Server side Web Development

Lecture 6

ORM and Dbcontext

Outline

- About the database
  - Installing SQL Server 2014
  - Using the sqlserver manager
  - Running a database script
- Using ORM
  - More about Entity Framework
  - Mapping of Entity classes to the View model classes
    - Manually
    - Using the tool Automapper
- Querying the database
  - Dbcontext

Installing SQL Server 2014

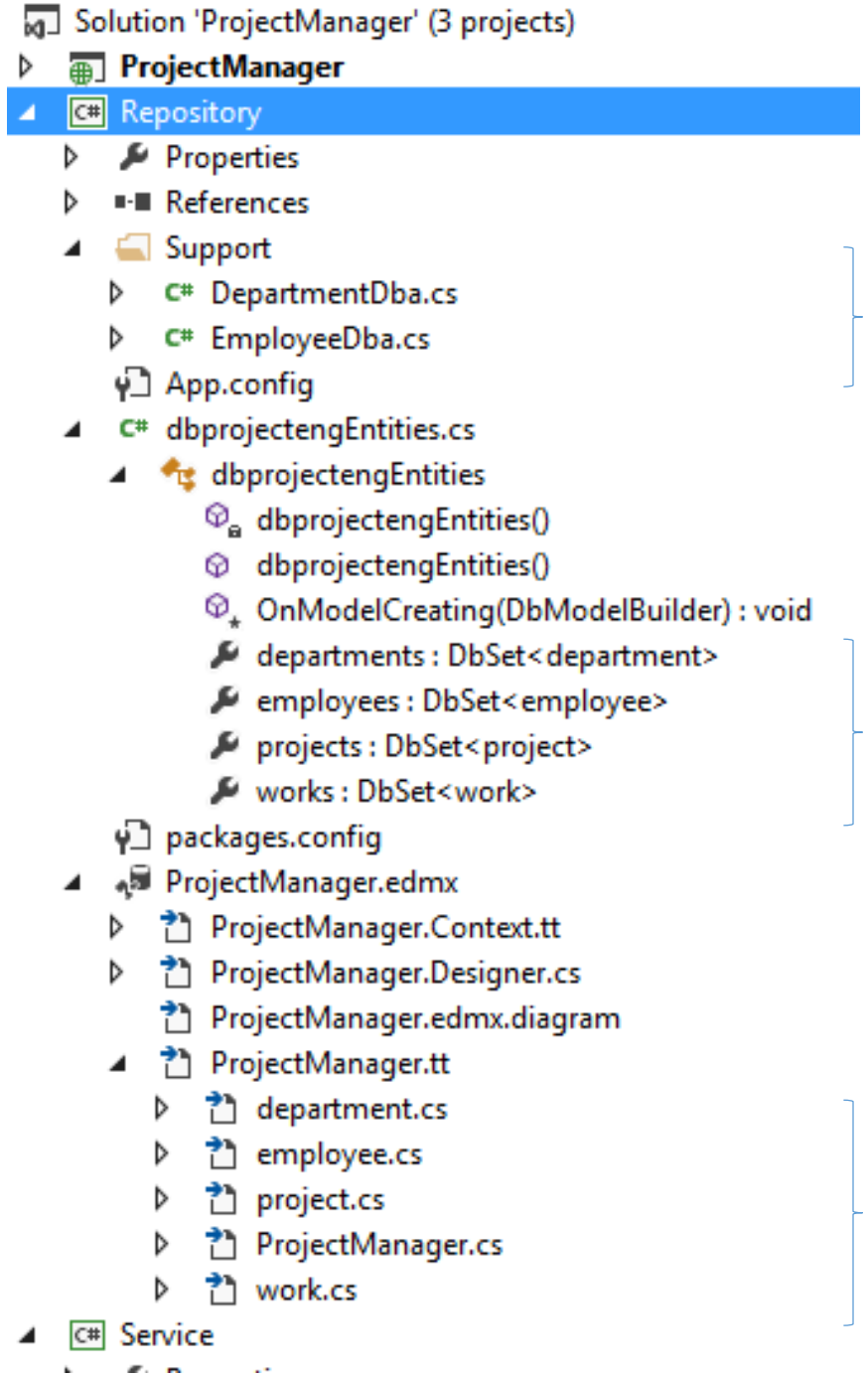
- Run installation video

  https://www.youtube.com/watch?v=E_zFM7mzFUg

More on using Entity FrameWork as ORM

- We will look more closely on what has been installed

After installation of EF



Code for CRUD operations using DbContext

Supports generating List based on the entity classes

Entity Classes defining how EF interprets the tables

Entity classes

Constructor provides lists for the rows referenced in other tables

Navigating the data is achieved easier and the query languages do no require to specify joins between tables

Automatically provided by EF

```
//
// <auto-generated>
//     This code was generated from a template.
//
//     Manual changes to this file may cause unexpected behavior in your application.
//     Manual changes to this file will be overwritten if the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Repository
{
    using System;
    using System.Collections.Generic;

    public partial class department
    {
        public department()
        {
            this.employees = new HashSet<employee>();
            this.projects = new HashSet<project>();
        }

        public int depid { get; set; }
        public string name { get; set; }
        public Nullable<int> bossid { get; set; }

        public virtual employee employee { get; set; }
        public virtual ICollection<employee> employees { get; set; }
        public virtual ICollection<project> projects { get; set; }
    }
}
```

The .edmx file

- This is a file describing the entity classes, shows a graphical diagram
- Written in three XML files With extension
  - .csdl for the conceptual schema
  - .ssdl for describing the storage schema
  - .msl for describing the mapping between schemas
- Also describes the relationships between the classes
- Each entity class and relationship has a property window
- Properties can be modified, but not advisable
  - E.g. adjusting the multiplicities of a relationship might render errors

Querying the conceptual model

- EF provides an API called DbContext
- Installing EF creates a class derived from DbContext
- To query the database an instance of this class is required
- Queries can be formulated using LINQ
  - Language Integrated Query
- Usually LINQ – queries are embedded in methods, which then are called by the service layer
  - In next slides the examples are from Project Manager case

List query example – alternative 1

```
public List<department> List()
{
    using(var db = new dbprojectengEntities())
    {
        return db.departments.ToList();
    }
}
```

Creates an instance of the projects class, derived from DbContext

Creates and returns a new list object

The using statement ensures that the instance is disposed

List query example – alternative 2

Alternative writing:

```
var query = db.departments.OrderBy(x => x.name);
```

```
public List<department> List()
{
    using(var db = new dbprojectengEntities())
    {
        var query = from dep in db.departments orderby dep.name select dep;

        return query.ToList();
    }
}
```

Returns a new list object

Read query example

```
public department Read(int id)  //Finds a particular "department"
        {
            using (var db = new dbprojectengEntities())
            {
                return db.departments.Find(id);

                // var query = from dep in db.departments where (dep.depid == id) select dep;

                // var query = db.departments.Where(x => x.depid == id);

                //return query.SingleOrDefault();
            }
        }
```

"Find" searches the list based on the key-attribute

Two alternative ways of writing the query with LINQ expressions

"SingleOorDefault" casts the result from the query to the correct return type, and return null if the result contains more than one object

Add query example (i.e. Insert in SQL)

```
public void Add(department depObject)
{
    using (var db = new dbprojectengEntities())
    {
      db.departments.Add(depObject);
      db.SaveChanges();
    }
}
```

Adds the change locally to a transaction to be run

"SaveChanges" runs the transaction
The query is formulated as an sql insert query by EF

Update query example

```
public void Update(department depObject)
{
    using (var db = new dbprojectengEntities())
    {
        db.departments.Attach(depObject);
        db.Entry(depObject).State = EntityState.Modified;
        db.SaveChanges();
    }
}
```

Finds the object to update in the database
Adds the updated object locally to a transaction to be run

Sets the status of the local object to "Modified"

Runs the transaction
The query is formulated as an sql update query by EF

Remove query example (i.e. Delete in SQL)

```
public void Delete(department depObject)
{
  using (var db = new dbprojectengEntities())
  {
    department depItem = db.departments.Find(depObject.depid);
    db.departments.Remove(depItem);
    db.SaveChanges();
  }
}
```

Finds the object to delete in the database and loads the object, to delete, locally to a transaction to be run

Removes the object locally and sets the status of the local object to "Deleted"

Runs the transaction
The query is formulated as an sql delete query by EF

Local vs the real database

- Queries using DbContext can be run locally
  - This means it uses already loaded data, if possible
  - Pros:
    - It goes faster to work with loaded data
    - It is possible to make several changes before applying them to the database
    - It is possible to run queries even if the application is detached from the database
  - Cons:
    - The real data in database may have changed from what is locally available
    - Might cause inconsistencies in database

Querying local data

- It is possible to formulate a query to work locally
  - E.g. counting the number of rows in the departments list loaded locally

```
var query = db.departments.Local.Count;
```

The property “Local” tells the query to only access the data loaded locally

Forcing the query to operate on database

- DbContext has a Load method that forces the data to be loaded before the query is executed
  - E.g. loading the departments list from database to the local memory (before actually running the query)

```
db.departments.Load();
```

  - It is also possible to load the data when running the query

```
var query = from emp in db.employees where emp.salary > 50000 select emp;

query.Load();
```

Using transactions

- It is possible to run the query (queries) in a transaction
  - This is especially important if several queries is needed to be run and you want no interference from other users
  - E.g. adding a new department and calculating the new depid

```
public void Add(department depObject)
{
    using (var db = new dbprojectengEntities())
    {
        using (var transaction = db.Database.BeginTransaction()) // Start a transaction
        {
            depObject.depid = (db.departments.ToList().Max(x => x.depid) + 1);  //Retrieve next depid
            db.departments.Add(depObject);  // Prepare query
            db.SaveChanges();   // Run the query
            transaction.Commit();   //  Permanent the result, writing to disc and closing transaction
        }
    }
}
```

Mapping entity and view models

- The entity classes are different from view model classes
- It is necessary to map data loaded in objects created from entity classes to objects created from view model classes
- Writing code for this manually is elaborate and tedious
- Example of such mapping for the Project Manager follows

Example from Project Manager application

- Fetching department information

```
static public Department getDepartment(int depId)
{
    DepartmentDba depDbaObj = new DepartmentDba(depId);
    Department depObj = new Department();
    depObj.depid = depDbaObj.DepartmentObj.depid;
    depObj.bossid = (int) depDbaObj.DepartmentObj.bossid;
    depObj.name = depDbaObj.DepartmentObj.name;
    return depObj;
}
```

Creates an instance of the support classes for retrieving data from database

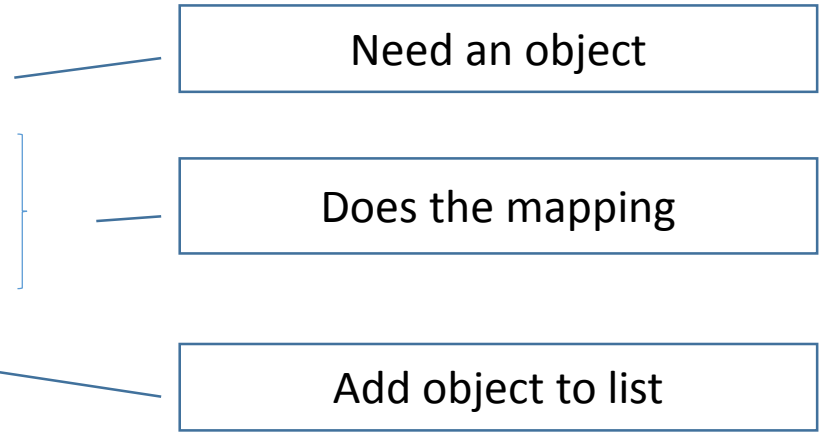
Creates an instance of the destination class

Does the mapping

Mapping for list objects

```
static public List<Department> getDepartments()
{
    DepartmentDba depDbaObj = new DepartmentDba();
    List<Department> depObjList = new List<Department>();
    foreach (var elem in depDbaObj.List())
    {
        Department depObj = new Department();
        depObj.depid = (int) elem.depid;
        depObj.bossid = (int) elem.bossid;
        depObj.name = (string) elem.name;
        depObjList.Add(depObj);
    }

    return depObjList;
}
```



The magic solving this…

- Example Automapper

- This is a tool that automatically solve the tedious work
  - Well it needs a configuration…
    - (that might take some day or two to figure out how it works)
  - … To know what to do

Mapping tool - Automapper

- Automapper installation
  - Rightclick ”References” in the ”Service” project
  - Click ”Manage NuGet packages”
  - Find Autmapper and install
    - A reference to ”Automapper” should come upp
- In service project
  - Add folder ”Configuration” and class ”AutomapperConfig.cs”
- In Global.asax.sc
  - Add the following line:
    Service.Configuration.AutoMapperConfig.Configure();

Configuring Automapper

- The AutoMapperConfig Class
  - Is used for configuring how objects of entity classes should be matched to view classes
  - Matching uses name matching of items inside the objects
  - Rules can also be added for specific item to item mapping
- Needs appropriate using statements
  - To Automapper (using AutoMapper;)
  - To repository (e.g. using Repository;)
  - To service models (e.g. using Service.Models;)

Configuring Automapper

- Add a public static method ”Configure”
  - This is a method run at application start
  - Can be something like this

```
public static class AutoMapperConfig
{
    public static void Configure()
    {
        Mapper.Initialize(cfg =>
        {
          cfg.AddProfile(new ToDepartmentProfile());
          cfg.AddProfile(new FromDepartmentProfile());
          cfg.AddProfile(new ToEmployeeProfile());
          cfg.AddProfile(new FromEmployeeProfile());
        });
    }
}
```

Run the Initialize method

Takes as argument objects each containing a mapping instruction

The Profile

- Profile is a class from AutoMapper
- Create for each mapping instruction a subclass of Profile
  - Should be in the Configuration namespace
  - Add the mapping to be performed
  - E.g. mapping entity class `department` to `Department`

```
public class ToDepartmentProfile : Profile
{
    public ToDepartmentProfile()
    {
        CreateMap<department, Department>();
    }
}
```

Uses name matching for the attributes in the classes to be mapped

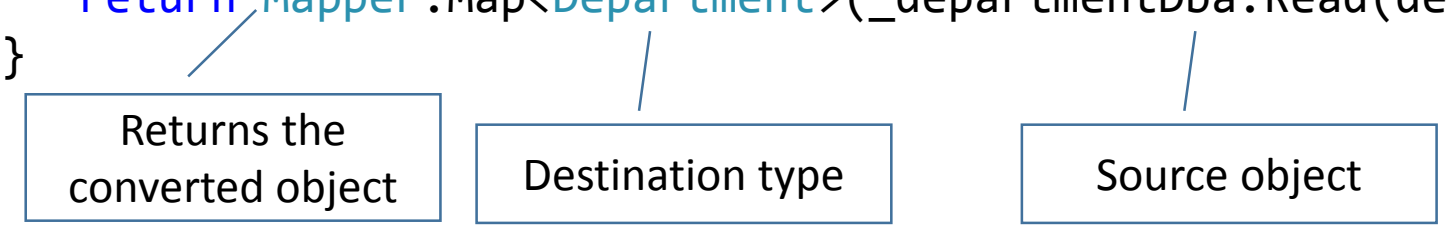
How do we use mapping features in our code?

- To be used in the service layer methods
- We need an instance of the repository class containing methods for querying the database
  - E.g. from Project Manager class for department queries is DepartmentDba:

```
static private DepartmentDba _departmentDba = new DepartmentDba();
```

  - To do the mapping for retrieving a specific department:

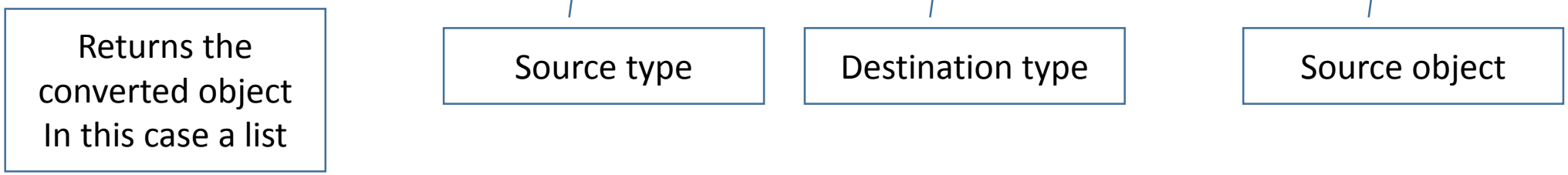
```
static public Department getDepartment(int depId)
{
    return Mapper.Map<Department>(_departmentDba.Read(depId));
}
```



Querying a List

- This is equally simple

```
static public List<Department> getDepartments()
{
  return Mapper.Map<List<department>, List<Department>>( _departmentDba.List());
}
```



Reading

- Books:
  - Kanjilal, Joydip “Entity Framework Tutorial - Second Edition”, Fulltext biblioteket
  - Driscoll, Brian ” Entity Framework 6 Recipes, Second Edition”, Fulltext biblioteket
  - Lerman, Julia “Programming Entity Framework: DbContext”, Fulltext biblioteket
- Links:
  - https://msdn.microsoft.com/en-us/library/jj592676(v=vs.113).aspx
  - https://github.com/AutoMapper/AutoMapper/wiki
  - http://stackoverflow.com/questions/34083796/using-transactionscope-with-entity-framework-6