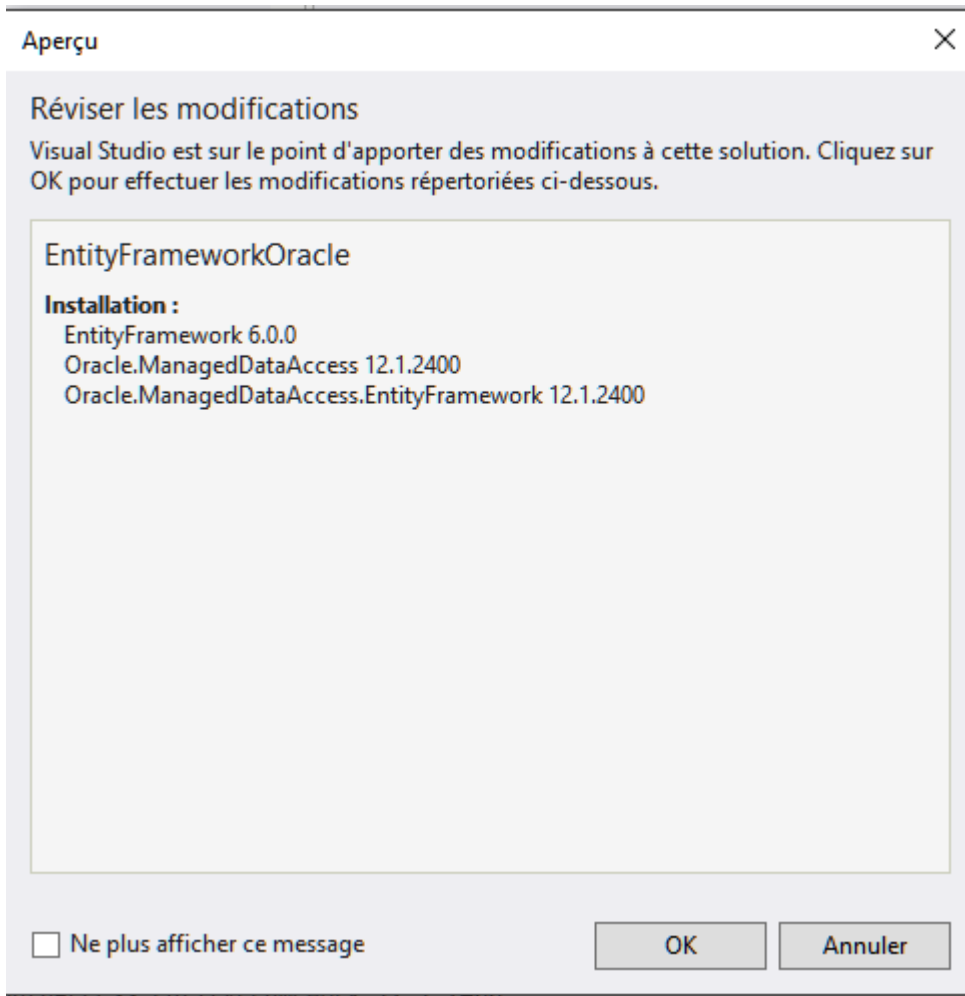
On verra dans ce TP la méthode Database First.

Partie 1 Mise en place

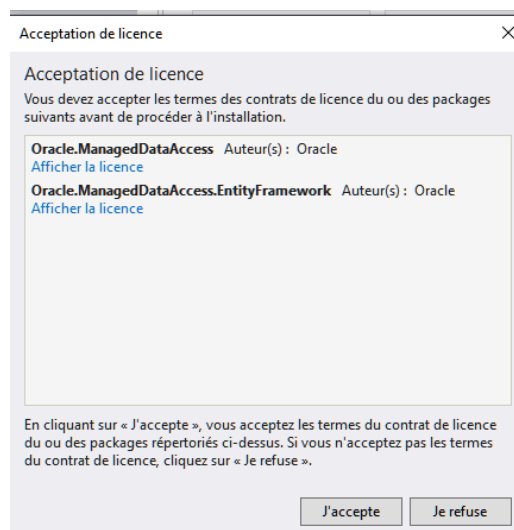
Créer un projet console appelé EntityFrameworkOracle

Dans le gestionnaire de dépendance NuGet, rajouter EntityFramework 6.



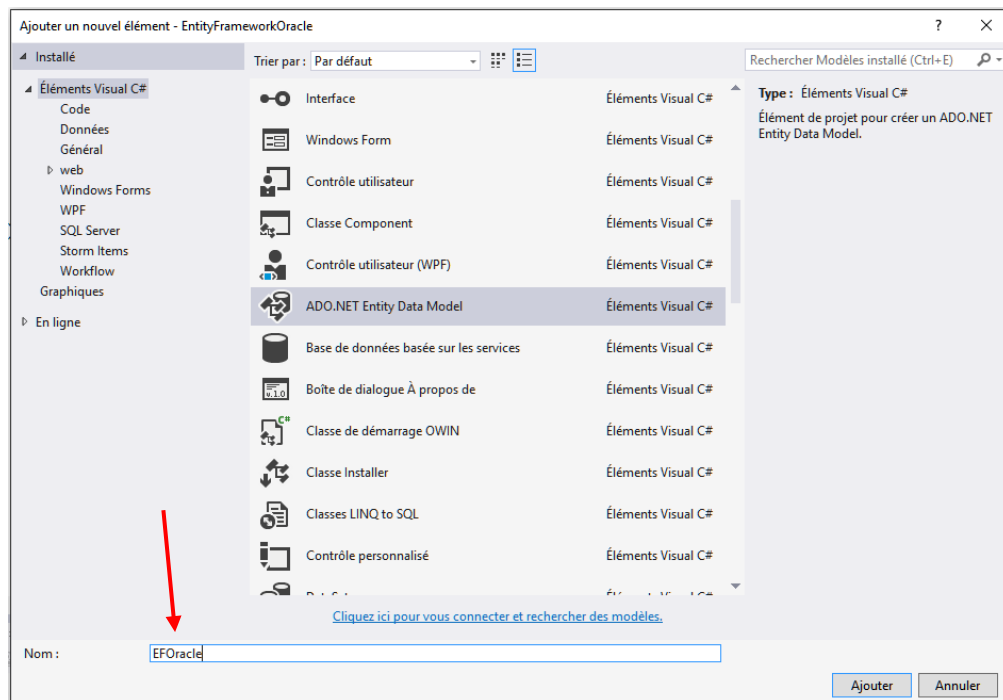


Dans ce projet rajouter un nouvel élément du type ADO.NET Entity Data Model appelé EFOracle

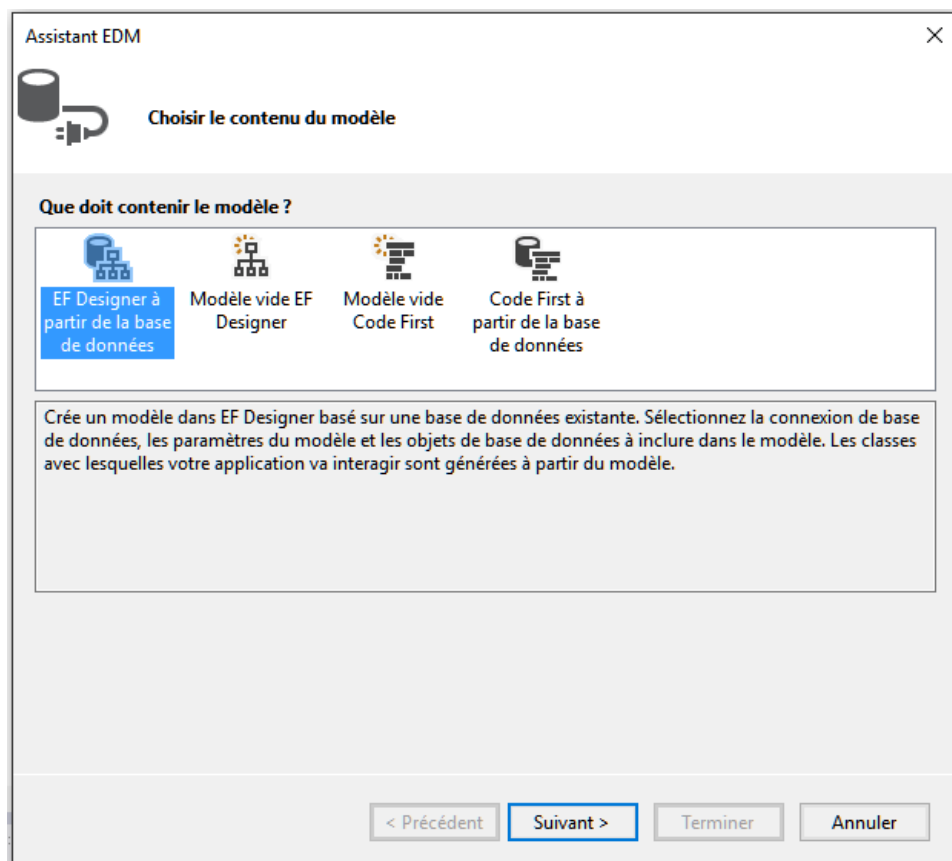


Vérifier le provider Oracle pour voir s'il est à jour :

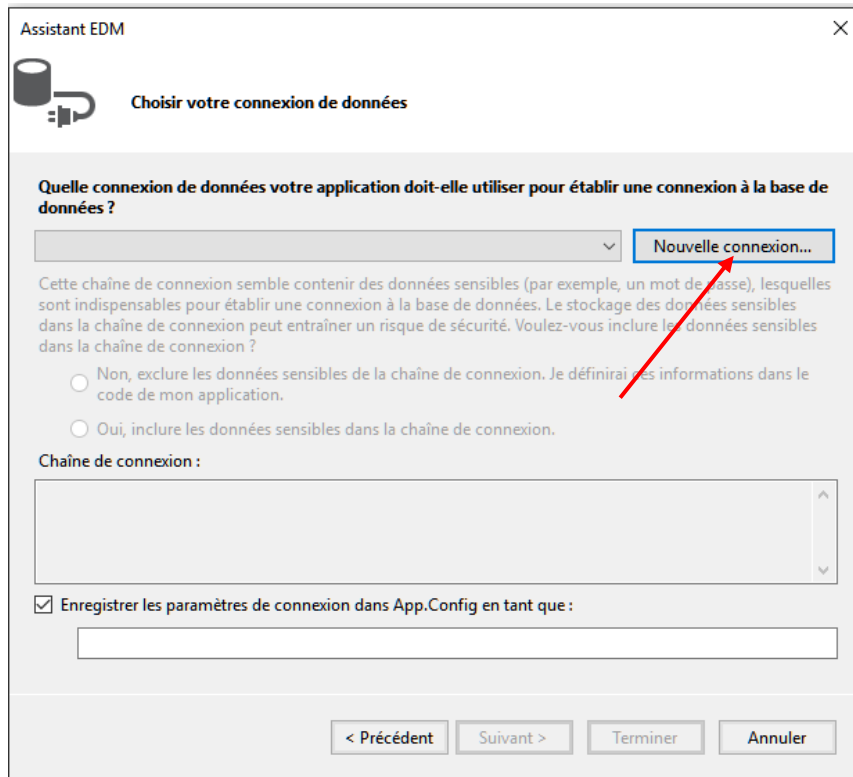
Il est installé et à jour.



On va faire du Database First :



On arrive sur cette fenêtre. Il va donc falloir créer la connexion.
Par défaut VS nous propose une connexion sur SQL Server



Assistant EDM

Choisir votre connexion de données

Quelle connexion de données votre application doit-elle utiliser pour établir une connexion à la base de données ?

Nouvelle connexion...

Cette chaîne de connexion semble contenir des données sensibles (par exemple, un mot de passe), lesquelles sont indispensables pour établir une connexion à la base de données. Le stockage des données sensibles dans la chaîne de connexion peut entraîner un risque de sécurité. Voulez-vous inclure les données sensibles dans la chaîne de connexion ?

☐ Non, exclure les données sensibles de la chaîne de connexion. Je définirai ces informations dans le code de mon application.

☐ Oui, inclure les données sensibles dans la chaîne de connexion.

Chaîne de connexion :

☒ Enregistrer les paramètres de connexion dans App.Config en tant que :

< Précédent Suivant > Terminer Annuler

Vous remplirez les champs avec les données propres à votre connexion :

Propriétés de connexion ? X

Entrez les informations pour vous connecter à la source de données sélectionnée ou cliquez sur "Modifier" pour sélectionner une autre source de données et/ou un autre fournisseur.

Source de données : Base de données Oracle (ODP.NET - Pilote géré) Modifier...

Détails de la connexion Filters

Indiquez les détails suivants pour ajouter une nouvelle

☐ Utiliser l'authentification intégrée Windows

Nom utilisateur : COURSADO

Mot de passe : *****

☒ Enregistrer le mot de passe

☐ Connexion avec le rôle SYSBDA

Type de connexion : TNS

Nom de source de données : localOracleOdp

Fichiers tnsnames.ora en cours d'utilisation : c:\app\oracle\product\12.1.0\client_1\network\admin\tnsnan Rechercher.

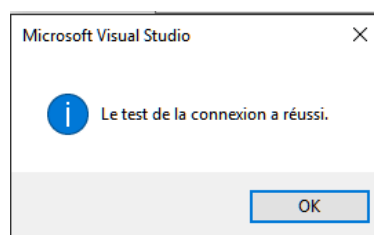
Nom de connexion : COURSADO.localOracleOdp

Avancées...

Tester la connexion OK Annuler

Vous remarquerez que VS vous a proposé directement Oracle. C'est normal, n'oubliez pas que vous avez téléchargé et installé sur vos poste les composants Oracle pour VS : ODP (Oracle Data Product for Visual Studio)

Testez la connexion



Assistant EDM

Choisir votre connexion de données

Quelle connexion de données votre application doit-elle utiliser pour établir une connexion à la base de données ?

COURSADO.localOracleOdp

Nouvelle connexion...

Cette chaîne de connexion semble contenir des données sensibles (par exemple, un mot de passe), lesquelles sont indispensables pour établir une connexion à la base de données. Le stockage des données sensibles dans la chaîne de connexion peut entraîner un risque de sécurité. Voulez-vous inclure les données sensibles dans la chaîne de connexion ?

☐ Non, exclure les données sensibles de la chaîne de connexion. Je définirai ces informations dans le code de mon application.

☒ Oui, inclure les données sensibles dans la chaîne de connexion.

Chaîne de connexion :

```
metadata=res://*/EFOracle.csd|res://*/EFOracle.ssd|
res://*/EFOracle.mst;provider=Oracle.ManagedDataAccess.Client;provider connection string="DATA
SOURCE=localOracleOdp;PERSIST SECURITY INFO=True;USER ID=COURSADO"
```

☒ Enregistrer les paramètres de connexion dans App.Config en tant que :

OracleEntities

< Précédent Suivant > Terminer Annuler

Dans cette fenêtre, on a un avertissement de sécurité. Vous le lirez. Nous sommes en phase de test et donc la sécurité n'est pas ici notre préoccupation principale, on va donc demander à ce que le mot de passe soit intégré à la chaîne de connexion.

Vous remarquerez que les paramètres de connexion sont enregistrés dans le fichier App.config.

Assistant EDM

Choisir vos paramètres et objets de base de données

Quels objets de base de données voulez-vous inclure dans votre modèle ?

☒ Tables

☒ COURSADO

☒ COURS

☒ EMPLOYE

☒ INSCRIT

☒ PARTICIPER

☒ PROJET

☒ SEMINAIRE

☐ Vues

☐ Procédures et fonctions stockées

☒ Mettre au pluriel ou au singulier les noms d'objets générés

☒ Inclure les colonnes de clés étrangères dans le modèle

☒ Importer les fonctions et les procédures stockées sélectionnées dans le modèle d'entité

Espace de noms du modèle :

CoursAdoModel

< Précédent Suivant > Terminer Annuler

On sélectionne toutes les tables, pour l'instant ... on verra les vues et procédures stockées ultérieurement.



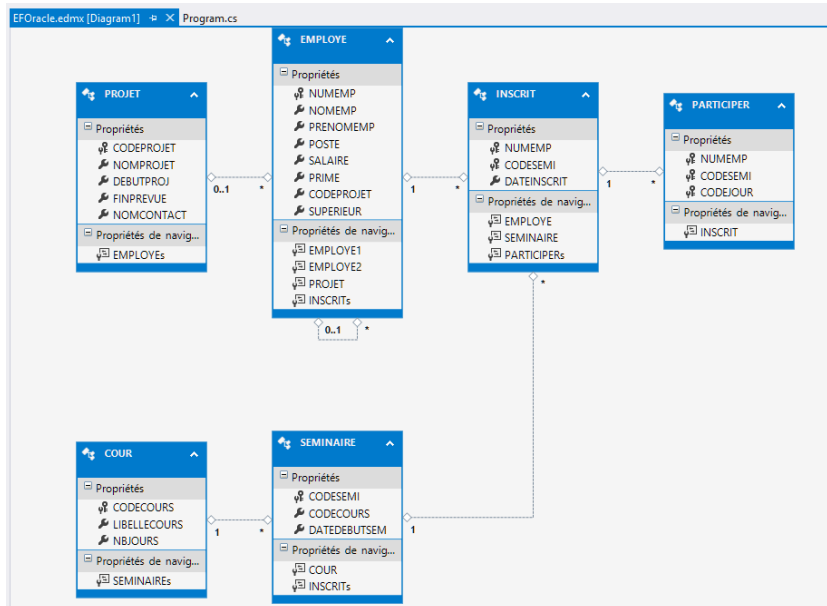
Cochez la case Mettre au pluriel ou au singulier les noms d'objets générés. Ainsi les classes seront au singulier, les collections au pluriel.

Donnez le nom CoursAdoModel à l'espace de noms de ce modèle.

Cliquez sur Terminer.

Beaucoup de choses vont se passer qui vont nous faire gagner beaucoup de temps.

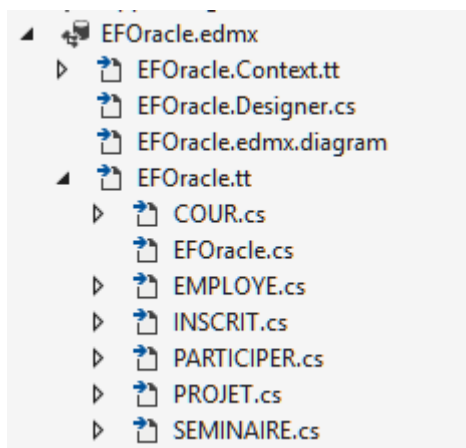
On arrive tout d'abord sur ce diagramme de classes :



Il est l'exacte représentation du MLD (Modèle Logique des Données) sur lequel on travaille depuis longtemps.

Regardons le reste ...

Dans l'explorateur de solutions : on voit que les classes métier ont été générées.



Une Table= Une Classe, appelée Entité ou **Entity** en anglais

Chaque table a été mappée.

Allons voir la classe Cours :

```

public partial class COUR
{
    0 références
    public COUR()
    {
        this.SEMINAIREs = new HashSet<SEMINAIRE>();
    }

    0 références
    public string CODECOURS { get; set; }
    0 références
    public string LIBELLECOURS { get; set; }
    0 références
    public decimal NBJOURS { get; set; }

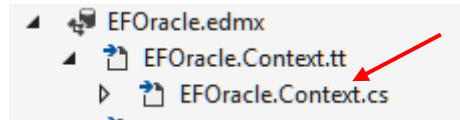
    1 référence
    public virtual ICollection<SEMINAIRE> SEMINAIREs { get; set; }
}

```



Que remarquez-vous ?
 ... beaucoup de choses
 D'où cela vient-il ?

Allons un peu plus loin, allons examiner la classe EFOracle.Context



```

1 référence
public partial class OracleEntities : DbContext
{
    0 références
    public OracleEntities()
        : base("name=OracleEntities")
    {
    }

    0 références
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    0 références
    public virtual DbSet<COUR> COURS { get; set; }
    0 références
    public virtual DbSet<EMPLOYEE> EMPLOYEEs { get; set; }
    0 références
    public virtual DbSet<INSCRIT> INSCRITs { get; set; }
    0 références
    public virtual DbSet<PARTICIPER> PARTICIPERs { get; set; }
    0 références
    public virtual DbSet<PROJET> PROJETs { get; set; }
    0 références
    public virtual DbSet<SEMINAIRE> SEMINAIREs { get; set; }
}

```


Cette classe OracleEntities référence chaque Entity, cette classe représente en fait la classe de services vue dans le précédent TP.



Exercice:

Explorer un peu les classes générées

Partie 2 Manipulation des classes : Linq

Et en pratique ???? On va écrire un peu de code pour mettre tout ça en œuvre.

Complétez le programme principal comme ceci :

```
static void Main(string[] args)
{
    using (OracleEntities oracleContexte = new OracleEntities())
    {
        // on va afficher tous les employes
        // requête linq
        var requeteEmployes = from EMPLOYE in oracleContexte.EMPLOYEs
                               select EMPLOYE;
        // exécution de la requête
        var lesEmployes = requeteEmployes.ToList();

        // affichage du résultat
        foreach(var unEmploye in lesEmployes)
        {
            Console.WriteLine(unEmploye.NUMEMP + " - " + unEmploye.NOMEMP);
        }
    }
    Console.ReadLine();
}
```

On notera l'utilisation d'un nouveau langage d'interrogation : LINQ

Il s'apparente au SQL, mais le SQL interroge des tables ou des vues d'une base de données, LINQ interroge des listes (collections) d'objets.

Résultat :

```
file:///T:/cours/Csharp/ADO_Net/TP/Ent
1 - DUPONT
2 - JOLIBOIS
3 - BEAUMONT
4 - DUCHATEL
5 - MARTIN
6 - MAZAUD
7 - GIMOND
8 - SAULT
9 - GALLI
10 - JACONO
11 - ANTHONY
12 - CANE
13 - GOMEZ
14 - RIVIERE
15 - RUSSOT
16 - BERNARDI
17 - BEUGNIES
18 - FARNY
19 - ESTIVAL
20 - LUNEAU
21 - BRESSON
```

On va s'amuser encore un peu, on va afficher les employés travaillant sur le projet PR1 :

Rajoutez le code suivant à la suite de l'affichage de tous les employés :

```
Console.WriteLine("-----");
var unCodeProjet = "PR1";
var requeteEmployesProjet = from EMPLOYE in oracleContexte.EMPLOYEs
                             where EMPLOYE.CODEPROJET.TrimEnd()==unCodeProjet
                             select EMPLOYE;

// exécution de la requête
lesEmployes = requeteEmployesProjet.ToList();

// affichage du résultat
foreach (var unEmploye in lesEmployes)
{
    Console.WriteLine(unEmploye.NUMEMP + " - " + unEmploye.NOMEMP );
}
```



On voit déjà beaucoup de similitude entre le SQL et Linq.



On remarque la condition : `where EMPLOYE.CODEPROJET.TrimEnd()==unCodeProjet`

La fonction TrimEnd() va supprimer les blancs inutiles en fin de chaîne. Ici c'est indispensable car la colonne codeprojet dans la table employe & été déclarée en char(4) c'est-à-dire en tant que chaîne de longueur fixe 4 caractères.

Amusons nous encore un peu.... On va faire la même requête mais la condition va porter sur l'id.... On va demander le nom, le prénom et le salaire de l'employé numéro 3.

Dans ce cas, la requête SQL sur la base va ramener un enregistrement au plus. Donc pas besoin de ramener une liste puisqu'au final on ne travaillera QUE sur un seul objet.



Exercice: Ecrire le code, en utilisant la méthode First().

Solution :

```
Console.WriteLine("-----");
var idEmploye = 3;
var requeteEmployesById = from EMPLOYE in oracleContexte.EMPLOYEs
                           where EMPLOYE.NUMEMP==idEmploye
                           select EMPLOYE;

// exécution de la requête
var employeeId = requeteEmployesById.First();
Console.WriteLine(employeeId.NOMEMP + " - " + employeeId.PRENOMEMP + " - " + employeeId.SALAIRE);
```



On remarque la déclaration de employeeId en type générique var....

Posez votre curseur sur la méthode First() et appuyez sur F12.
Que constatez-vous pour le prototype de cette fonction ?
Qu'en déduisez-vous?

Faites un test avec l'employé numéro 33.
Que se passe-t-il ?

Pour éviter ce problème, on va utiliser la méthode FirstOrDefault qui retournera l'employé si il existe ou null.

Ainsi le code devient :

```
Console.WriteLine("-----");
var idEmploye = 33;
var requeteEmployesById = from EMPLOYE in oracleContexte.EMPLOYEs
                           where EMPLOYE.NUMEMP==idEmploye
                           select EMPLOYE;

// exécution de la requête
var employeeId = requeteEmployesById.FirstOrDefault();

if (employeeId != null)
{
    Console.WriteLine(employeeId.NOMEMP + " - " + employeeId.PRENOMEMP + " - " + employeeId.SALAIRE);
}
else
{
    Console.WriteLine("L'employé numéro " + idEmploye + " n'existe pas.");
}
```

A l'exécution on obtient :

```
-----
L'employé numéro 33 n'existe pas.
```



Vous pouvez vous entraîner à l'utilisation d'autres méthodes.



Et si on faisait une jointure ?

On voudrait afficher pour chaque séminaire, son code, le code du cours correspondant et le libellé du cours.

La requête SQL serait :

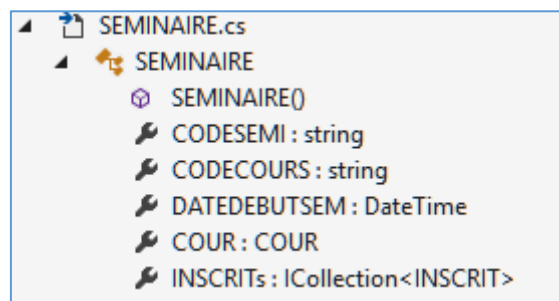
```
select codesemi, seminaire.codecours, libellecours
from seminaire inner join cours
on seminaire.codecours=cours.codecours;
```

La requête linq correspondante sera :

```
var requete = from s in oracleContexte.SEMINAIRES
              join COUR in oracleContexte.COURS on s.CODECOURS equals COUR.CODECOURS
              select s;
var lesSeminaires = requete.ToList();
```

Beaucoup de choses à voir :

`select s;` On va ramener des objets de la classe Seminaire. On peut s'interroger ???
Si on regarde de plus près les attributs de la classe Seminaire, on voit ceci :



La classe Seminaire contient un attribut COUR qui correspond à un objet de la classe COUR. La jointure va donc consister à hydrater cet objet pour chaque onbjet de la classe Seminaire, avec le bon objet cours .

C'est exactement ce que l'on a fait ici :

```
join COUR in oracleContexte.COURS on s.CODECOURS equals COUR.CODECOURS
```

Pour afficher les séminaires avec leur libellé de cours, on n'a plus qu'à balayer la collection lesSeminaires.

```
foreach(var unSeminaire in lesSeminaires)
{
    Console.WriteLine(unSeminaire.CODESEMI + " - " + unSeminaire.CODECOURS + " - " + unSeminaire.COUR.LIBELLECOURS);
}
```

Et voici le résultat :

```
-----
BR0350216 - BR035 - UML : Perfectionnement
BR0350316 - BR035 - UML : Perfectionnement
BR0350525 - BR035 - UML : Perfectionnement
BR0350907 - BR035 - UML : Perfectionnement
BR0351019 - BR035 - UML : Perfectionnement
BR0340413 - BR034 - UML : Initiation
BR0340518 - BR034 - UML : Initiation
BR0340921 - BR034 - UML : Initiation
BR0341020 - BR034 - UML : Initiation
BR0341207 - BR034 - UML : Initiation
BR0130223 - BR013 - Administration BDR
BR0130914 - BR013 - Administration BDR
BR0560210 - BR056 - SQL2 : La norme
BR0560525 - BR056 - SQL2 : La norme
BR0560914 - BR056 - SQL2 : La norme
BR0570323 - BR057 - Conception BD relationnelle
BR0570928 - BR057 - Conception BD relationnelle
BR0571012 - BR057 - Conception BD relationnelle
BR0701012 - BR070 - Administration d'une BD
```



Avant de poursuivre, mettez un point d'arrêt sur l'instruction foreach, et lancez l'exécution. Allez dans



Exercice:

En vous basant sur cette requête, écrire le code permettant d'obtenir ceci :

```
----- Cours et séminaires -----
-----
BR035 - UML : Perfectionnement - 5
        16/02/2015 00:00:00
        16/03/2015 00:00:00
        25/05/2015 00:00:00
        07/09/2015 00:00:00
        19/10/2015 00:00:00

BR034 - UML : Initiation - 2
        13/04/2015 00:00:00
        18/05/2015 00:00:00
        24/09/2015 00:00:00
        20/10/2015 00:00:00
        07/12/2015 00:00:00

BR013 - Administration BDR - 3
        23/02/2015 00:00:00
        14/09/2015 00:00:00

BR056 - SQL2 : La norme - 3
        11/02/2015 00:00:00
        25/05/2015 00:00:00
        14/09/2015 00:00:00

BR057 - Conception BD relationnelle - 4
        23/03/2015 00:00:00
        28/09/2015 00:00:00
        12/10/2015 00:00:00

BR070 - Administration d'une BD - 4
        12/10/2015 00:00:00
```



Et si on faisait un regroupement ?

On va afficher le nombre d'employés par projet (codeprojet, nombre)

2 points nouveaux vont être vus :

- ✓ Le group by en linq
- ✓ Le retour d'une collection d'objets qui ne correspond pas à une table.

Voici le code :

```
var employeProjet = from emp in oracleContexte.EMPLOYEs
                    group emp by emp.CODEPROJET into groupeEmployes
                    select new
                    {
                        Projet = groupeEmployes.Key,
                        Nombre = groupeEmployes.Count()
                    };
foreach (var ligne in employeProjet.ToList())
{
    Console.WriteLine(ligne.Projet + " - " + ligne.Nombre);
}
```

Que s'est-il passé ?

On a récupéré les employés dans emp

```
var employeProjet = from emp in oracleContexte.EMPLOYEs
```

Dans emp, on fait un regroupement par CODEPROJET

```
group emp by emp.CODEPROJET into groupeEmployes
```

Et on récupère le résultat dans une nouvelle Entity anonyme.

```
select new
{
    Projet = groupeEmployes.Key,
    Nombre = groupeEmployes.Count()
};
```

Dans cette entité anonyme, on crée

- ✓ l'attribut Projet auquel on attribue la clé du group by
- ✓ l'attribut Nombre auquel on affecte le résultat du count du nombre d'employés sur un projet

Parcours du résultat :

```
foreach (var ligne in employeProjet.ToList())
{
    Console.WriteLine(ligne.Projet + " - " + ligne.Nombre);
}
```

On ne passe pas par une variable intermédiaire.



On aurait pu créer des Entités (Entities) anonymes sur toutes les requêtes que l'on a fait jusqu'à présent, ce qui aurait permis de ne mettre dans ces Entités anonymes QUE les attributs dont on avait besoin.

Et voici le résultat :

```
----- Nombre d'employés par projet -----
PR1  - 4
PR3  - 3
      - 6
PR5  - 2
PR4  - 1
PR2  - 5
```

Il y a un petit problème d'affichage pour le codeprojet null, mais ici cela n'a pas d'importance.



Exercice: Corriger la requête et faire apparaître le nom de projet.
Le résultat final doit être :

```
- Nombre d'employés par projet Version 2 -
PR4  - Refonte base pieces - 1
PR2  - Paye ADCV - 5
PR3  - Pensions - 3
PR5  - Miroiterie Bonnet - 2
PR1  - Anciens combattants - 4
```

Solution :

```
var employeProjetNom = from emp in oracleContexte.EMPLOYEs
                        join prj in oracleContexte.PROJETs on emp.CODEPROJET equals prj.CODEPROJET
                        group emp by new { prj.CODEPROJET, prj.NOMPROJET } into groupeEmployes
                        select new
                        {
                            Code = groupeEmployes.Key.CODEPROJET,
                            Nom= groupeEmployes.Key.NOMPROJET,
                            Nombre = groupeEmployes.Count()
                        };

```



Vous remarquerez une l'entity temporaire et anonyme qu'il a fallu créer pour le regroupement sur plusieurs colonnes

La modification de l'entity anonyme et l'utilisation de l'attribut Key...

N'oubliez pas,

On est en TOUT OBJET !!!



Exercice:

Un exercice ambitieux....

Sans toucher la moindre ligne de code de Main(), afficher les mêmes résultats, mais depuis une base mysql.....

Partie 3 Mise à jour des données

La mise à jour des données est relativement simple.

Le principe est de mettre à jour les objets ou les objets de la collection et de faire appliquer les changements sur la base, grâce à la méthode DbContext.SaveChanges().

Extrait de la table employe sur le serveur Oracle :

NUMEMP	NOMEMP	PRENOMEMP	POSTE	SALAIRE	PRIME	CODEPROJET	SUPERIEUR
1	DUPONT	Pierre	A22	8000	1000	PR1	4
2	JOLIBOIS	Rolland	A12	10500	1500	PR2	5
3	BEAUMONT	Jean	A25	12000	2000	PR1	6

Testez le code suivant :

```
using (OracleEntities oracleContexte = new OracleEntities()) {
    var emp = oracleContexte.EMPLOYEs.Find(2);
    emp.SALAIRE = emp.SALAIRE * (decimal) 1.1;
    Console.WriteLine("Le nouveau salaire = " + emp.SALAIRE);
    oracleContexte.SaveChanges();
}
```

Affichage :

```
Le nouveau salaire = 11550,0
```

Allons vérifier dans la BD :

NUMEMP	NOMEMP	PRENOMEMP	POSTE	SALAIRE	PRIME	CODEPROJET	SUPERIEUR
1	DUPONT	Pierre	A22	8000	1000	PR1	4
2	JOLIBOIS	Rolland	A12	11550	1500	PR2	5
3	BEAUMONT	Jean	A25	12000	2000	PR1	6



Exercice:

Essayez avec l'employé numéro 200. Eventuellement corrigez le code.
Vous devriez afficher ceci :

```
L'employé numéro 33 n'existe pas
```

On peut également, sur le même principe, rajouter des enregistrements dans la base.

On va rajouter un nouveau cours

L'idée est donc de

Créer un objet de la classe Cour ,
De le rajouter à la collection Cours du contexte
De mettre à jour le contexte :

```
COUR unCours = new COUR();  
unCours.CODECOURS = "BR099";  
unCours.LIBELLECOURS = "Entity Framework 6 avec Oracle";  
unCours.NBJOURS = 4;  
oracleContexte.COURS.Add(unCours);  
oracleContexte.SaveChanges();  
Console.WriteLine("Le cours a été créé");
```

Et voici dans la base ...

6	BR070	Administration d'une BD	4
7	BR099	Entity Framework 6 avec Oracle	4



Pourquoi a-t-on instancié l'objet cours de cette façon ?

Partie 4 Ecriture de code

Ecrire les méthodes ToString() des classes Employe et Cours.
Vous adapterez l'affichage en tenant compte des valeurs *NULL*.

Testez les dans la méthode Main.

Partie 5 Modification de la structure de la base

Ajoutez dans la table de la base de données le champ Tarif (not null, valeur par défaut 1200, number(6,2)).

Mettez à jour la partie Entity Framework.
Testez à nouveau la méthode Main()

Bonne chance, travaillez avec la banane !!!

