**Lab NHibernate**

# Getting ready:

Create the following folder structure:

* D:\dev\Lab\_NHibernate\MovieStore\lib
* D: \dev\ Lab\_NHibernate \MovieStore\src

Copy the content of the hol\lib folder to D:\dev\ Lab\_NHibernate \MovieStore\lib

Create a solution MovieStore in D:\dev\ Lab\_NHibernate \MovieStore\src

Add 3 new projects to the solution

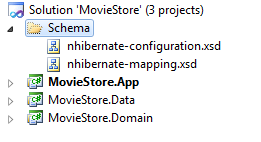
* MovieStore.Data (class lib)
* MovieStore.Domain (class lib)
* MovieStore.App (console app)

Right-click on the **Solution**, and select **Add** **| New Solution Folder**

Name the folder *Schema*.

Browse to the Lib folder (D:\dev\ Lab\_NHibernate \MovieStore\lib) and add two files: *nhibernate-configuration.xsd* *and nhibernate-mapping.xsd*. When the files open in the editor, just close them.

Set the MovieStore.App project as startup-project. Right-click the project and chose *Set as Startup project*.

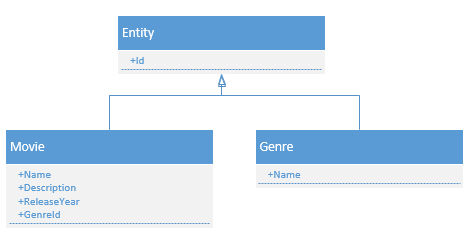


# The first steps:

Now, let’s start by creating our Genre class with the following steps:

In *MovieStore.Domain*, create a new C# class named *Entity* with the following code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace MovieStore.Domain

{

public class Entity

{

public virtual int Id { get; set; }

}

}

Create a new class named *Genre* with the following code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace MovieStore.Domain

{

public class Genre: Entity

{

public virtual string Name { get; set; }

}

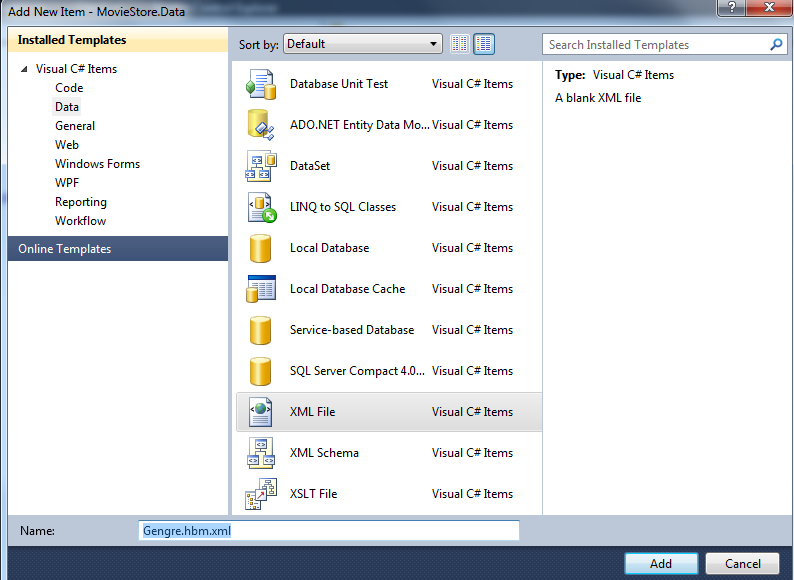
}

Build you application and correct any compiler errors.

Now we’ll create our first mapping file.

Right-click the MovieStore.Data project, and choose **Add | New Folder**

Name the folder **Mappings**

Right-click the new created folder and add a new xml file**: Add | New File …**

Name the file **Genre.hbm.xml**

In the solution explorer, right-click on **Genre.hbm.xlm**, and choose **Properties**.

Change **Build Action** from **Content** to **Embedded Resource**.

In the editor, enter the following XML in Genre.hbm.xml. Let the IntelliSense guide you. (The intelliSense is made possible because you’ve added the two xsd-files in the schema solution folder).

<?xml version="1.0" encoding="utf-8" ?>

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="MovieStore.Domain" namespace="MovieStore.Domain">

<class name="Genre">

<id name="Id">

<generator class ="native" />

</id>

<property name="Name" />

</class>

</hibernate-mapping>

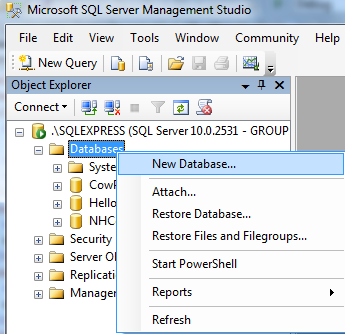
In the **MovieStore.Data** project, add a reference to the **MovieStore.Domain** project.

In the **MovieStore.App** project, add a reference to the **MovieStore.Data** project and the **MovieStore.Domain** project.

Add the following references in the **MovieStore.App** project from the **lib-folder**:

* NHibernate.dll
* NHibernate.ByteCode.Castle.dll

Open the SQL Server Management Studio and add a new database **MovieStore**.



Ad an application configuration file to the **MovieStore.App** project.

In the editor, enter the following xml in the App.config file:

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<configSections>

<section name="hibernate-configuration"

type="NHibernate.Cfg.ConfigurationSectionHandler, NHibernate"/>

</configSections>

<connectionStrings>

<add name="db" connectionString="Server=.\SQLEXPRESS;Database=MovieStore; Trusted\_Connection=SSPI"/>

</connectionStrings>

<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">

<session-factory>

<property name="proxyfactory.factory\_class">NHibernate.ByteCode.Castle.ProxyFactoryFactory, NHibernate.ByteCode.Castle</property>

<property name="dialect">NHibernate.Dialect.MsSql2008Dialect, NHibernate</property>

<property name="connection.connection\_string\_name">db</property>

<property name="show\_sql">true</property>

<mapping assembly="MovieStore.Data"/>

</session-factory>

</hibernate-configuration>

</configuration>

Edit the program.cs file in the **MovieStore.App** project. Add the following using statements at the beginning of the file:

* using NHibernate;
* using NHibernate.Cfg;
* using NHibernate.Tool.hbm2ddl;
* using MovieStore.Domain;

Add the following code in the Main method:

var nhConfig = new Configuration().Configure();

var sessionFactory = nhConfig.BuildSessionFactory();

Console.WriteLine("NHibernate is now configured");

var schemaExport = new SchemaExport(nhConfig);

//let NHibernate create your database with the following statement

//it's easy for now, but usually not appropriate for your application to

//recreate database tables each time

schemaExport.Create(false, true);

//use the following statement if you want the db.sql script so you can execute it yourself

//schemaExport.SetOutputFile(@"db.sql").Execute(false, false, false);

Console.WriteLine("Database schema created");

Console.ReadKey();

Build and run the application.

Check your database.

# Working with persistent objects

Edit the **program.cs** file in the **MovieStore.App** project.

Enter the following code **above** the statement Console.ReadKey();

using (ISession session = sessionFactory.OpenSession())

{

using (session.BeginTransaction())

{

Genre g1 = new Genre();

g1.Name = "Thriller";

session.Save(g1);

}

}

Build your solution and run the application.

Note that you can see the query that is executed in the console window. That’s because we’ve added the following statement in the app.config file:

<property name="show\_sql">true</property>

Check your database.

Nothing is added to the database because we didn’t commit the transaction. Add the following code beneath the “session.Save(g1);” statement: .

session.Transaction.Commit();

Run the application again and check your database. If you aren’t excited now, you’re definitely in the wrong course. We didn’t write any SQL code yet!

We can get the data out of the database as well. Add the following code **above** the statement Console.ReadKey(); to retrieve our genre:

Genre myGenre;

using (ISession session = sessionFactory.OpenSession())

{

using (session.BeginTransaction())

{

myGenre = session.Get<Genre>(1);

}

}

Console.WriteLine("Found a " + myGenre.Name + " genre!");

# Implementing the Movie class

Add a **movie** class to the **MovieStore.Domain** project:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace MovieStore.Domain

{

public class Movie : Entity

{

public virtual string Name { get; set; }

public virtual string Description { get; set; }

public virtual int ReleaseYear { get; set; }

public virtual Genre Genre { get; set; }

}

}

Add a new mapping file **movie.hbm.xml** to the **MovieStore.Data** project:

<?xml version="1.0" encoding="utf-8" ?>

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="MovieStore.Domain" namespace="MovieStore.Domain">

<class name="Movie">

<id name="Id">

<generator class ="native" />

</id>

<property name="Name" />

<property name="Description" />

<property name="ReleaseYear" />

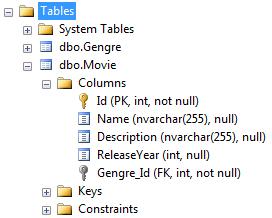
<many-to-one name="Genre" column="Genre\_Id" class="Genre" not-null="true" />

</class>

</hibernate-mapping>

Don’t forget to change the **build action** to **embedded resource**.

Run the application and check your database. Note the foreign key constraint!



Add the following code **before** the “Console.ReadKey();” statement:

using (ISession session = sessionFactory.OpenSession())

{

using (session.BeginTransaction())

{

Movie m1 = new Movie { Name = "Old Boy",

Description = "Korean movie",

ReleaseYear = 2008,

Genre = myGenre };

session.Save(m1);

session.Transaction.Commit();

}

}

Run the application and check the database. Note that NHibernate took care of the foreign key Genre\_Id.

We’ll try to request the data by adding the following code **before** the “Console.ReadKey();” statement:

Movie myMovie;

using (ISession session = sessionFactory.OpenSession())

{

using (session.BeginTransaction())

{

myMovie = session.Get<Movie>(1);

}

}

Console.WriteLine("Founed a " + myMovie.Genre.Name + " movie: " + myMovie.Name);

Run the application and see what’s happening. Our application throws an error because it can’t retrieve the data. Remember that NHibernate is using lazy loading by default. It only loads the data when needed. Once we’re outside our session scope, the objects are detached (=no longer linked to NHibernate).

Change the code to:

Movie myMovie;

using (ISession session = sessionFactory.OpenSession())

{

using (session.BeginTransaction())

{

myMovie = session.Get<Movie>(1);

}

Console.WriteLine("Founed a " + myMovie.Genre.Name + " movie: " + myMovie.Name);

}

Run the application and you should retrieve the thriller movie.

We’ll add some more data to work with:

* Add a new genre “Drama”
* Add a new thriller movie “Loft”
* Add a new drama movie “127 hours”

You can add all the new data in one transaction. Beware to save the new genre “Drama” before you save the movie “127 hours”.

The code can look like this:

using (ISession session = sessionFactory.OpenSession())

{

using (session.BeginTransaction())

{

Genre g1 = new Genre { Name = "Drama" };

Movie m1 = new Movie

{

Name = "Loft",

Description = "Bart De Pauw did his best",

ReleaseYear = 2009,

Genre = myGenre

};

Movie m2 = new Movie

{

Name="127 Hours",

Description = "Time is ticking",

ReleaseYear = 2010,

Genre = g1

};

// session.Save(g1); not needed because of the cascase="save-update" option in mapping file

session.Save(m1);

session.Save(m2);

session.Transaction.Commit();

}

}

Note: If you don’t use the cascade option in the movie mapping file you need to save the genre before you use it in any movie object.

To retrieve a list of all Movies we can use an IQuery. Add the following code **before** the “Console.ReadKey();” statement:

using (ISession session = sessionFactory.OpenSession())

{

using (session.BeginTransaction())

{

IQuery query = session.CreateQuery("from Movie as m order by m.Name");

IList<Movie> movieList = query.List<Movie>();

foreach (Movie m in movieList)

{

Console.WriteLine("Found movie " + m.Name + ".");

}

}

}

The query we’ll use looks like SQL but it’s actually HQL (Hibernate Query Language). Note that HQL is case sensitive. If you’d have written “movie”, the program would throw an error.

The movie Loft was not released in 2009 but in 2008. Update the record.

Add a new class Actor with properties FirstName, LastName, BirthDate and make the mapping file.

A movie can have different actors. Change the class and adapt the mapping file. (view solution)