

Entity Framework

« Mapping Relationnel-Objet »

Woodson JUSTE
wjuste@dawan.fr



OBJECTIFS

- ☐ Mise en place d'Entity Framework dans un projet .Net
- ☐ Réalisation du mapping des entités
- ☐ Manipulation des données de la BDD



Plan

- Introduction
- Présentation d'Entity Framework
- Mise en place
- Gestion du contexte
- Mapping des entités
- Opérations du contexte
- Requêtes LINQ To Entities, SQL
- Migrations

Introduction

Le modèle relationnel et le modèle objet

■ Les différences entre ces deux modèles :

Il existe différents problèmes concernant la correspondance entre le modèle objet et relationnel :

- Le modèle objet propose plus de fonctionnalité : le polymorphisme, l'héritage.
- Les relations entre deux entités sont différentes : associations pour le modèle relationnel et collection (liste) pour le modèle objet.
- Les objets ne possèdent pas d'identifiant unique contrairement au modèle relationnel.

Accès aux données en .NET

Aujourd'hui pour accéder à la base de données il existe des solutions selon la technologie utilisée :

- JDBC pour la technologie Java.
- **ADO.NET** pour la technologie .Net

Ce sont des API qui permettent l'accès aux données

Accès aux données en .NET

■ Inconvénients

Il existe deux principaux inconvénients à l'utilisation de ces APIs :

- L'écriture de nombreuses lignes de codes répétitives.
- Il faut connaître l'ensemble des champs et tables de la base de données.
- La liaison entre les objets et les tables est un travail de bas niveau donc fastidieux et compliqué.

Mapping Relationnel Objet (ORM)

■ Définition

Concept permettant de connecter un modèle objet à un modèle relationnel.

Couche qui va interagir entre l'application et la base de données.



Mapping Relationnel Objet (ORM)

- Pourquoi utiliser ce concept ?

- Pas besoin de connaître l'ensemble des tables et des champs de la base de données
- Faire abstraction de toute la partie SQL d'une application.

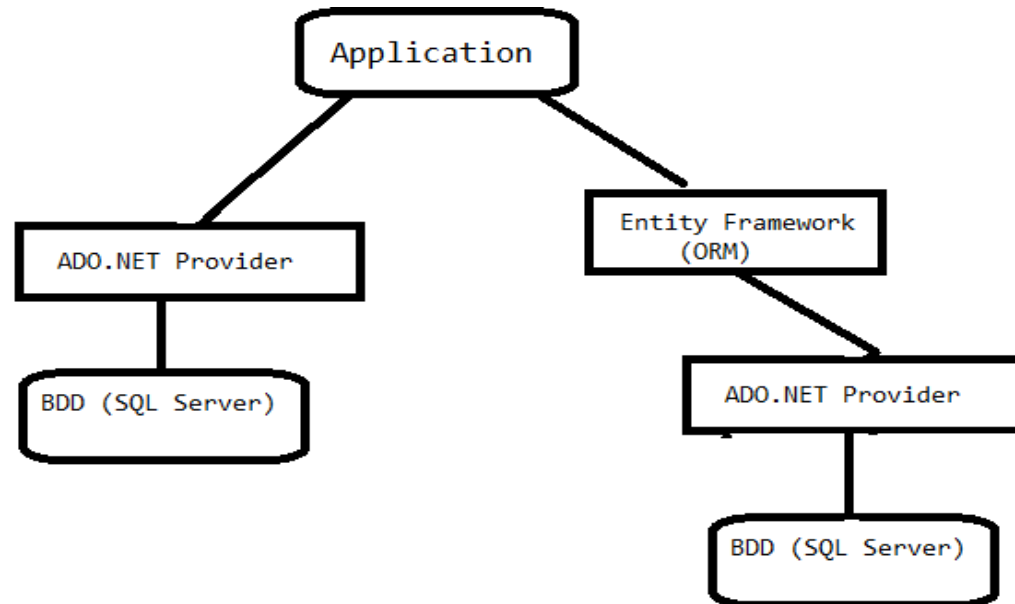
- Frameworks disponibles :

- **Entity Framework**, NHibernate, **Linq To SQL**

Mapping Relationnel Objet (ORM)

■ Avantages

- Gain de temps au niveau du développement d'une application.
- Abstraction de toute la partie SQL.
- La portabilité de l'application d'un point de vue SGBD.



Mapping Relationnel Objet (ORM)

■ Inconvénients

- L'optimisation des frameworks/outils proposés
- La difficulté à maîtriser les frameworks/outils.

Critères de choix d'un ORM

- La facilité du mapping des tables avec les classes, des champs avec les attributs.
- Les fonctionnalités de bases des modèles relationnel et objet.
- Les performances et optimisations proposées : gestion du cache, chargement différé.
- Les fonctionnalités avancées : gestion des sessions, des transactions
- Intégration IDE : outils graphiques
- La maturité.

Entity Framework

Présentation

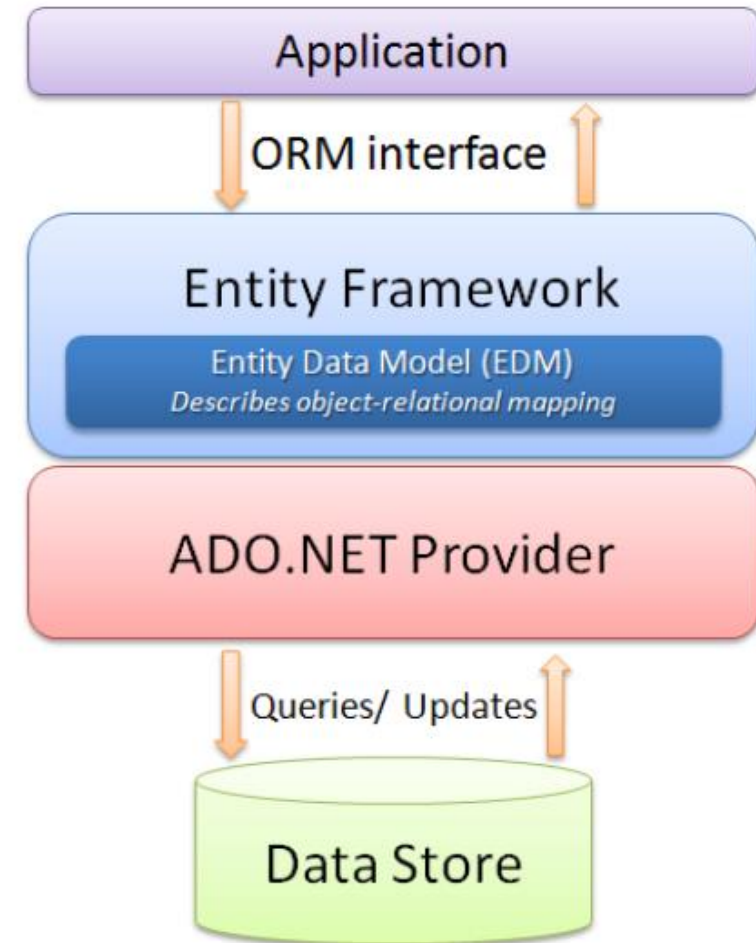
Entity Framework est un mappeur objet/relationnel qui permet aux développeurs .NET d'utiliser des données relationnelles à l'aide d'objets spécifiques au domaine. Il rend inutile la plupart du code d'accès aux données que les développeurs doivent généralement écrire

[Vue d'ensemble d'Entity Framework 6 - EF6 | Microsoft Learn](#)

[Bien démarrer avec Entity Framework 6 - EF6 | Microsoft Learn](#)

Mapping Objet Relationnel avec Entity Framework

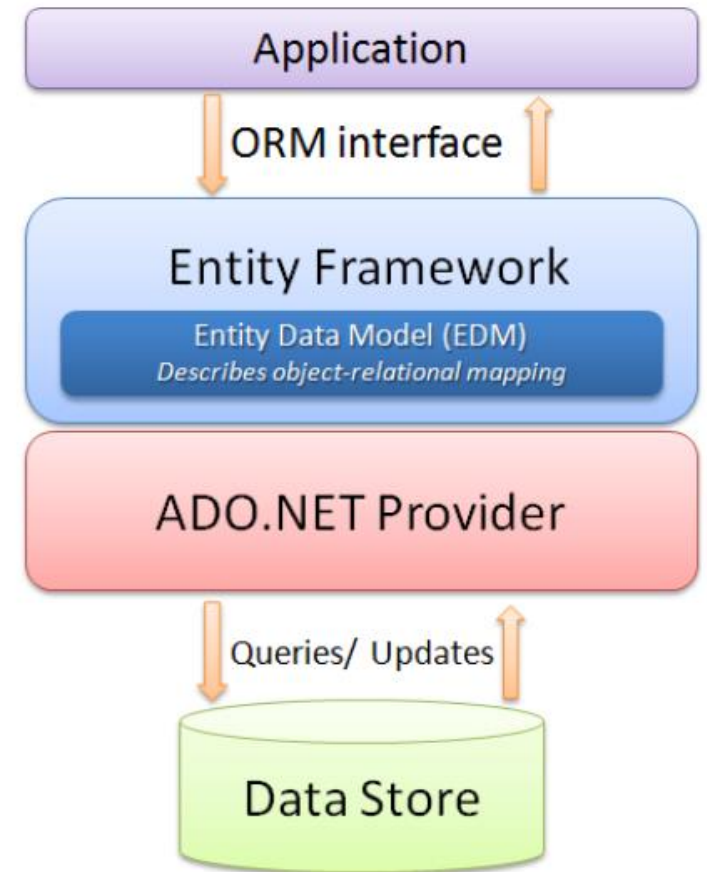
Il fournit une couche d'abstraction nécessaire aux développeurs pour qu'ils n'accèdent plus directement à la base de données, mais par l'intermédiaire d'entités définies par un modèle appelé EDM (Entity Data Model).



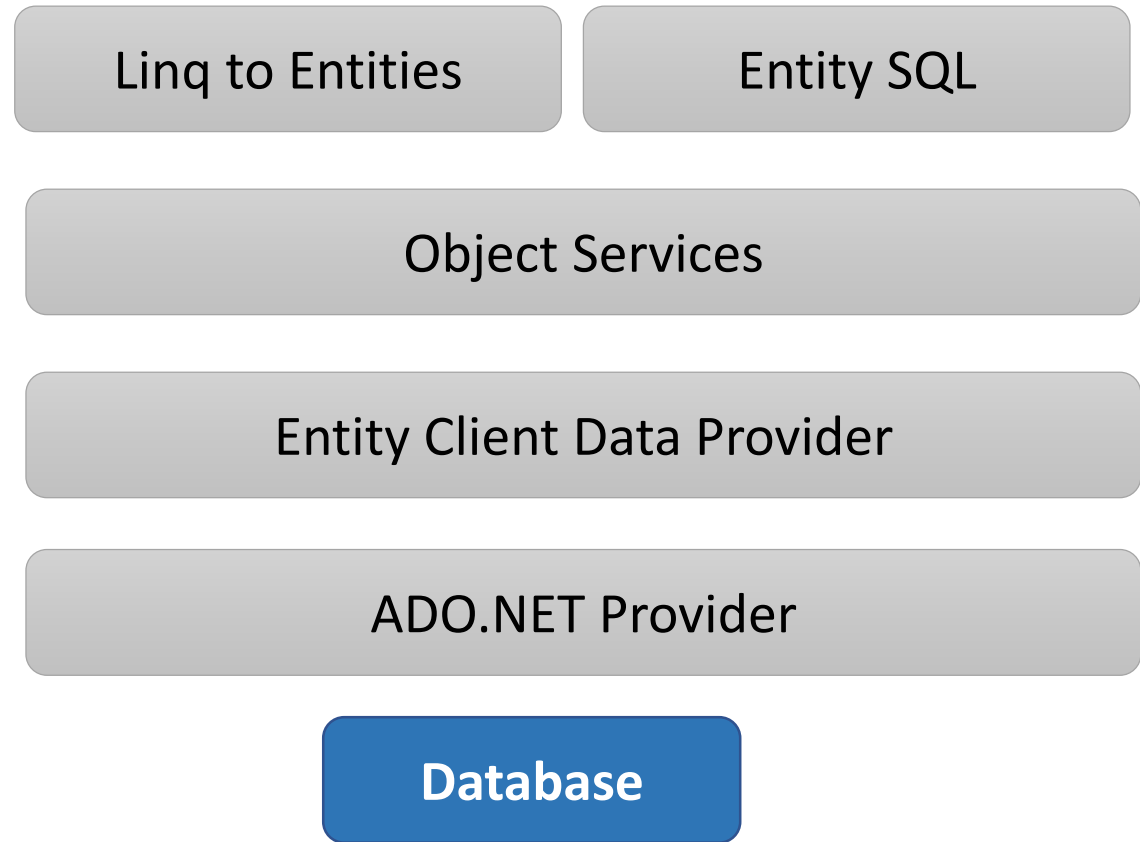
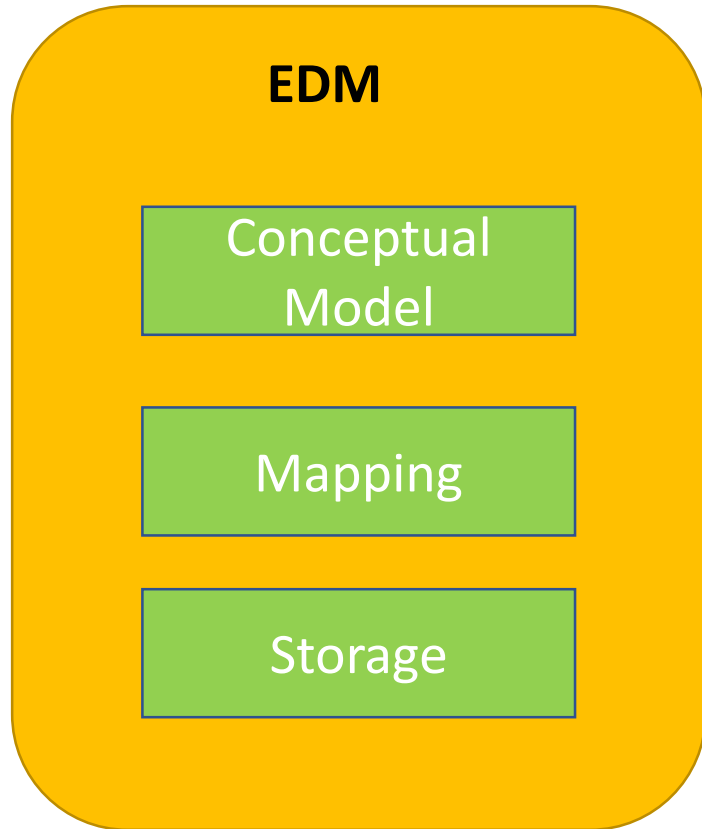
Mapping Objet Relationnel avec Entity Framework

Le Mapping peut se faire dans les deux sens :

- De l'application vers la base de données : Il va générer automatiquement les requêtes SQL nécessaire pour créer les tables de la base de données.
- De la base de données vers l'application : Il va créer automatiquement les entités du Model basées sur les tables de la base de données



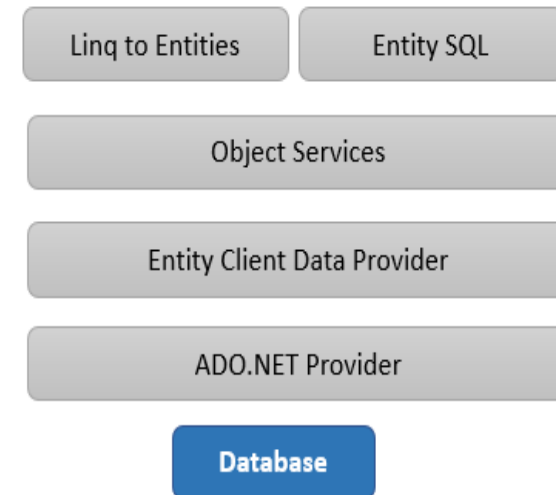
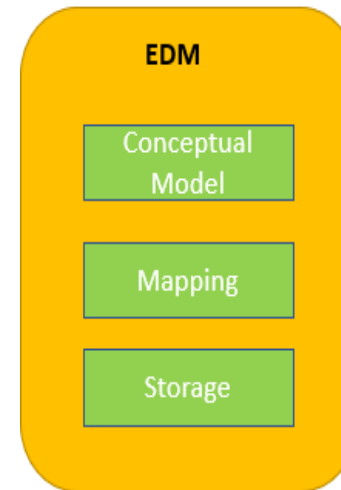
Architecture



Architecture

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

- ***Model conceptuel*** : Contient ou décrit les entités et leurs relations
- ***Model de stockage*** : Représente le modèle de conception de la base de données qui comprend les tables , les clés, les procédures stockées et leurs relations
- ***Model de Mapping*** : Contient les informations sur la façon dont le modèle conceptuel est mappé au modèle de stockage



Architecture

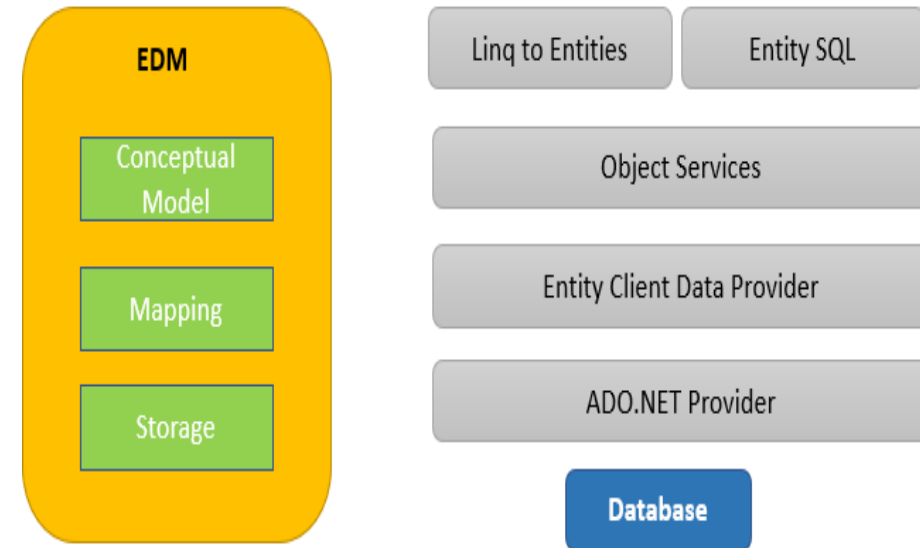
- Il existe deux façons d'écrire des requêtes avec Entity Framework :

Linq to Entities :

- C'est un langage de requête utilisé pour écrire les requêtes sur les entités (les objets).
- Il retourne et interroge les entités qui sont définies dans le modèle conceptuel

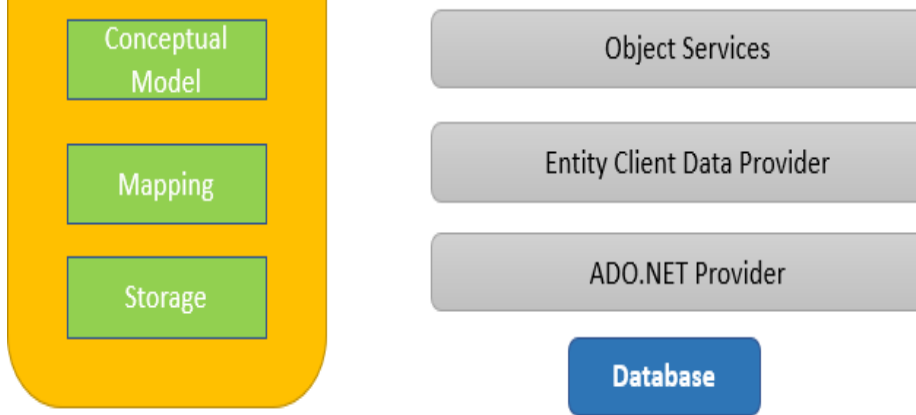
Entity SQL :

- C'est un autre langage de requête tout comme Linq to Entities
- Par contre c'est un langage de type SQL qui nous permet d'interroger des modèles conceptuels.
- Les modèles conceptuels représentent les données sous forme d'entités, et Entity SQL nous permet d'interroger ces entités et leurs relations dans un format similaire à SQL.



Architecture

Object Services :

- La couche Object services est le point d'entrée pour accéder aux données de la base de données et ensuite les retourner.
 - Elle représente le contexte de l'objet (entité)
- 
- The diagram illustrates the Entity Framework architecture. On the left, a yellow rounded rectangle labeled 'EDM' contains three green rounded rectangles stacked vertically: 'Conceptual Model', 'Mapping', and 'Storage'. To the right of the EDM layer is a stack of five gray rounded rectangles: 'Linq to Entities', 'Entity SQL', 'Object Services', 'Entity Client Data Provider', and 'ADO.NET Provider'. Below these is a blue rounded rectangle labeled 'Database'. Arrows indicate the flow of data and operations between these components.
- L'utilisation principal du **contexte** est d'effectuer différentes opérations telles que l'ajout, la mise à jour, et la suppression vers la base de données
 - Ce service permet au développeur d'utiliser certaines des fonctionnalités des ORM telles que le suivi des modifications, le chargement paresseux (lazy loading) etc. en écrivant des requêtes à l'aide de LINQ ou Entity SQL.

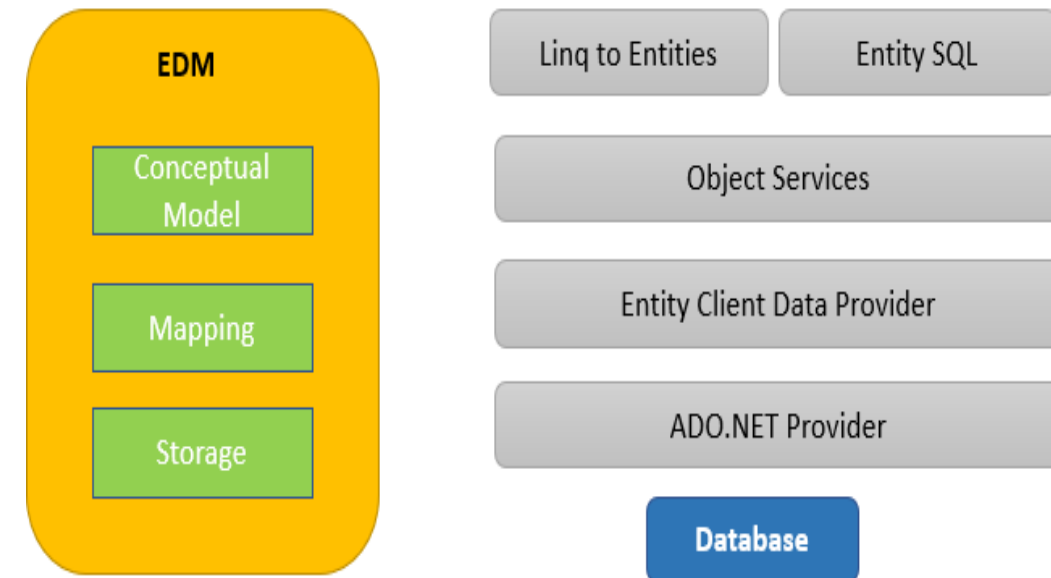
Architecture

Entity Client Data Provider :

- Cette couche convertit les requêtes LINQ-to-Entities ou Entities SQL en une requête SQL
- Il communique avec le fournisseur de données ADO.NET qui à son tour envoie ou récupère les données de la base de données.

ADO.NET provider :

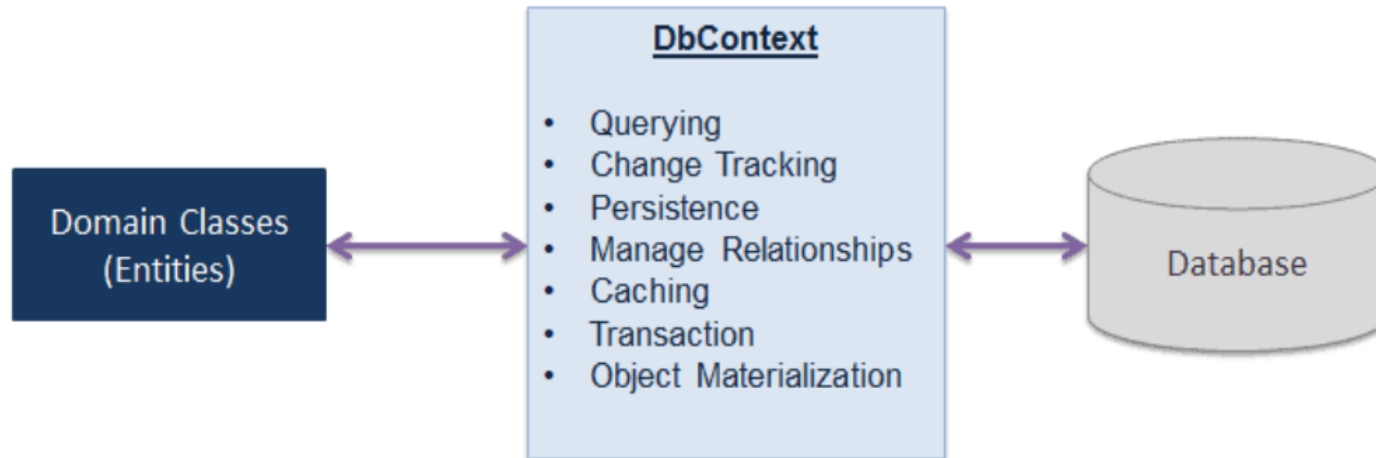
- Cette couche communique avec la base de données à l'aide du standard/Norme ADO.NET



Contexte de persistance

■ Définition :

- C'est une classe qui dérive de **DbContext** et qui regroupe l'ensemble des entités Managées.
- Le contexte fournit les fonctionnalités permettant de suivre les modifications, et de gérer les identités, l'accès concurrentiel et les relations.
- C'est un pont entre nos classes de domaine (Entités) et la base de données.



Contexte de persistance

- **EntityState (L'état des entités) :**
 - Les entités Managées ont un état dans le Contexte
 - Le contexte contient non seulement les objets référence de tous les (entités) dès qu'ils sont extraits de la base de données, mais conserve également une trace **des états de l'entité** et les modifications apportées aux propriétés de l'entité. Cette fonctionnalité est connue sous le nom de **Change Tracking** (suivi des modifications).
 - L'état de l'entité est représenté par une énumération **System.Data.Entity.EntityState** dans EF 6

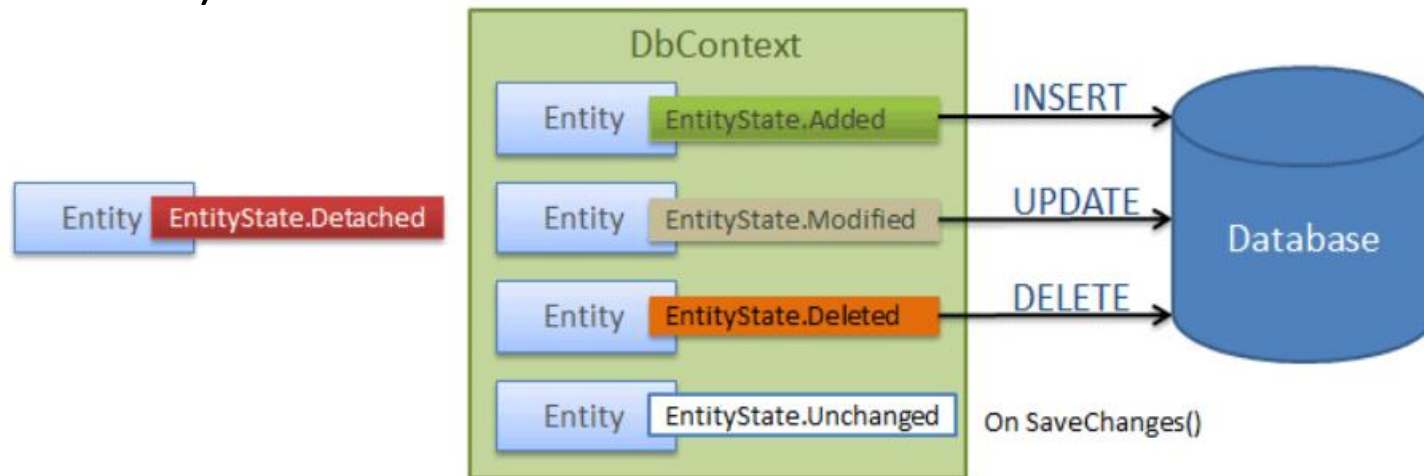
Contexte de persistance

■ EntityState (L'état des entités) :

L'API EF crée et exécute les commandes INSERT, UPDATE et DELETE en fonction de l'état d'une entité lorsque la méthode **context.SaveChanges()** est appelée.

Il exécute la commande INSERT pour les entités à l'état **Added**, la commande UPDATE pour les entités à l'état **Modified** et la commande DELETE pour les entités à l'état **Deleted**.

Le contexte ne suit pas les entités dans l'état **Détaché** (L'objet est créé mais n'est pas attaché au contexte)



Contexte de persistance

■ EntityState (L'état des entités) :

Etat	Description
Added	L'objet vient d'être créé et a été ajouté au contexte. La méthode SaveChanges() n'a pas été appelée.
Modified	L'une des propriétés de l'objet a été modifiée et la méthode SaveChanges() n'a pas été appelée
Deleted	L'objet a été supprimé du contexte. Une fois les modifications enregistrées, l'état de l'objet passe à Detached.
Detached	<ul style="list-style-type: none">• L'objet existe, vient d'être créé mais n'est pas suivi par le contexte• Une entité est également dans cet état après avoir été supprimée du contexte en appelant la méthode Detach(Object)
Unchanged	L'objet n'a pas été modifié depuis son attachement au contexte ou depuis le dernier appel de la méthode SaveChanges().

Contexte de persistance

■ Méthodes DbContext:

Méthode	Description
Entry	La méthode Entry nous permet de visualiser les entités suivies, d'avoir des informations sur l'entité par exemple son état.
SaveChanges « Sauvegarder les modifications »	Exécute les commandes INSERT, UPDATE et DELETE dans la base de données pour les entités ayant pour état Added, Modified et Deleted.
SaveChangesAsync	Méthode asynchrone de SaveChanges ()
Set	Crée un DbSet<TEntity> qui peut être utilisé pour interroger et enregistrer des instances TEntity (Objet)
OnModelCreating	Un exemple d'utilisation : Mapper les entités avec l'API Fluent

Contexte de persistance

```
public class MyDbContext : DbContext
{
    public MyDbContext() : base("chCnx1") //nom de la chaine de connexion
    {
        //PROPRIETES MODIFIABLES
        //ChangeTracker : Fournit l'accès aux fonctionnalités du contexte relatives au
            suivi des modifications d'entités.
        //Configuration : Fournit l'accès aux options de configuration pour le contexte.
        //Database : Crée une instance de base de données pour ce contexte et vous permet
            de créer, supprimer ou vérifier l'existence de la base de données sous-jacente.
    }

    // les DbSet pris en charge par EF
    //DBSet = objet représentant la collection d'entités persistées
    public DbSet<Client> Clients { get; set; }

    //Méthode pour définir des critères d'association
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        //base.OnModelCreating(modelBuilder);
    }
}
```

Contexte de persistance

- **DbSet :**

- La classe DbSet représente un ensemble d'entités qui peut être utilisé pour les opérations de création, de lecture, de mise à jour et de suppression (CRUD)
- La **classe de contexte** (dérivée de **DbContext**) doit inclure les propriétés de type DbSet des entités mappées aux tables de la base de données

Contexte de persistance

■ **DbSet et DbContext :**

- Un DbContext correspond à notre base de données (ou à un ensemble de tables alors qu'un DbSet correspond à une table de votre base de données. Il est donc parfaitement logique que vous obteniez une combinaison des deux
- Vous utiliserez un objet DbContext pour accéder à vos tables (qui seront représentées par un DbSet) et vous utiliserez vos DbSet pour accéder, créer, mettre à jour, supprimer et modifier les données de votre table.

Contexte de persistance

■ DbSet :

Méthode	Description
Add	Ajoute l'entité donnée au contexte. Cette entité aura comme état Added.
Attach(Entity)	Contrairement à la méthode Add, elle attache des entités qui existe déjà dans la base de données. Plutôt que de définir EntityState sur Added, Attach génère un EntityState Unchanged , ce qui signifie qu'il n'a pas changé depuis qu'il a été attaché au contexte.
Create	Permet de créer nouvelles entités en tant que proxys. Cette méthode renvoie une nouvelle instance qui n'est pas ajoutée ou attachée au contexte
Find(int)	Utilise la valeur de la clé primaire pour tenter de trouver une entité suivie par le contexte Si l'entité est introuvable dans le contexte, une requête est envoyée à la base de données pour y rechercher l'entité. Null est retourné si l'entité est introuvable dans le contexte ou dans la base de données.

Contexte de persistance

■ DbSet :

Méthode	Description
Remove	Marque l'entité donnée comme supprimée (état Deleted). L'entité doit exister dans le contexte dans un autre état avant que cette méthode ne soit appelée.
AsNoTracking<Entity>	Renvoie une nouvelle requête dans laquelle les entités renvoyées ne seront pas mises en cache dans le DbContext Les entités renvoyées comme AsNoTracking ne seront pas suivies par le DbContext. Cela augmentera considérablement les performances des entités en lecture seule.
Include	Permet de charger une entité associée à une autre entité Permet d'utiliser le chargement tardif (LazyLoading) et le chargement immédiat (Eager Loading)

Approche de développement

Avec Entity Framework, il existe trois approches différents qu'on peut utiliser lors du développement nos applications :

Code First :

- On écrit d'abord les classes (le model) et on génère notre base de données à partir du modèle

Database First :

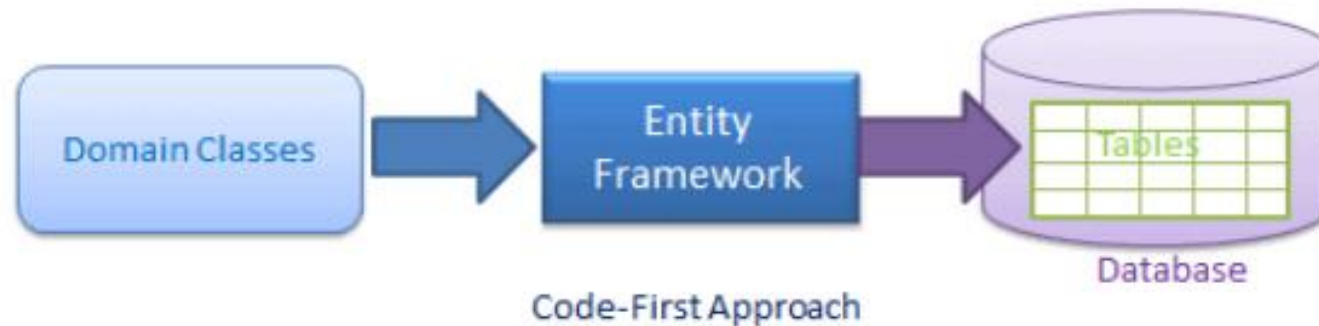
- On va d'abord créer la base de données. Et partir de celle-ci, on génère notre modèle

ModelFirst :

- On crée d'abord le modèle.
- A partir du modèle on génère les entités et les tables dans la bases de données.

Approche de développement

- Code First



On peut utiliser cette approche lorsqu'on dispose pas d'une base de données existante pour notre application

1. Dans l'approche code First, on commence par écrire nos entités (classe de domaine)
2. L'approche Code-First nécessite un contexte qui dérive de la classe DbContext.
3. La base de données est créée à partir de ces classes à l'aide des commandes de migration.

Approche de développement

■ Migration

Les migrations permettent de mettre à jour la structure de la BDD après des changements sur le modèle

Les commandes de migrations :

- **Activer les migrations** (à exécuter qu'une seule fois dans le projet) :
 - Exécuter la commande « **enable-migrations** » dans la Console du Gestionnaire de Package
 - **Aide** : Dans le menu Outils, sélectionnez Gestionnaire de package NuGet> Console du Gestionnaire de package
 - Cette commande a ajouté un dossier Migrations à notre projet. Il contient un fichier (une classe) nommé « **configuration.cs** »

Approche de développement

■ Migration

Les commandes de migrations (suite) :

- **Activer les migrations** (suite) :
 - Le fichier **configuration.cs** contient une méthode « **Seed** » : L'objectif de la méthode « Seed » est de nous permettre d'insérer ou de mettre à jour des données de test. La méthode est appelée lors de la création de la base de données et chaque fois que la base de données est mise à jour après une modification du modele (des classes).

Approche de développement

■ Migration

Les commandes de migrations (suite) :

- A chaque mise à jour des entités :
 - Utiliser la commande « `add-migration nomMigration` »
 - Cette commande crée une nouvelle classe avec le nom spécifié (ex : `215415454_nomMigration.cs`)
 - Cette classe contient deux méthodes :
Up() : Elle met à jour la base de données, de son état actuel représenté par la migration précédente vers l'état attendu représenté par la migration actuelle (en cours).

Approche de développement

■ Migration

Les commandes de migrations (suite) :

- A chaque mise à jour des entités (suite) :

Down() : Elle effectue l'opération inverse.

Elle supprime toutes les modifications de la migration actuelle et rétablit la base de données vers l'état où elle se trouvait (représenté par la migration précédente)

Cette commande exécute le script Down des migrations

- **Update-Database –TargetMigration: NomMigration**
 - Cette commande exécute le script Down des migrations
 - Permet de revenir à une migration précédente
- Pour restaurer une base de données vide, utilisez la commande :
 - **Update-Database –TargetMigration: \$InitialDatabase**

Approche de développement

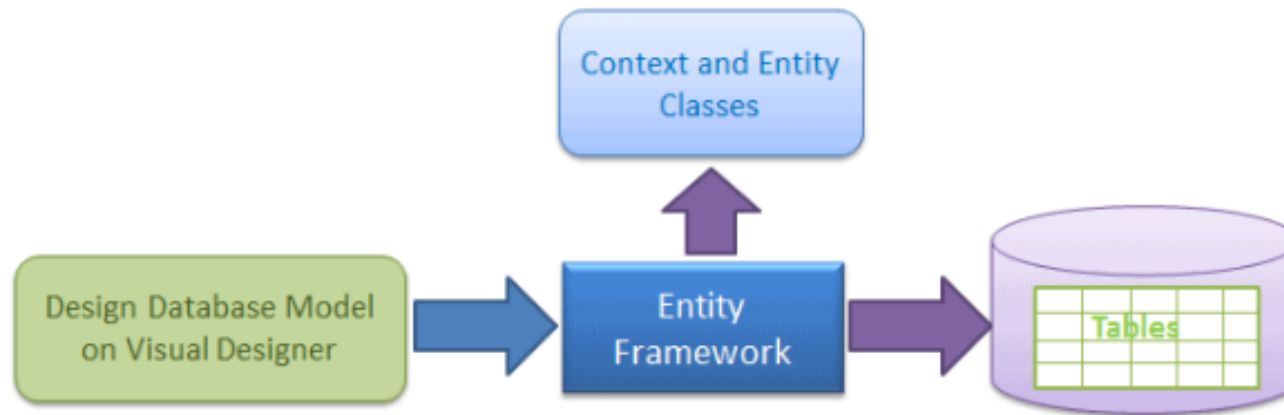
■ Migration

Les commandes de migrations (suite) :

- Applique les migrations en attente à la base de données (mise à niveau vers la migration la plus récente)
 - Utiliser la commande « **update-database** »
 - Elle exécute le dernier fichier de migration créé par la commande « add-migration »

Approche de développement

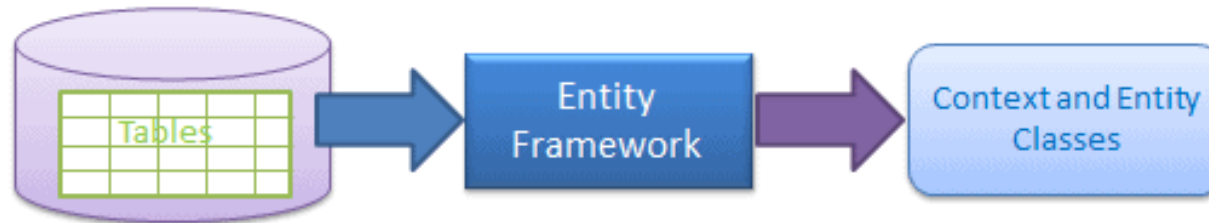
- Model First



Le model est créé avec le concepteur visuel intégré à Visual Studio puis génère les entités, la classe contexte et le script de la base de données à partir de notre modèle.

Approche de développement

- Database First



- L'approche permet de concevoir un modèle à partir d'une base de données existante.
- Le modèle est stocké dans un fichier EDMX (extension EDMX) et peut être consulté et modifié dans le Concepteur Entity Framework.
- Les classes avec lesquelles vous interagissez dans votre application sont générées automatiquement à partir du fichier EDMX.