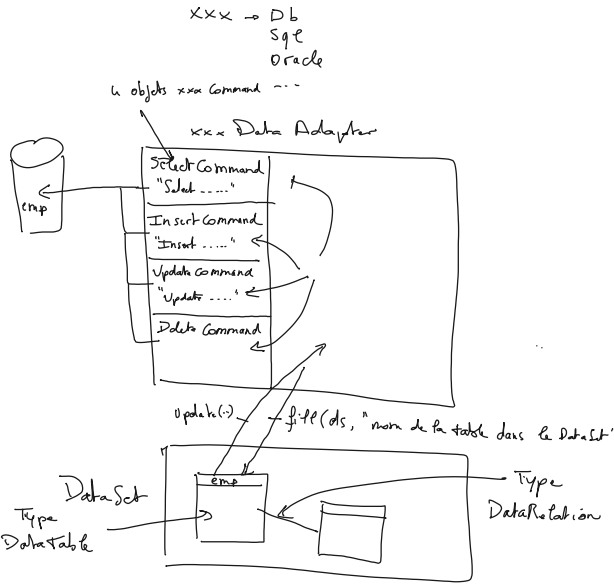
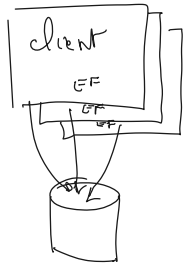



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace tests
{
    // class avec méthode d'extension :
    public static class MesExtensions
    {
        public static string mul(this string me, int v)
        {
            return (Convert.ToInt32(me) * v).ToString();
        }
    }

    delegate int MyDelegate(string msg);

    class Program
    {
        static int m(string msg)
        {
            Console.WriteLine("ok : "+msg);
            return 0;
        }
        static int m2(string msg)
        {
            Console.WriteLine("ok2 : " + msg);
```

```
            return 0;
        }

        static void test(MyDelegate m)
        {
            Console.WriteLine(m("50"));
        }

        static void Main(string[] args)
        {
            test(e => Convert.ToInt32(e) * 100);

            MyDelegate mdel = null; // pointera sur 1..n méthodes de la même signature
            mdel += new tests.MyDelegate(m2);    // attention à ne pas mettre () après la
    méthode
            mdel += new tests.MyDelegate(m2);    // attention à ne pas mettre () après la
    méthode
            mdel += new tests.MyDelegate(m);    // attention à ne pas mettre () après la méthode

            // ok3 via methode anonyme
            mdel += new tests.MyDelegate(
              delegate (string msg)
              {
                Console.WriteLine("ok3 : " + msg);
                return 0;
              }
            );

            // ok4 via methode Lambda
            mdel += ( (msg) =>
              {
                Console.WriteLine("ok4 : " + msg);
                return 0;
              }
            );
            // Expression Lambda = simplification de la methode Lambda pour laquelle il y a
    explicitement
            // un return de l'expression

            mdel += m => Convert.ToInt32(m)*2;   // la méthode renvoie 1
            // Génère :
            // mdel += (m) => { return 1; }

            var res = mdel("10");
            Console.WriteLine(res);

            List<int> li = new List<int>();
            li.Add(10);
            li.Add(7);
            li.Add(15);
            li.Add(3);

            foreach (var i in ( from c in li orderby c select c ))
                Console.WriteLine(i);

        }
    }
}
```

Client / serveur

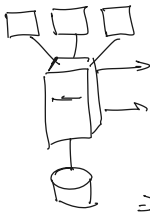


→ ADO.net → gère manuellement le cycle de la connexion

→ EF → cache
→ using → durée de vie

Appli Web

3 Tier :



→ Front

→ Metier / ORM

↑ Appli
n instances du contexte dans le même process

⇒ si haute dispo =
n SRV Web
↳ n Instances du Ctx dans n process

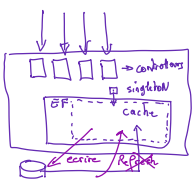
---

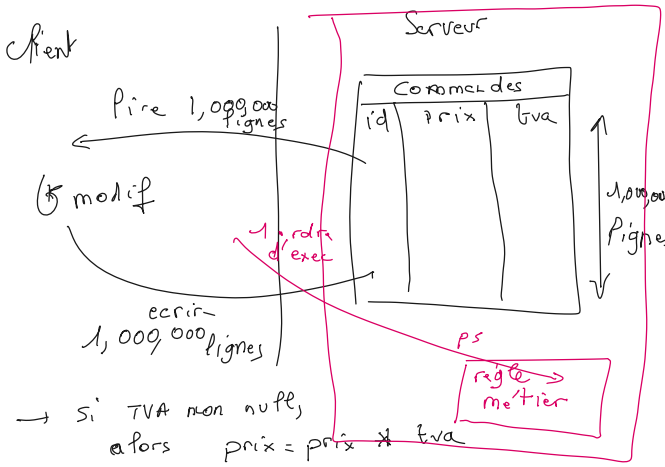
n Tier

→ navigateur

→ Front Web

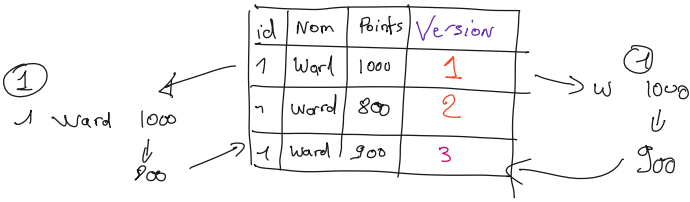
→ Business et ORM (mono, multi process)

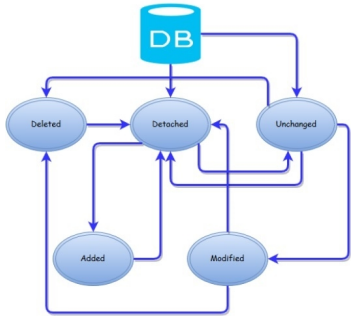




Conseils de performance Microsoft :
https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/performance-considerations#strategies-for-improving-performance

Multiple Version Concurrency Check





Design pattern "Unit of work" :
https://www.c-sharpcorner.com/UploadFile/b1df45/unit-of-work-in-repository-pattern/