# Oefening 1

# Doelstellingen:

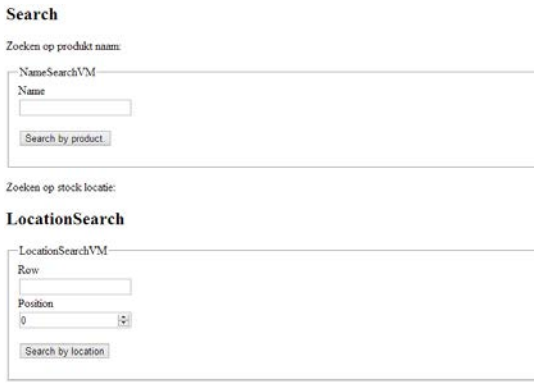
- Herhaling ASP.NET MVC semester 3
- Leren werken met Entity Framework 6 (EF6)
- Migreren van klassieke ASP.NET MVC toepassing naar EF6 Database code first

# Database First

De eerst oefening omvat het omzetten van een ASP.NET MVC web applicatie (gegeven door docenten) en deze aanpassen naar EF Database First. De toepassing die je krijgt bevat de klassieke DAL met SQL statements in de code. De bedoeling is dat je de applicatie aanpast zodat ze blijft gebruik maken van de database maar de toegang tot de database moet via EF 6 gebeuren.

## Opzetten project

Voor deze oefening maken we gebruik van een bestaande oefening uit semester 1. Open de solution **SalesWeb** uit het bronmateriaal. Indien nodig moet U ook nog een restore van de database doen in SQL Server. U kan de database ook terugvinden bij het bronmateriaal. Pas ook de connectionstring aan zodat deze verwijst naar de juiste SQL Server. Indien alles correct is geconfigureerd moet u volgende website zien verschijnen. Test ook de zoekfunctie om te zien of de connectie met de database inorde is.



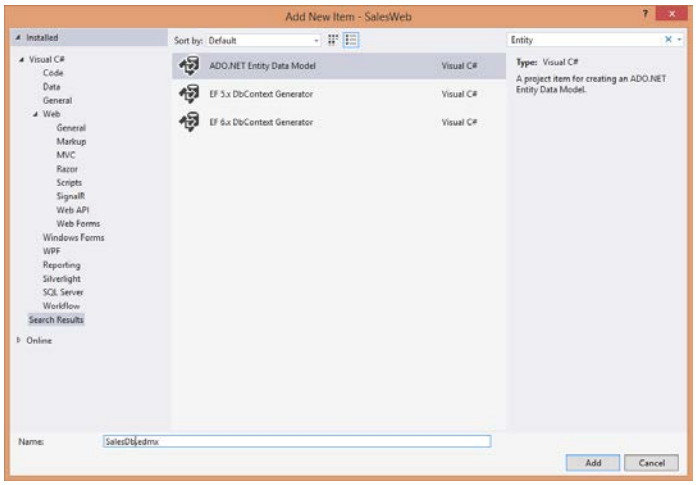

Resultaat na zoeken op naam “ML”

## Aanmaken modellen

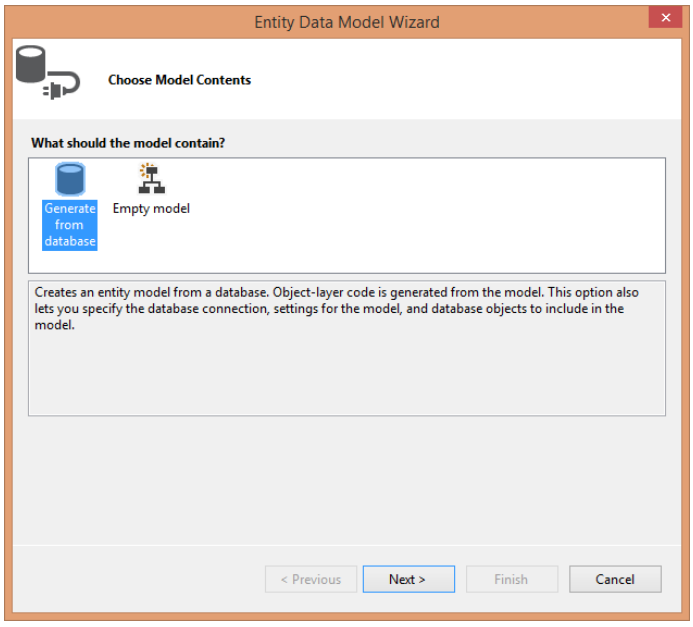
Als we kijken in de map Model zijn er al twee modellen aanwezig:

- Inventory
- Product

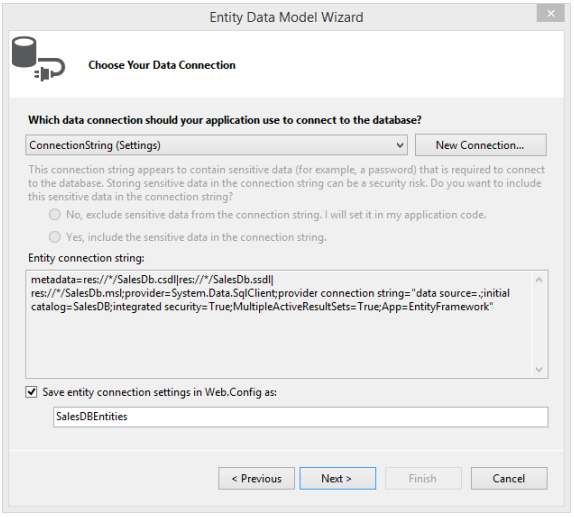
We verwijderen deze uit het project aangezien entity framework deze voor ons zal aanmaken. Zoals we gezien hebben in de theorie maakt entity framework gebruik van een context om te communiceren met de database. We gaan nu deze context genereren. Via rechtermuisknop op het project kiezen we “Add New Item” en zoeken achter “Entity”. Als naam kiezen we “SalesDb”.



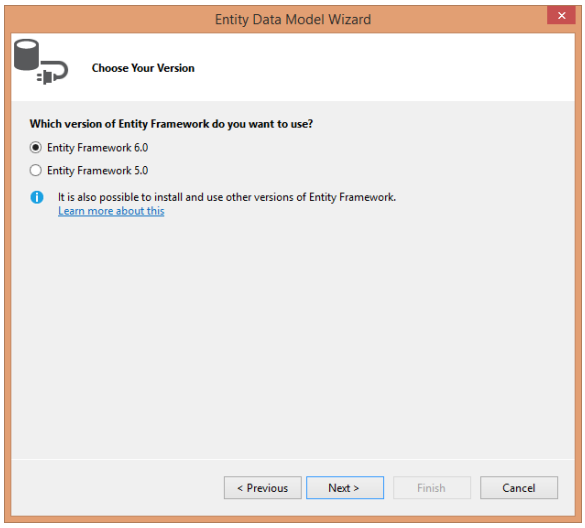

In de volgende stap kiezen we voor “Generate from database”.



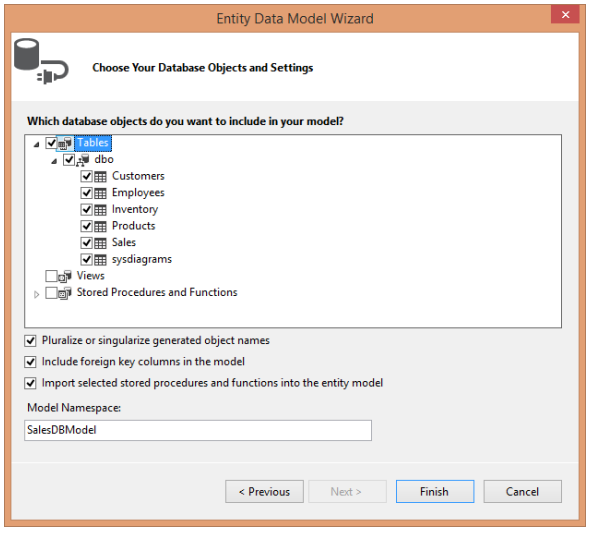

In deze stap zal hij een eigen connectionstring opstellen op basis van de reeds bestaande zodat de context kan communiceren met de database. De naam SalesDBEntities laten we staan.



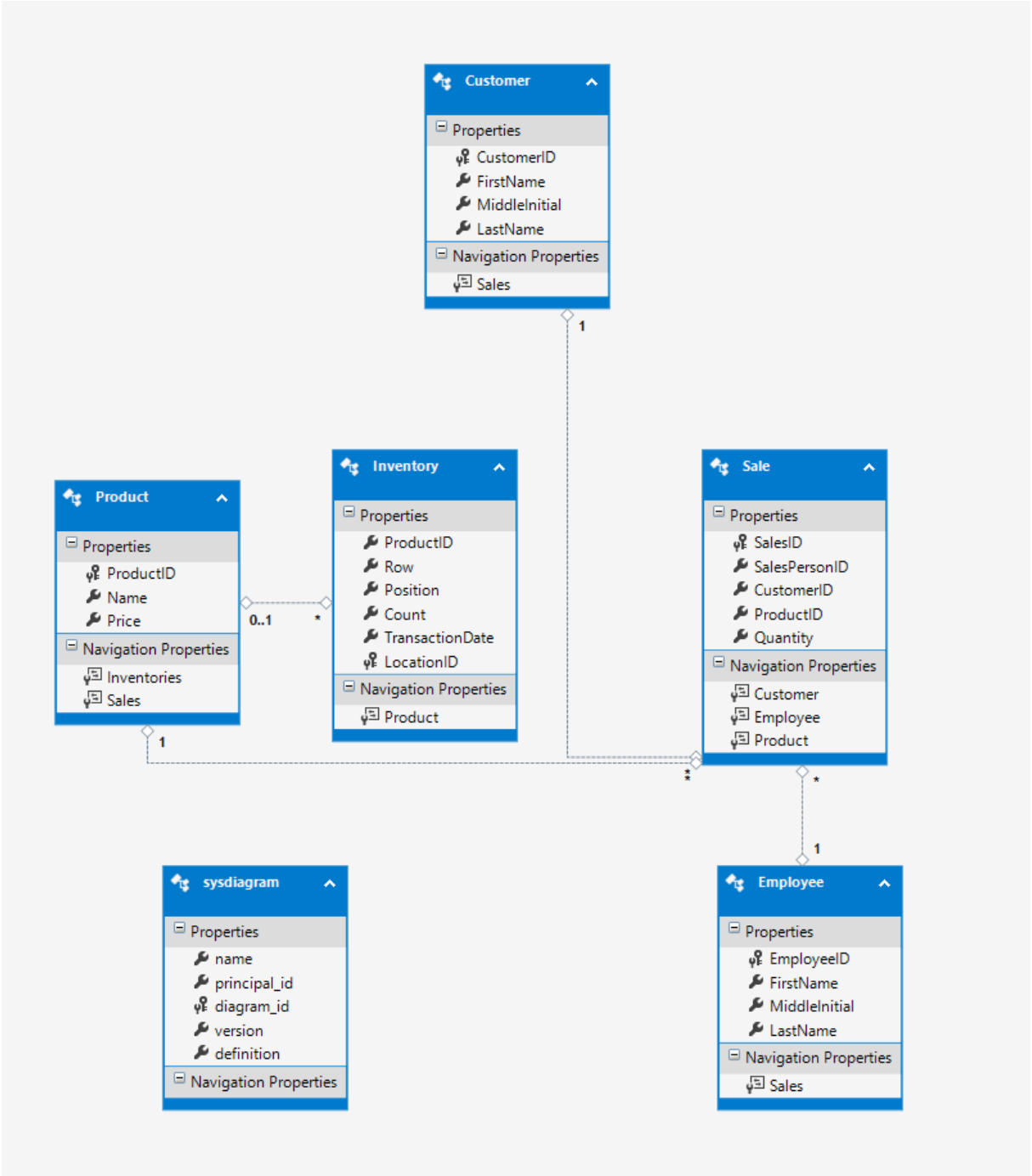

In de volgende stap kiezen we voor Entity Framework 6.0




In de daar op volgende stap moeten we aanduiden wat we wensen te genereren. We vinken alle tabellen aan. Daarnaast duiden we ook “Pluralize or singularize object names” aan. De andere twee vinkjes laten we aanstaan net als de namespace.




Na het klikken op finish zal er een volledig scheme weergegeven worden met relaties.




Als we nu compileren krijgen we normaal een aantal fouten. Er is een probleem met de namespace van Inventory en Product. Probeer zelf dit probleem op te lossen door de juiste namespace toe te voegen (tip: Resolve).

```
                }

                foreach (Inventory inventory in model.Inventories)
                {
                    inventory.Product = ProductRepository.FindById(inventory.ProductID);
                }


                return View("~/Views/Inventory/Index.cshtml", model.Inventories);
            }
            else
            {
                return View("Search", new SearchVM() { NameSearch = new NameSearchVM(), LocationSearch = model });
            }
        }


    }
}
```

100 %

Error List

2 Errors | 0 Warnings | 0 Messages | Search Error List

| | Description | File | Line | Column | Project |
|---|---|---|---|---|---|
| 2 | The type or namespace name 'Inventory' could not be found (are you missing a using directive or an assembly reference?) | SearchController.cs | 87 | 25 | SalesWeb |
| 1 | The type or namespace name 'Product' could not be found (are you missing a using directive or an assembly reference?) | SearchController.cs | 49 | 26 | SalesWeb |

Daarna zal er een nieuwe fout ontstaan (dit is normaal bij migreren van projecten). We lossen dit later op.

```
                foreach (Inventory inventory in model.Inventories)
                {
                    inventory.Product = ProductRepository.FindById(inventory.ProductID);
                }


                return View("~/Views/Inventory/Index.cshtml", model.Inventories);
            }
            else
            {
                return View("Search", new SearchVM() { NameSearch = new NameSearchVM(), LocationSearch = model });
            }
        }


    }
}
```

100 %

Error List

2 Errors | 0 Warnings | 0 Messages | Search Error List

| | Description | File | Line | Column | Project |
|---|---|---|---|---|---|
| 2 | Argument 1: cannot convert from 'int?' to 'int' | SearchController.cs | 90 | 68 | SalesWeb |
| 1 | The best overloaded method match for 'SalesWeb.Models.DAL.ProductRepository.FindById(int)' has some invalid arguments | SearchController.cs | 90 | 41 | SalesWeb |

## Wijzigen Repositories

In deze opgave bevatten de repositories SQL statements die we uitvoeren op de database.

```
4 references
public class ProductRepository
{

    //methoden
    1 reference
    public static List<Product> GetProducts()
    {
        //0.vars
        //1. SQL instructie
        string sSQL = "SELECT * FROM [Products]";

        //2.reading uitvoeren
        return GetList(sSQL);
    }


    2 references
    public static Product FindById(int productID)
    {
        //0.vars
        //1. SQL instructie
        string sSQL = "SELECT * FROM [Products]";
        sSQL += " WHERE [Products].[ProductID] = @productID";

        //2. SQL parameters
        DbParameter idPar = Database.AddParameter("@productID", productID);

        //3. Haal data op en controleer op null/lege velden
        return GetList(sSQL, idPar)[0];
    }
```

Deze SQL statements moeten we vervangen door LINQ queries. Met behulp van het Using statement maken we de context aan. Weet u nog waarom we “using” gebruiken ? Na het aanmaken van de context stellen we onze query op. Pas wanneer we aan het return statement komen zal de query worden uitgevoerd op de database. De methode ToList<T> zal een lijst terugkeren van producten. De methode Single<T> zal één object van het type T terugkeren of null. Pas nu zelf de andere statements aan. De SQL helpers mogen tevens ook verwijderd worden uit het project (staan onderaan in de repostitory).

```
//methoden
1 reference
public static List<Product> GetProducts() {
    using (SalesDBEntities context = new SalesDBEntities()) {

        var query = (from p in context.Products
                     select p);

        return query.ToList<Product>();
    }
}


2 references
public static Product FindById(int productID) {
    using (SalesDBEntities context = new SalesDBEntities()) {

        var query = (from p in context.Products
                     where p.ProductID == productID
                     select p);

        return query.Single<Product>();
    }
}
```

Als we klaar zijn met aanpassen dan zitten we nog steeds met onderstaande fout. In deze code gaan we voor iedere inventory de producten ophalen.

```
0 references
public ActionResult LocationSearch(LocationSearchVM model)
{
    if (ModelState.IsValid)
    {

       model.Inventories = InventoryRepository.FindInventoriesByLocation(model.Row, model.Position);
       if (model.Inventories == null)
        {
            return HttpNotFound();
        }

       if (model.Inventories.Count == 0)
       {

            ModelState.AddModelError("NoLocationData", "No data found for this location.");
            return View("Search", new SearchVM() { NameSearch = new NameSearchVM(), LocationSearch = model });

        }

        return View("~/Views/Inventory/Index.cshtml", model.Inventories);
    }
    else
    {
        return View("Search", new SearchVM() { NameSearch = new NameSearchVM(), LocationSearch = model });
    }
}
```

We mogen deze code echte verwijderen als we in onze repository bij de methode FindInventoriesByLocation een aanpassing doen.

```
//GET
0 references
public ActionResult LocationSearch(LocationSearchVM model)
{
    if (ModelState.IsValid)
    {

       model.Inventories = InventoryRepository.FindInventoriesByLocation(model.Row, model.Position);
       if (model.Inventories == null)
        {
            return HttpNotFound();
        }

       if (model.Inventories.Count == 0)
       {

            ModelState.AddModelError("NoLocationData", "No data found for this location.");
            return View("Search", new SearchVM() { NameSearch = new NameSearchVM(), LocationSearch = model });

        }

        return View("~/Views/Inventory/Index.cshtml", model.Inventories);
    }
    else
    {
        return View("Search", new SearchVM() { NameSearch = new NameSearchVM(), LocationSearch = model });
    }
}
```

We gaan nu expliciet opgeven dat entity framework direct bij het inladen ook de gerelateerde propertie “Product” mee zal inladen.

```
1 reference
public static List<Inventory> FindInventoriesByLocation(string row, int position) {
    using (SalesDBEntities context = new SalesDBEntities()) {
        var query = (from i in context.Inventories.Include(i => i.Product)
                     where i.Row == row && i.Position == position
                     select i);

        return query.ToList<Inventory>();
    }
}
```

## Aanpassen Views

Voor we kunnen testen moeten we nog een aantal kleine aanpassingen doen in de Views. Er zijn namelijk een aantal namespaces gewijzigd. Controleer alle views en kijk bij de @model of de namespaces nog correct is:

```
@model IEnumerable<SalesWeb.Models.Inventory>
```

Moet worden

```
@model IEnumerable<SalesWeb.Inventory>
```

In die view NameSearchIndex.cshtml kunnen we nu de tweede for each die de inventories afprint als volgt aanpassen:

```
@foreach (var item in Model.Products) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
        <td>
            @if (item.Inventories != null) {


                foreach (var inv in item.Inventories) {
                    @inv.Count.ToString()  <text> pieces  on </text>
                    @inv.Row.ToString()
                    @inv.Position.ToString();
                }
            }
```

Als laatste verwijderen we nog de referentie naar de DbHelper.dll. Daarna kan je de applicatie testen en zou deze moeten werken.