

I sidste opgave skulle du selv lave et simpelt datalag. Og opgaven sluttede af med at du skulle lave Insert, Delete og Update funktioner....

Og som du sikkert har lagt mærke til er der tilføjet et ID felt til Person classen, som egentlig ikke har nogen reel mening. Men når man arbejder med større mængder ens data, giver det i mange tilfælde mening at indsætte et ID felt i hver enkelt post.

Fordelen ved dette er at når man skal finde en enkelt post i flere tusind, kan man nøjes med at kigge på denne ene værdi (Hvis den altså er det man kalder en unique identifier) frem for at skulle sammenligne samtlige værdier i hver post med en reference.

Og specielt i vores delete og update funktioner hvor vi som regel kun skal påvirke et enkelt element i listen giver det mening at benytte en unique identifier. - kigger du lidt nærmere på tabellerne i en rigtig database, vil du også se at de fleste tabeller har unique identifiers, enten som en enkeltstående kolonne eller som en sammensat enhed af flere kolonner.

I vores lille liste bliver denne unique identifier kaldt ID. Vi har dog ikke bygget nogen sikkerhed ind, så der er ikke noget der forhindrer os i at lave flere poster i listen, med samme ID. Og i stedet for at lave kode der sikrer unikke ID'er vil vi i denne omgang, koncentrere os om søgningerne i listen...

Men princippet med at bruge ID som en unique identifier, er stadig det samme.

Jeg gav et par hints, til hvorledes det kunne løses med en for løkke samt remove og insert funktionerne på listen. Og gætter på at delete og insert funktionerne var forholdsvis nemme?

Herunder er vist hvorledes Delete kunne være lavet, med en for løkke:

```
public int Delete(Person person)
{
    int returværdi = 0;

    for (int i = _publicListe.Count-1; i > 0; i--)
    {
        if (_publicListe[i].ID == person.ID)
        {
            _publicListe.RemoveAt(i);
            returværdi++;
        }
    }
    return returværdi;
}
```

Og hvad med Update?

Jeg foreslog at du brugte en for løkke til at rende igennem listen med. Og i Delete funktionens tilfælde er det faktisk det nemmeste.

I update funktionen ser der lidt anderledes ud. Her kan vi nemlig også gøre det med en foreach løkke, som vist herunder.

```
public int Update(int ID, string Fornavn, string Efternavn, int Formue)
{
    // Vi laver en returværdi, så hvis vi ikke finder noget, returnerer vi 0
    int returværdi = 0;

    //Løber igennem listen en efter en, for at sammenligne ID
    foreach (Person p in _publicListe)
    {
        if (p.ID == ID)
        {
            p.Fornavn = Fornavn;
            p.Efternavn = Efternavn;
            p.Formue = Formue;
            returværdi++;
        }
    }
    return returværdi;
}
```

Fælles for dem begge er at de løber hele listen igennem, uanset om de har fundet en ID der matcher eller ej. Noget der er nødvendigt da vi jo ikke har helt unikke ID's. Det er også grunden til at begge funktioner returnerer en int, så vi kan se hvor mange poster der blev påvirket af vores ændring. :)

LINQ

LINQ forespørgsler er en anden metode hvorved vi kan iterere igennem lister af data. LINQ bruger en sql lignende syntax direkte i C#. Det lyder måske avanceret, men det er forholdsvis lige til. Kig på eksemplet fra før:

```
foreach (Person p in _publicListe)
{
    if (p.ID == ID)
    {
        p.Fornavn = Fornavn;
        p.Efternavn = Efternavn;
        p.Formue = Formue;
        returværdi++;
    }
}
```

Her løber vi igennem en liste af typen Person.

Lille p er den aktuelle Person, og vi kan derfor sammenligne properties i p, med hvad vi nu har behov for.

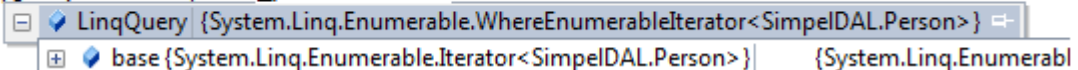
I en LINQ forespørgsel, der gør absolut det samme, kunne det se ud som herunder:

```
//Laver en LINQ forespørgsel, der kan finde den pågældende person i listen
var LinQuery = from p in _publicListe
                where p.ID == ID
                select p;

// Udfører forespørgslen
foreach (Person p in LinQuery.ToList())
{
    p.Fornavn = Fornavn;
    p.Efternavn = Efternavn;
    p.Formue = Formue;
    returværdi++;
}
```

Kigger vi lidt på hvad følgende linje kode gør, vil det vi en debug se således ud:

```
//Laver en LINQ forespørgsel, der kan finde den pågældende person i listen
var LinQuery = from p in _publicListe
```



The screenshot shows the Visual Studio IDE with a tooltip for the variable `LinQuery`. The tooltip indicates that `LinQuery` is of type `System.Linq.Enumerable.WhereEnumerableIterator<SimpelDAL.Person>`. Below this, it shows the base type `base {System.Linq.Enumerable.Iterator<SimpelDAL.Person>}` and the interface `{System.Linq.Enumerabl`.

Herover kan du se at det der returneres i bund og grund det der kaldes en Enumerable Iterator. Og en Enumerable Iterator bruges til at gennemløbe data lister af forskellig art.

I dette tilfælde kan den løbe igennem vores Person Liste og finde det vi har specificeret i vores forespørgsel. - de poster hvor "Aktuelle person).ID er == ID.

Og det er igen vigtigt at ligge mærke til at den rent faktisk kan finde flere, hvis der er flere der har samme id i listen! Derfor render vi igennem denne listen af fundne elementer og retter i dem. (hvis vi var sikker på at der kun var en, kunne vi havde brugt et kald til `LinqQuery.FirstOrDefault()`; for kun at returnere en enkelt post.

Det var en masse teori. Og hvorfor er det nu så fedt, når foreach i dette tilfælde virker lige så god og egentlig også fylder mindre???

Jow, LINQ er et universalt sprog, forstået på den måde, at det udover helt almindelige objekter (som det vi har gang i nu) også kan bruges direkte til SQL. Det kan bruges direkte til XML og ikke mindst kan det bruges direkte til Entity frameworket, som vi skal i gang senere.

Men udover at LINQ er brugbart i mange forskellige sammenhæng, kan det også rigtig mange andre fede ting, man ikke kan med en foreach løkke. Noget vi vender tilbage til i senere opgaver.

Men først skal vi have lavet om på vores Delete funktion, så den også benytter LINQ.

Kom med et forslag til hvorledes dette kan gøres, inden du går i gang med næste opgave: