NHibernate - badanie możliwości i opracowanie przykładów wykorzystania technologii ORM

2010-02-15

Pollitechnika Koszalinska
Wlodzimierz Khadzhynov

1. Wstęp 2
2. Tworzenie podstawowej struktury 3
2.1. Opis przykładowego systemu 3
2.2. Tworzenie projektu oraz klas trwałych 3
2.2.1. Wstęp teoretyczny 3
2.2.2. Część praktyczna 4
2.3. Tworzenie plików mapujących HBM 7
2.3.1. Wstęp teoretyczny 7
2.3.2. Część praktyczna 10
2.4. Konfiguracja NHibernate 11
2.4.1. Wstęp teoretyczny 11
2.4.2. Część praktyczna 12
2.5. Automatyczne tworzenie bazy danych na podstawie plików HBM 13
2.5.1. Wstęp teoretyczny 13
2.5.2. Część praktyczna 14
2.6. Implementacja funkcji systemu 17
2.6.1. Wstęp teoretyczny 17
2.6.2. Część praktyczna 19
3. Tworzenie systemu trójwarstwowego 22
3.1. Wstęp teoretyczny 23
3.2. Część praktyczna 24
4. Podsumowanie 28
5. Zadanie Indywidualne 28

1. Wstęp

Celem ćwiczeń jest badanie technologii NHibernate dla tworzenia aplikacji w środowisku DOTNET z możliwościami obiektowo relacyjnego mapowania obiektów klas programowych. Ćwiczenie przebiega w formie opracowania projektu aplikacji bazodanowej realizującej funkcji biblioteki.

Wymagania dla realizacji ćwiczenia

- Biblioteka NHibernate - Przed przystąpieniem do samodzielnej realizacji ćwiczeń należy pobrać ze strony domowej NHibernete bibliotekę w wersji 1.2.1 GA (plik powinien nazywać się „NHibernate-1.2.1.GA.msi”): (http://sourceforge.net/project/showfiles.php?group_id=73818&package_id=73969). Po ściągnięciu tego pliku należy go uruchomić celem instalacji w systemie;
- MS Visual Studio 2005 Professional albo MS Visual C# 2005 Express Edition – środowisko programistyczne dla platformy .NET;
- MS SQL Server 2005 lub MS SQL Server 2005 Express – relacyjna baza danych;
- MS SQL Server Management Studio Express – narzędzie do zarządzania bazą danych z interfejsem graficznym;
- Znajomość podstaw programowania w języku C# oraz .NET WinForms;
- Umiejętność korzystania z narzędzia do zarządzania bazą danych MS SQL Server Management Studio Express.

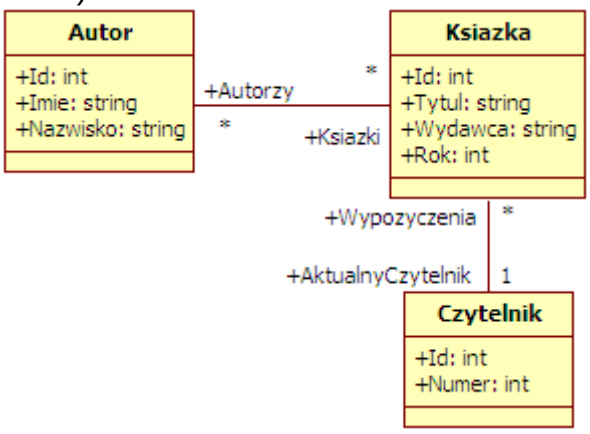
Uwaga: w trakcie przygotowywania niniejszego opracowania biblioteka NHibernate była w wersji 1.21. Jednak dostępna do testów alpha była wersja 2.0. Jednak nie należy korzystać z tej wersji przy wykonywaniu ćwiczeń, ponieważ wersja 2.0 nie jest w pełni kompatybilna wstecz z wersją 1.2.

2. Tworzenie podstawowej struktury

Ten rozdział opisuje budowę systemu dwuwarstwowego oraz implementację podstawowych funkcji tego systemu przy wykorzystaniu biblioteki NHibernate.

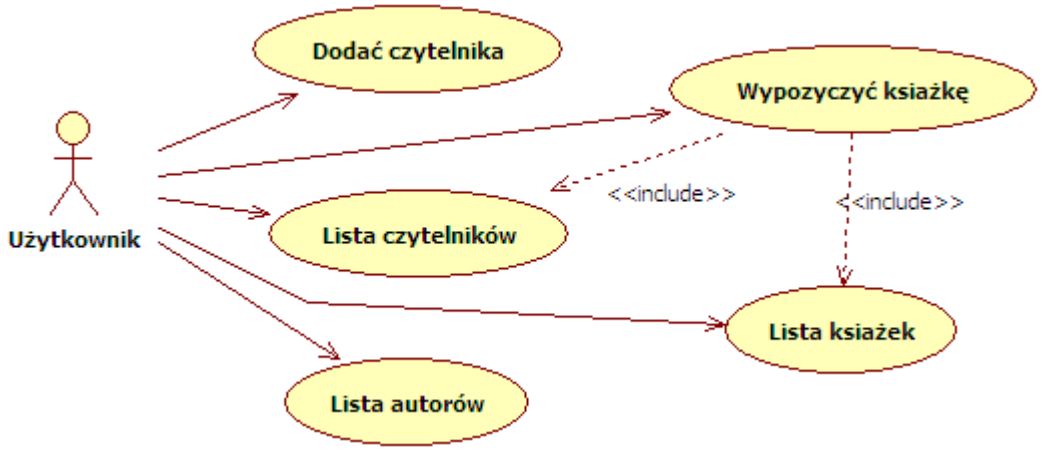
2.1. Opis przykładowego systemu

Przykładowym projektem będzie część systemu Biblioteki. Wykorzystuje on niektóre klasy z prezentacji (rys. 2.1).



Rys. 2.1. Diagram klas przykładowego systemu

Funkcjonalność systemu przedstawiona jest na diagramie przypadków użycia (rys. 2.2).



Rys. 2.2. Diagram przypadków użycia przykładowego systemu

Użytkownik może: dodać czytelnika, wypożyczyć książkę czytelnikowi, obejrzeć listę czytelników oraz listę książek i autorów.

2.2. Tworzenie projektu oraz klas trwałych

2.2.1. Wstęp teoretyczny

Głównym celem biblioteki NHibernate jest umożliwienie zapisywania, ładowania i wyszukiwania obiektów trwałych w bazie danych. Obiekty trwałe są instancjami klas zdefiniowanych do tego celu przez użytkownika (wszystkie takie klasy muszą być opisane w plikach mapujących HBM (p. 2.3)).

Warunki dla klas trwałych
Klasa trwała (zwana także encją) musi spełniać pewne warunki, aby mogła współpracować z NHibernate:
- musi być publiczna;
- ma konstruktor bezparametrowy lub nie ma żadnego konstruktora;
- właściwości, które udostępniają pola muszą być publiczne i wirtualne;
- pola, które są udostępniane przez właściwości, muszą być prywatne;
- właściwości, które nie będą zapisywane do bazy także muszą być wirtualne;
Dodatkowe uwagi:
- jest możliwość zapisywania prywatnych pół bezpośrednio do bazy z pominięciem właściwości;
- klasa może zawierać metody zawierające logikę biznesową;
- można używać kolekcji generycznych i niegenerycznych;
Należy zauważyć, że NHibernate narzuca bardzo mało warunków na klasy trwałe, w porównaniu do
innych (zwłaszcza starszych) narzędzi ORM.
Tworzenie struktury projektu
Dodatkowym warunkiem nałożonym na cały projekt warstwy danych jest posiadanie referencji do biblioteki NHibernate (NHibernate.dll) oraz opcjonalnie niezbędnych dla niej bibliotek pomocniczych (Castle.DynamicProxy.dll, Iesi.Collections.dll, log4net.dll).
Referencje mogą odwoływać się do bibliotek umieszczonych w katalogu instalacyjnym NHibernate, jednak w większych projektach (umieszczanych na systemach kontroli wersji) w projekcie tworzy się dodatkowy katalog (np. Lib) i tam kopiuje się wymagane biblioteki.
Aby odwzorować logiczne warstwy systemu w projektach odpowiednio tworzy się strukturę katalogów. Klasy encyjne umieszcza się w podkatalogu „Domain” lub „Logic”, a pliki mapujące w katalogi „HBM”.

2.2.2. Część praktyczna

a) Tworzenie projektu
- uruchomić MS Visual Studio 2005 Professional lub MS Visual C# 2005 Express;
- stworzyć nowy projekt o nazwie „NHTutorial” (nie tworzyć osobnego katalogu na solution) i zapisać go;
b) Dodawanie referencji do bibliotek
- w oknie Solution Explorer zaznaczyć element „References”, kliknąć prawym przyciskiem myszy i wybrać z menu „Add reference”.
- z nowego okna wybrać zakładkę „Browse”, odnaleźć katalog bin w katalogu, w którym zainstalowano NHibernate, a następnie wybrać bibliotekę „NHibernate.dll” (opcjonalnie można dodać biblioteki: Castle.DynamicProxy.dll, Iesi.Collections.dll, log4net.dll, jednak nie jest to konieczne);
c) Tworzenie struktury projektu
- w projekcie stworzyć nowy folder o nazwie „Domain” (prawym przyciskiem myszy kliknąć na nazwę projektu i wybrać Add->New Folder).
- analogicznie dodać nowy folder o nazwie „HBM”.
d) Dodać w folderze „Domain” następujące klasy:
- Autor:

```
using System;
using System.Collections.Generic;
namespace NHTutorial.Domain {
```

```
public class Autor
{
private int _id;
private string _imie;
private string _nazwisko;
private IList<Ksiazka> _ksiazki;
public virtual int Id {
get { return _id; }
set { _id = value; }
}
public virtual string Imie {
get { return _imie; }
set { _imie = value; }
}
public virtual string Nazwisko {
get { return _nazwisko; }
set { _nazwisko = value; }
}
public virtual IList<Ksiazka> Ksiazki {
get { return _ksiazki; }
set { _ksiazki = value; }
}
public override string ToString() {
return String.Format("[{0}, {1}, {2}, {3}]", _id, _imie,
_nazwisko, _ksiazki.Count);
}
}
}
```

- Książka:

```
using System;
using System.Collections.Generic;
namespace NHTutorial.Domain {
public class Ksiazka
{
private int _id;
private string _tytul;
private string _wydawca;
private int _rok;
private IList<Autor> _autorzy;
private Czytelnik _aktualnyCzytelnik;
public virtual int Id {
get { return _id; }
set { _id = value; }
}
public virtual string Tytul {
get { return _tytul; }
set { _tytul = value; }
}
public virtual string Wydawca {
get { return _wydawca; }
set { _wydawca = value; }
}
public virtual int Rok {
get { return _rok; }
set { _rok = value; }
}
public virtual IList<Autor> Autorzy {
get { return _autorzy; }
set { _autorzy = value; }
}
```

```
public virtual Czytelnik AktualnyCzytelnik {
get { return _aktualnyCzytelnik; }
set { _aktualnyCzytelnik = value; }
}
public override string ToString() {
return String.Format("[{0}, {1}, {2}, {3}]", _id, _tytul,
_wydawca, _rok);
}
}
}
```

- Czytelnik:

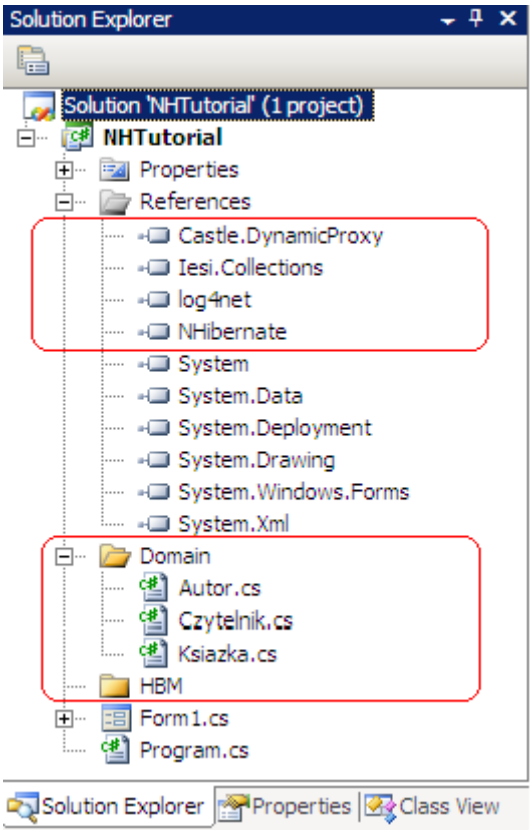
```
using System;
using System.Collections.Generic;
namespace NHTutorial.Domain {
public class Czytelnik
{
private int _id;
private int _numer;
private IList<Ksiazka> _wypozyczenia;
public virtual int Id {
```



```
get { return _id; }
set { _id = value; }
}
public virtual int Numer {
get { return _numer; }
set { _numer = value; }
}
public virtual IList<Ksiazka> Wypozyczenia {
get { return _wypozyczenia; }
set { _wypozyczenia = value; }
}
public override string ToString() {
return String.Format("[{0}, {1}]", Id, _numer);
}
}
}
```

e) Sprawdzenie projektu

- widok solution powinien wyglądać jak na rysunku 2.3;



Rys. 2.3. Widok solution prawidłowo stworzonego projektu

- projekt powinien się kompilować;

2.3. Tworzenie plików mapujących HBM

2.3.1. Wstęp teoretyczny

Pliki mapujące są pomostem pomiędzy obiektową częścią systemu w pamięci, a relacyjną częścią w bazie danych. W tym rozdziale zostanie opisana budowa plików mapujących HBM oraz zagadnienia z nimi związane.

Opis znaczników mapujących

Poniżej opisane są podstawowe znaczniki wykorzystywane w plikach mapujących:

- ***hibernate-mapping***

```
<hibernate-mapping
 schema="schemaName"
assembly="Asse"
namespace="Asse.Domain" />
```

Ten znacznik określa pewne domyślne wartości dla całego opisu. Zawiera wszystkie inne znaczniki. Elementy:

schema – opcjonalna nazwa dla schematu bazy danych;

assembly – nazwa domyślnego assembly dla klas, będzie wykorzystywana gdy klasa nie będzie miała jawnie wyszczególnionego assembly;

namespace – nazwa domyślnego namespace'u (użycie jak dla assembly);

- ***class***

```
 <class
name="ClassName"
table="tableName"
dynamic-update="true|false"
dynamic-insert="true|false"
polymorphism="implicit|explicit"
```

```
where="arbitrary sql where condition"
optimistic-lock="none|version|dirty|all"
lazy="true|false"
/>
```
Ten znacznik służy do opisu klasy. Elementy:
name – nazwa klasy opisywanej, w razie konieczności użyć Full Name;
table – nazwa tabeli na którą klasa jest mapowana;
dynamic-update – dla wartości true kody sql update będą generowane w trakcie działania programu tylko dla kolumn, których wartości zostały zmienione;
dynamic-insert - dla wartości true kody sql insert będą generowane w trakcie działania programu tylko dla kolumn, których wartości nie są równe null;
polymorphism – określa czy używany jest domyślny czy jawny polimorfizm dla klasy;
where – opcjonalny dodatek do klauzuli where przy wybieraniu klasy;
optimistic-lock – określenie rodzaju optymistycznej konkurencyjności;
lazy – włączenie lub wyłączenie opóźnionego ładowania klasy;

- ***id***

```
<id
name="PropertyName"
type="typename"
column="column_name"
access="field|property|nosetter|ClassName">
<generator class="generatorClass"/>
</id>
```
Ten znacznik służy do określenia nazwy i zachowania identyfikatora klasy. Elementy:
- **name** – określa nazwę pola w klasie, które zawiera Id;
- **type** – określa typ pola w klasie, które zawiera Id;
- **column** – klucz główny w tabeli docelowej;
- **access** – rodzaj strategi przy dostępie do pola w klasie;
- **znacznik generator** – określa tryb generowania wartości dla pola Id;

- ***property***

```
 <property
name="propertyName"
column="column_name"
type="typename"
update="true|false"
insert="true|false"
access="field|property|ClassName"
optimistic-lock="true|false"
/>
```
Ten znacznik opisuje pola klasy, które będą zapisywane w bazie. Elementy:
- **name** – nazwa właściwości;
- **column** – nazwa kolumny w tabeli;
- **type** – typ właściwości;
- **update, insert** – określenie czy dana właściwość powinna być uwzględniana przy kodach sql update i insert;
- **access** – to samo co w id;
- **optimistic-lock** – czy dana właściwość powinna brać udział w Optimictic Lock;

- ***many-to-one i oneto-one***

```
<many-to-one
name="PropertyName"
column="column_name"
class="ClassName"
```

```
cascade="all|none|save-update|delete"
/>
<one-to-one
name="PropertyName"
class="ClassName"
cascade="all|none|save-update|delete"
/>
```
Znaczniki opisującą relację wiele do jednego i jeden do jednego dla danej klasy.
Elementy:
- **name** – nazwa właściwości do której relacja jest przypisana;
- **column** – nazwa kolumny realizującej relację;
- **class** – nazwa klasy po drugiej stronie relacji;
- **cascade** – typ operacji przenoszonych kaskadowo;

- bag, set, list, map (kolekcje)

```
<map
name="propertyName"
table="table_name"
lazy="true|false"
inverse="true|false"
cascade="all|none|save-update|delete|all-delete-orphan"
sort="unsorted|natural|comparatorClass"
order-by="column_name asc|desc"
>
...
</map>
```
Kolekcje opisują relację one-to-many oraz many-to-many. Kolekcja map zawiera reprezentatywny zestaw właściwości. Elementy:
- **name** – opis właściwości realizującej kolekcję w danej klasie;
- **table** – nazwa tabeli realizującej kolekcję (nie dla one-to-many);
- **lazy** – czy kolekcja ma być ładowna z opóźnieniem;
- **inverse** – dla własnego spokoju należy ustawiać tą wartość na true ;)
- **sort** – określenie sposobu sortowania kolekcji;
- **order-by** – określenie kolumny, według której będzie sortowana kolekcja przy pobieraniu z bazy;

Powyższy spis znaczników nie jest wyczerpujący. Dodatkowo dla jasności w większość wymienionych
znaczników wyszczególniono tylko niektóre atrybuty.

Dodatkowe uwagi
- pliki mapujące powinny mieć rozszerzenie „.hbm.xml”. Ułatwia to pracę z nimi;
- przy dodawaniu plików HBM do projektu w Visual Studio trzeba ustawić właściwość (w oknie
Properties) o nazwie „Build Action” na „Embedded Resource”;

- NHibernate pozwala na zastąpienie plików HBM poprzez atrybuty w klasach, przypisane do
konkretnych właściwości. Jednak ten sposób nie jest dobrze opisany, w dodatku zaciemnia kod klasy
i powoduje spory coupling warstwy Domain z DAL;
- dodatkowe znaczniki opisane są w dokumentacji;
- nieopisany znacznik idbag służy do tworzenia relacji wiele-do-wielu z dodatkowym kluczem
pierwotnym w tabeli realizującej relację;

2.3.2. Część praktyczna

a) Stworzenie pliku mapującego dla klasy Czytelnik

- zaznaczyć folder „HBM” w projekcie, następnie kliknąć prawym przyciskiem i wybrać Add->NewItem;
- z listy wybrać „XML File” i podać nazwę „Czytelnik.hbm.xml”;
- zaznaczyć stworzony plik i kliknąć na nim prawym przyciskiem myszy, z menu wybrać element Properties. We właściwościach pliku TRZEBA koniecznie ustawić „Build Action” na „Embedded Resource”;
- przejść do edycji pliku i wpisać:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
namespace="NHTutorial.Domain" assembly="NHTutorial">
<class name="Czytelnik" table="Czytelnicy">
<id name="Id" type="Int32">
<column name="CzytelnikId" not-null="true"/>
<generator class="identity"/>
</id>
<property name="Numer" type="Int32">
<column name="Numer" not-null="true"/>
</property>
<bag name="Wypozyczenia" cascade="save-update"
lazy="true" inverse="true">
<key column="CzytelnikId"/>
<one-to-many class="Ksiazka"/>
</bag>
</class>
</hibernate-mapping>
```

b) Stworzenie plików mapujących dla klas Autor i Ksiazka

- dla klasy Autor powtórzyć punkt a wpisując w plik „Autor.hbm.xml” zawartość:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
namespace="NHTutorial.Domain" assembly="NHTutorial">
<class name="Autor" table="Autorzy">
<id name="Id" type="Int32">
<column name="AutorId" not-null="true"/>
<generator class="identity"/>
</id>
<property name="Imie" type="String">
<column name="Imie" length="100" not-null="true"/>
</property>
<property name="Nazwisko" type="String">
<column name="Nazwisko" length="100" not-null="true"/>
</property>
<idbag name="Ksiazki" table="Ksiazki_Autorzy"
cascade="save-update" lazy="true">
<collection-id column="KsiazkiAutorzyId" type="Int32">
<generator class="identity"/>
NHibernate - badanie możliwości i opracowanie przykładów wykorzystania technologii ORM 11
</collection-id>
<key column="AutorId"/>
<many-to-many class="Ksiazka" column="KsiazkaId"/>
</idbag>
</class>
</hibernate-mapping>
```

- dla klasy Ksiazka powtórzyć punkt a wpisując w plik „Ksiazka.hbm.xml” zawartość:

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
namespace="NHTutorial.Domain" assembly="NHTutorial">
```

```
<class name="Ksiazka" table="Ksiazki">
<id name="Id" type="Int32">
<column name="KsiazkaId" not-null="true"/>
<generator class="identity"/>
</id>
<property name="Tytul" type="String">
<column name="Tytul" length="500" not-null="true"/>
</property>
<property name="Wydawca" type="String">
<column name="Wydawca" length="100" not-null="true"/>
</property>
<property name="Rok" type="Int32">
<column name="Rok" not-null="true"/>
</property>
<many-to-one name="AktualnyCzytelnik" column="CzytelnikId"
cascade="save-update"/>
<idbag name="Autorzy" table="Ksiazki_Autorzy"
cascade="save-update" lazy="true">
<collection-id column="KsiazkiAutorzyId" type="Int32">
<generator class="identity"/>
</collection-id>
<key column="KsiazkaId"/>
<many-to-many class="Autor" column="AutorId"/>
</idbag>
</class>
</hibernate-mapping>
```

2.4. Konfiguracja NHibernate

2.4.1. Wstęp teoretyczny

Konfirurowanie NHibernate polega na określeniu połączenia z bazą danych oraz wskazaniu położenia
plików HBM. Na tym etapie można także modyfikować globalne działanie biblioteki.

Klasy Configuration oraz ISessionFactory

Klasa Configuration zbiera informacje o konfiguracji NHibernate'a. Poprzez odpowiednie właściwości ustawia się połączenie oraz rodzaj bazy danych.
Najważniejszą właściwością jest *„hibernate.connection.connection_string”*, określa ona Connection String (łańcuch znaków opisujący połączenie z bazą danych).
Poprzez metodę AddResource można określić położenie plików HBM.
Po skonfigurowaniu obiektu Configuration można stworzyć obiekt ISessionFactory metodą
BuildSessionFactory(). ISessionFactory służy do tworzenia obiektów ISession, które umożliwiają wykonywanie operacji na bazie danych.
Tworzenie obiektu ISessionFactory jest długotrwałe, dlatego powinno być wykonywane tylko raz podczas działania aplikacji, aby nie spowalniać jej działania.

Klasa PersistenceManager

Aby łatwo zarządzać połączeniami z bazą tworzy się klasę pomocniczą, która nazywa się najczęściej PersistenceManager (nie jest ona częścią biblioteki!). W niej umieszcza się kod służący do:
- konfigurowania biblioteki NHibernate;
- tworzenie obiektu ISessionFactory;
- tworzenie i zarządzanie obiektami ISession;
- itp;

Klasę PersistenceManager (PM) implementuje się wykorzystując wzorzec Singleton, aby była łatwo dostępna w programie;

2.4.2. Część praktyczna

a) Tworzenie klasy PersistenceManager
- Dodać do projektu nową klasę (Add->New Item, wybrać element Class) o nazwie „PersistenceManager”.
- wypisać poniższy kod do stworzonego pliku „PersistenceManager.cs”:

```
using System;
using NHibernate;
using NHibernate.Cfg;
namespace NHTutorial {
public class PersistenceManager
{
#region Singleton
private static readonly PersistenceManager instance
= new PersistenceManager();
private PersistenceManager() {
Init();
}
public static PersistenceManager Instance {
get {
return instance;
}
}
#endregion
private const string DIALECT = "NHibernate.Dialect.MsSql2005Dialect";
private const string CONNECTION_PROVIDER
= "NHibernate.Connection.DriverConnectionProvider";
private const string CONNECTION_STRING_SERVER =
@"Server=.\SQLEXPRESS;Integrated Security=SSPI;Connection Timeout=10;";
private const string CONNECTION_STRING_DB
= @"initial catalog=NHTutorialDB;";
private const string CONNECTION_STRING
= CONNECTION_STRING_SERVER + CONNECTION_STRING_DB;
private readonly string[] HBM_ENTITY_FILES = {
"NHTutorial.HBM.Autor.hbm.xml",
"NHTutorial.HBM.Ksiazka.hbm.xml",
"NHTutorial.HBM.Czytelnik.hbm.xml",
};
private ISessionFactory _sessionFactory;
private Configuration _nhconfig;
private void Init() {
NHibernate - badanie możliwości i opracowanie przykładów wykorzystania technologii ORM 13
try {
_nhconfig = new Configuration();
_nhconfig.SetProperty("hibernate.dialect", DIALECT);
_nhconfig.SetProperty("hibernate.connection.provider",
CONNECTION_PROVIDER);
_nhconfig.SetProperty("hibernate.connection.connection_string",
CONNECTION_STRING);
foreach (string res in HBM_ENTITY_FILES)
_nhconfig.AddResource(res, System.Reflection.Assembly
.GetExecutingAssembly());
_sessionFactory = _nhconfig.BuildSessionFactory();
}
catch (Exception ex) {
// wyjątek w tym miejscu oznacza błąd w plikach HBM
Console.Write(ex);
```

```
System.Diagnostics.Debugger.Break();
}
}
public ISessionFactory SessionFactory {
get {
return _sessionFactory;
}
}
}
}
```

- Uwaga: w stałej CONNECTION_STRING umieszczone są informacje o połączeniu z bazą danych. Aktualnie odnoszą się do bazy NHTutorialDB umieszczonej na lokalnym serwerze MS SQL Express 2005. Jeżeli przy wykonywaniu ćwiczenia parametry są inne to należy poprawić tą stałą;

b) Dostęp do instancji klasy PersistenceManager wewnątrz głównego okna Form1

- otworzyć okno Form1 w trybie edycji kodu;
- dodać prywatne pole typu PersistenceManager o nazwie „_pm”;
- w konstruktorze przypisać zmiennej „_pm” właściwość statyczną Instance z klasy PersistenceManager;
- kod okna powinien wyglądać następująco:

```
namespace NHTutorial
{
public partial class Form1 : Form
{
private PersistenceManager _pm;
public Form1()
{
InitializeComponent();
_pm = PersistenceManager.Instance;
}
}
}
```

c) Uruchomienie programu

- program powinien się skompilować bez błędów i powinno po uruchomieniu pojawić się puste okno nazwane Form1;
- jeżeli wystąpią błędy (np. w plikach HBM) to program zatrzyma się w bloku catch w klasie PersistenceManager i obiekt wyjątku (ex) zostanie wypisany na konsole. Należy odszukać błąd według opisu wyjątku i poprawić go;

2.5. Automatyczne tworzenie bazy danych na podstawie plików HBM

2.5.1. Wstęp teoretyczny

W warunkach szybkiego rozwoju projektu baza danych może być nieustannie zmieniana. Manualne poprawianie schematu bazy danych jest podatne na błędy. W takich przypadkach można zastosować funkcjonalność automatycznego generowania schematu bazy danych.

Klasa SchemaExport

Klasa SchemaExport znajduje się w namespace „NHibernate.Tool.hbm2ddl”. Dzięki niej można:

- wygenerować skrypt tworzący bazę danych (także do pliku);

- wygenerować skrypt usuwający bazę danych;
- wykonać skrypt tworzący/usuwający bazę bezpośrednio na bazie danych.

Jako parametr konstruktora klasy trzeba podać instancję klasy Configuration. Analizując pliki HBM klasa ta generuje skrypty. Jeżeli trzeba połączyć się z bazą to korzysta się z połączenia zdefiniowanego w klasie Configuration.

Metody:

- SetOutputFile(string name) - służy do ustawienia pliku wyjściowego dla skryptu;
- Execute(bool script, bool export, bool justDrop, bool format) – służy do wygenerowania skryptu.

Uwaga, jeżeli skrypt ma być wykonany bezpośrednio na bazie, to musi ona już istnieć.

Parametry:

- script – określa czy wypisać tworzony skrypt na konsole;
- export – określa czy stworzony skrypt ma być wykonany na bazie danych;
- justDrop – generowanie tylko skryptu usuwającego bazę;
- format – określa czy skrypt ma być sformatowany tak, aby był czytelny dla ludzi;

Tworzenie bazy

Przed pierwszym generowaniem skryptu trzeba stworzyć bazę danych na serwerze. Można to wykonać przy pomocy dodatkowych narzędzi (np.: SQL Management Studio). Alternatywnie można stworzyć bazę poprzez połączenie ADO.NET do serwera bazy danych bez precyzowania konkretnej bazy. Można wtedy wykonać polecenie SQL: „CREATE DATABASE <name>”, które spowoduje stworzenie bazy na serwerze.

Wady automatycznego tworzenia schematu bazy

- brak bezpośredniej kontroli nad tworzoną bazą (brak możliwości tworzenie indeksów i innych
elementów bazy nie opisywanych w plikach HBM);
- każdorazowe usuwanie danych testowych z bazy;
- tworzenie losowych nazw constraintów (np nazw kluczy obcych);

2.5.2. Część praktyczna

a) Metoda eksportu schematu na podstawie HBM

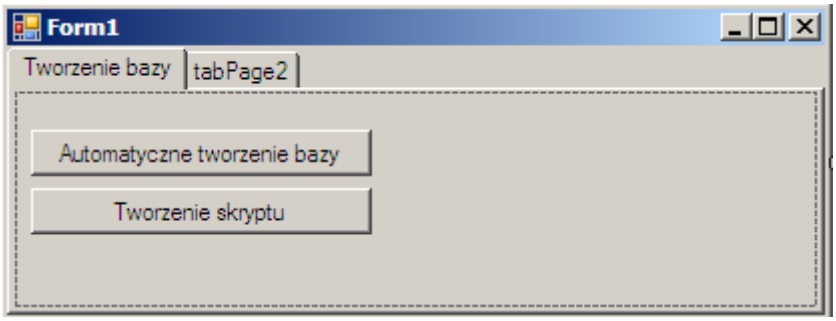
- w klasie PersistenceManager (plik „PersistenceManager.cs”) umieścić następujące metody:

```
public void ExportSchema(bool createDb)
{
NHibernate.Tool.hbm2ddl.SchemaExport se
= new NHibernate.Tool.hbm2ddl.SchemaExport(_nhconfig);
if (createDb) CreateDatabase();
else se.SetOutputFile("schema.sql");
NHibernate - badanie możliwości i opracowanie przykładów wykorzystania technologii ORM 15
se.Execute(false, createDb, false, true);
}
private void CreateDatabase()
{
SqlConnection conn = new SqlConnection();
conn.ConnectionString = CONNECTION_STRING_SERVER;
try
{
SqlCommand com = new SqlCommand();
com.Connection = conn;
com.CommandText
= String.Format("CREATE DATABASE {0}", "NHTutorialDB");
```

```
conn.Open();
com.ExecuteNonQuery();
}
catch (Exception ex)
{
Console.Write(ex);
throw new Exception("Nie udało się stworzyć bazy");
}
finally
{
conn.Close();
}
}
```

b) Przygotowanie interfejsu graficznego

- na głównym oknie umieść kontrolkę TabControl, korzystając z okna Properties ustaw jej właściwość „Dock” na wartość „Fill”;
- kliknij na wnętrze pierwszej zakładki i ustaw jej właściwość „Text” na „Tworzenie bazy”;
- umieść na pierwszej zakładce dwa przyciski, w pierwszym ustaw właściwość „Text” na „Automatyczne tworzenie bazy”, a drugim „Tworzenie skryptu”;
- okno powinno wyglądać następująco:



Rys. 2.4. Główne okno

c) Kod wywołujący metody klasy PersistenceManager

- w kodzie obsługi przycisku „Automatyczne tworzenie bazy” (w trybie edycji dwa razy kliknąć na przycisk na formie) umieścić następujący kod:

```
private void button1_Click(object sender, EventArgs e)
{
_pm.ExportSchema(true);
}
```

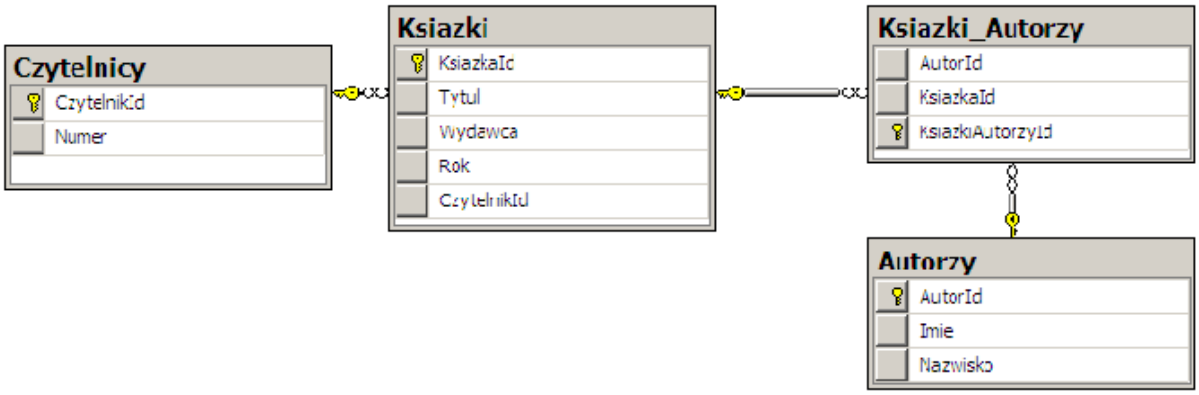
- w kodzie obsługi przycisku „Tworzenie skryptu” umieścić następujący kod:

```
private void button2_Click(object sender, EventArgs e)
{
_pm.ExportSchema(false);
}
```

d) Uruchomienie programu i generowanie bazy

- należy uruchomić program i ewentualnie poprawić błędy kompilacji;
- kliknąć przycisk „Tworzenie skryptu”. W katalogu projektu „.\bin\debug” powinien zostać utworzony plik „schema.sql”. Otworzyć ten plik edytorem tekstowym i sprawdzić, czy zawiera skrypt tworzenia schematu bazy (nie wykonywać go!);

e) - kliknąć przycisk „Automatyczne tworzenie bazy”. Odczekać parę sekund, a następnie sprawdzić
czy w bazie danych została utworzona nowa baza, a w niej tabele jak na rysunku 2.5. Sprawdzenia
można dokonać programem „MS SQL Server Management Studio [Express]”.



Rys. 2.5. Wygenerowany schemat bazy danych

f) Wypełnienie danymi testowymi
- w MS SQL Management Studio wykonać następujące polecenia, aby wstawić dane testowe:

```
insert into Czytelnicy (Numer) values (201);
insert into Czytelnicy (Numer) values (202);
insert into Czytelnicy (Numer) values (203);
insert into Czytelnicy (Numer) values (210);
insert into Ksiazki (Tytul, Wydawca, Rok, CzytelnikId)
values ('C# i .NET', 'Helion', 2003, null); -- 1
insert into Ksiazki (Tytul, Wydawca, Rok, CzytelnikId)
values ('Hibernate in Action', 'Manis', 2004, 1); -- 2
insert into Ksiazki (Tytul, Wydawca, Rok, CzytelnikId)
values ('Polska w europie', 'PWN', 2000, null); -- 3
insert into Ksiazki (Tytul, Wydawca, Rok, CzytelnikId)
values ('MCTS 70-526', 'MS Press', 2005, 1); -- 4
insert into Ksiazki (Tytul, Wydawca, Rok, CzytelnikId)
values ('Teleinformatyka', 'WKŁ', 2000, 2); -- 5
insert into Autorzy (Imie, Nazwisko) values ('Stephen', 'Perry'); -- 1
insert into Autorzy (Imie, Nazwisko) values ('Christian','Bauer');-- 2
insert into Autorzy (Imie, Nazwisko) values ('Gavin', 'King'); -- 3
insert into Autorzy (Imie, Nazwisko) values ('', 'nieznany'); -- 4
insert into Autorzy (Imie, Nazwisko) values ('Matthew','Stoecker');-- 5
insert into Autorzy (Imie, Nazwisko) values ('Steven','Stein'); -- 6
insert into Autorzy (Imie, Nazwisko) values ('M','Norris'); -- 7
insert into Ksiazki_Autorzy (AutorId, Ksiazkald) values (1, 1);
insert into Ksiazki_Autorzy (AutorId, Ksiazkald) values (2, 2);
insert into Ksiazki_Autorzy (AutorId, Ksiazkald) values (3, 2);
insert into Ksiazki_Autorzy (AutorId, Ksiazkald) values (4, 3);
insert into Ksiazki_Autorzy (AutorId, Ksiazkald) values (5, 4);
NHibernate - badanie możliwości i opracowanie przykładów wykorzystania technologii ORM 17
insert into Ksiazki_Autorzy (AutorId, Ksiazkald) values (6, 4);
insert into Ksiazki_Autorzy (AutorId, Ksiazkald) values (7, 5);
```

Uwaga: jeżeli w trakcie tworzenia bazy wystąpił błąd w trakcie dodawania danych testowych to należy usunąć bazę i wygenerować ją jeszcze raz. W danych testowych są hard-codowane wartości kluczy obcych, które są automatycznie przydzielane przez dbms. Jeżeli wystąpił błąd w trakcie insert'a to wartość klucza została wygenerowana, ale nie przypisana do żadnego wiersza.

Wartość sekwencji będzie przez to zaburzona;

2.6. Implementacja funkcji systemu

2.6.1. Wstęp teoretyczny

W tym podrozdziale są omówione podstawy korzystania z interfejsu obiektowego biblioteki NHibernate.
Omówiono cykl życia sesji, sposoby wybierania obiektów z bazy oraz wykonywania na nich operacji.

Tworzenie sesji oraz jej cykl życia

Nazwa	**Kod i opis**
Rozpoczęcie	ISession session = _sessionFactory.OpenSession(); Obiekt sesji tworzy się poprzez wywołanie metody OpenSession() z obiektu ISessionFactory. Od tego momentu sesja jest gotowa do użycia.
Korzystanie	Session.XYZ(); Korzystanie z sesji polega na wywoływaniu tych metod, które operują na obiektach trwałych. Najczęściej są to metody CreateCriteria(), CreateQuery(), Save(), Delete().
Zapisywanie	session.Flush(); Zapisywanie stanu sesji synchronizuje model obiektowy z relacyjnym (jest to synchronizacja jednostronna!)
Anulowanie	session.Clear(); Wywołanie metody Clear() powoduje, że wszelkie zmiany dokonane na obiektach są anulowane. Jednocześnie wszystkie pobrane z bazy obiekty są odłączane od sesji (należy pobrać wszystko od nowa).
Zamknięcie	Session.Close(); Wywołanie tej metody jest konieczne, aby zakończyć połączenie z bazą danych.

Podstawowe operacje na obiektach trwałych

Zapisywanie	ISession.Save(obj) Wywołanie tej metody powoduje, że podany obiekt zostanie zapisany w bazie po raz pierwszy. Stosuje się ją po stworzeniu nowej instancji obiektu.
Usuwanie	ISession.Delete(obj) Ta metoda służy do usuwania obiektu trwałego z bazy danych. Obiekt pozostanie jednak w pamięci, ale będzie odłączony do sesji.
Aktualizacja	ISession.Update(obj) Wywołanie tej metody powoduje zaznaczenie, że dany obiekt musi być zsynchronizowany z bazą. Bardzo często jednak wywoływanie jej jawnie nie jest potrzebne, ponieważ NH potrafi sam wykryć zmiany i w razie potrzeby zapisać zmiany automatycznie.
(Zapis lub aktualizacja) LUB	ISession.SaveOrUpdate(obj) Wywołanie tej metody spowoduje, że nowy obiekt będzie zapisany, a istniejący

Dołączenie	uaktualniony (metoda łącząca dwie funkcje, może być stosowana dla wygodny). Dodatkowym przeznaczeniem tej funkcji jest dołączanie obiektów z innej sesji (np. zamkniętej) do aktualnej sesji.

Pobieranie obiektów z bazy

Pojedynczego obiektu według Id	Obj = ISession.Get<TYP>(id) Wywołanie tej metody powoduje załadowanie z bazy obiektu podanego typu, którego identyfikator jest równy id.
Pobieranie listy według kryteriów	Ilist<TYP> q = ISession.CreateCriteria(TYP)... Ta metod służy do pobierania listy obiektów, według zadanych kryteriów. Kryteria dodaje się jako kolejne wywołania metod na zwróconym obiekcie. Pobranie listy wymaga wywołania metody List<TYP>();
Pobieranie listy poprzez zapytanie HQL	Ilist<TYP> q = ISession.CreateQuery(string HQL) Ta metod służy do pobierania listy obiektów przy ykorzystaniu zapytania w języku HQL. Pobranie listy wymaga wywołania metody List<TYP>();

Strategie używania sesji w aplikacji WinForms

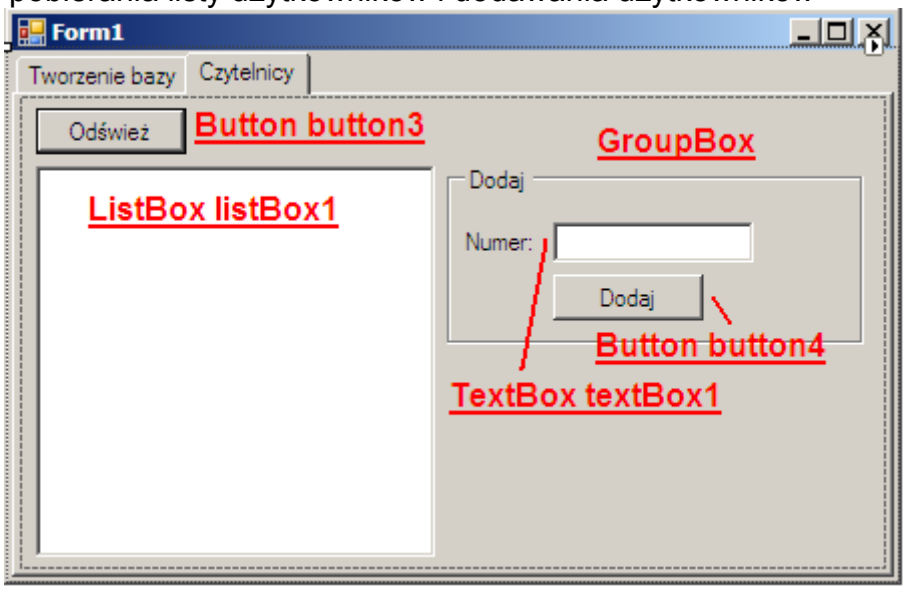
Podczas tworzenia architektury dla systemu, w którym ma być wykorzystany NHibernate, należy jednocześnie zaprojektować sposób używania sesji. Dla aplikacji lokalnych można wyróżnić trzy główne strategie:

- Jedna sesja dla aplikacji – w takim podejściu przy starcie aplikacji tworzy się sesję, a następnie wykorzystuje się ją przez cały czas życia aplikacji. Jest to najbardziej wygodny sposób używania sesji, jednak występuje problem rozrastania się w pamięci kontekstu sesji. Sesja po długim czasie działania takiej aplikacji może zająć całą dostępną pamięć komputera. Jest to spowodowane tym, że wszystkie załadowane obiekty będą ciągle przechowywane w pamięci. Jednocześnie jest duże prawdopodobieństwo występowania w pamięci obiektów, które nie są aktualne (w bazie ich stan może się zmienić przez inną aplikację).
- Nowa sesja na każdą wykonywaną grupę funkcji systemu – taki sposób wykorzystania sesji, że ogranicza się ją tylko do grupy operacji, wykonujących jedną funkcję systemu (feature) jest przeciwieństwem poprzednio opisanej strategii. Sesje są małe, bo mają krótki czas życia. Wadą tego rozwiązania jest to, że wszystkie obiekty załadowane w innych sesjach są niewidoczne. Powoduje to konieczność dołączania obiektów do sesji (poprzez metodę SaveOrUpdate(), co komplikuje metody operujące na obiektach. Ta strategia jest wykorzystana w przykładach zawartych w podrozdziale 2.6.2.
- Jedna sesja na przypadek użycia (use case) – jest to jedna z najbardziej popularnych strategii. Stanowi złoty środek pomiędzy sesją trwającą przez całe życie aplikacji, a sesją dla funkcji systemu. W tej strategii czas życia sesji ograniczony jest do przypadku użycia, co przekłada się najczęściej na jedno okno lub grupę okien współpracujących ze sobą. Takie podejście eliminuje wadę „sesji na funkcję systemu", która polegała na potrzebie dołączania obiektów do sesji, ponieważ wszystkie obiekty, których funkcja potrzebuje, będą dostępne w danej sesji (przypadek użycia nie zezwala na wyjście poza jego ramy). Jednocześnie czas życia

takiego przypadku użycia zazwyczaj jest na tyle krótki, że nie powinien spowodować rozrastania się kontekstu sesji.

2.6.2. Część praktyczna

a) Funkcje pobierania listy użytkowników i dodawania użytkowników



Rys. 2.6. Wygląd zakładki „Czytelnicy”

- otworzyć Form1 w trybie edycji wizualnej i umieścić na niej kontrolki tak jak pokazano na rysunki 2.6 (na formie należy umieścić przycisk button3, który będzie odpowiedzialny za odświeżanie listy czytelników oraz pole tekstowe textBox1 i przycisk button4, kontrolki te będą użyte przy dodawaniu nowego czytelnika);
- kliknąć dwukrotnie na przycisk „Odśwież” i w utworzoną metodę obsługi przycisku wypisać następujący kod:

```
private void button3_Click(object sender, EventArgs e)
{
ISession session = _pm.SessionFactory.OpenSession();
IList<Czytelnik> readers = session
.CreateCriteria(typeof(Czytelnik))
.List<Czytelnik>();
listBox1.DataSource = null;
listBox1.DataSource = readers;
}
```

- w trybie edycji kodu Form1 dodać poniższy kod do bloku using (przed namespace'em). Umożliwi to stosowanie krótszej formy zapisy odwołań do klas:

```
using NHibernate;
using NHTutorial.Domain;
```

- w trybie edycji wizualnej dwukrotnie kliknąć na przycisk „Dodaj” i w wygenerowaną metodę obsługi zdarzenia wpisać kod:

```
private void button4_Click(object sender, EventArgs e)
{
int numer;
bool ok = Int32.TryParse(textBox1.Text, out numer);
if (!ok) return;
ISession session = _pm.SessionFactory.OpenSession();
NHibernate - badanie możliwości i opracowanie przykładów wykorzystania technologii ORM 20
Czytelnik reader = new Czytelnik();
reader.Numer = numer;
session.Save(reader);
session.Flush();
```

```
session.Close();
}
```
- skompilować uruchomić i projekt. Sprawdzić czy po kliknięciu „Odśwież” zostaje pobrana lista czytelników dodanych do bazy w trakcie wstawiania danych testowych. Następnie wpisać nowy numer do pola „Numer” (np: 301) i kliknąć przycisk „Dodaj”, a następnie przycisk „Odśwież”. Po ponownym pobraniu listy powinien na niej znaleźć się nowy element;

b) Funkcja sprawdzania książek czytelnika

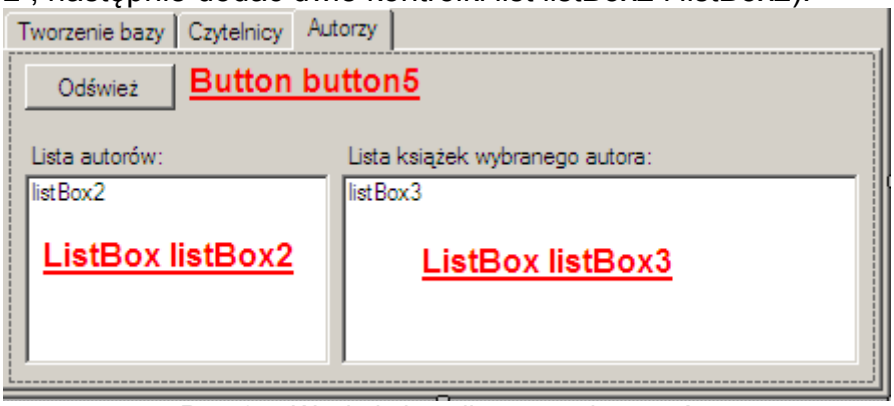
- W edytorze wizualnym zaznaczyć kontrolkę listBox1, następnie poprzez okno Properties-Events dodać obsługę zdarzenia „DoubleClick”. Przejść do edycji kodu i obsługę wygenerowanego zdarzenia uzupełnić według poniższego kodu:

```
private void listBox1_DoubleClick(object sender, EventArgs e)
{
Czytelnik reader = listBox1.SelectedItem as Czytelnik;
string message = String.Format("Czytelnik numer: {0}\nKsiążki:\n",
reader.Numer);
ISession session = _pm.SessionFactory.OpenSession();
session.SaveOrUpdate(reader);
if (reader.Wypozyczenia == null || reader.Wypozyczenia.Count == 0)
message += "brak";
else
foreach (Ksiazka k in reader.Wypozyczenia)
message += String.Format("- '{0}', {1}\n",
k.Tytul, k.Wydawca);
session.Close();
MessageBox.Show(message);
}
```
- Skompilować uruchomić i projekt. Po uruchomieniu programu i pobraniu listy dwukrotne

kliknięcie na element z kontrolki listBox1 spowoduje pokazanie okna informacyjnego zawierającego dane o wypożyczonych książkach przez wybranego użytkownika.

c) Funkcja wypisywania autorów i ich książek

- Dodać nową zakładkę do kontrolki TabControl tabControl1. Następnie przygotować jej wygląd tak jak jest to pokazane na rysunku 2.7 (dodać przycisk button5 i nazwać go „Odśwież”, następnie dodać dwie kontrolki list listBox2 i listBox2).



Rys. 2.7. Wygląd okna dla tworzenia autorów

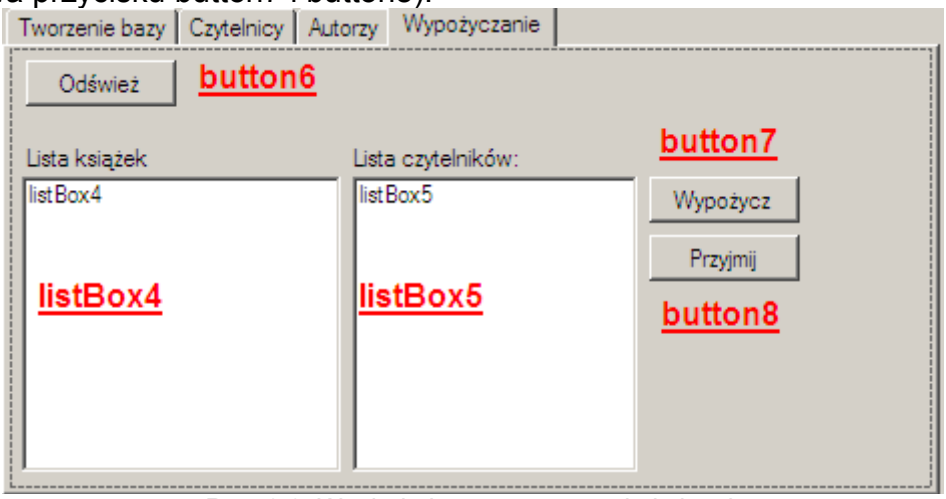
- kliknąć dwukrotnie na przycisku „Odśwież”, aby wygenerować obsługę zdarzenia. Kod metody uzupełnić następująco:

```
private void button5_Click(object sender, EventArgs e)
{
ISession session = _pm.SessionFactory.OpenSession();
IList<Autor> authors = session.CreateQuery("from Autor")
.List<Autor>();
listBox2.DataSource = null;
```

```
listBox2.DataSource = authors;
session.Close();
}
```
- Następnie przejść do trybu wizualnego i dodać obsługę zdarzenia „DoubleClick” dla kontrolki listBox2 (procedura dodania obsługi zdarzenia identyczna jak w punkcie 2.6.2.b). Wygenerowaną metodę uzupełnić następująco:
```
private void listBox2_DoubleClick(object sender, EventArgs e)
{
Autor author = listBox2.SelectedItem as Autor;
ISession session = _pm.SessionFactory.OpenSession();
session.SaveOrUpdate(author);
listBox3.DataSource = null;
listBox3.DataSource = author.Ksiazki;
session.Close();
}
```
- Skompilować uruchomić i projekt. Sprawdzić działanie dodanych funkcji poprzez kliknięcie przycisku „Odśwież” (pojawi się lista autorów), a następnie poprzez dwukrotne kliknięcie na element z listu listBox2 sprawdzić, czy w kontrolce listBox3 pojawi się lista książek wybranego autora;

d) Lista książek i wypożyczanie książek

- Dodać nową zakładkę do kontrolki TabControl tabControl1. Następnie przygotować jej wygląd tak jak jest to pokazane na rysunku 2.8 (dodać przycisk button6 i nazwać go „Odśwież”, następnie dodać dwie kontrolki list listBox4 i listBox5, ostatecznie dodać dwa przycisku button7 i button8).



Rys. 2.8. Wygląd okna wypożyczania książek

- Kliknąć dwukrotnie ma przycisk „Odśwież” i dodać następującą obsługę zdarzenia:
```
private void button6_Click(object sender, EventArgs e)
{
ISession session = _pm.SessionFactory.OpenSession();
IList<Ksiazka> books = session.CreateQuery("from Ksiazka")
.List<Ksiazka>();
IList<Czytelnik> readers = session.CreateQuery("from Czytelnik")
.List<Czytelnik>();
listBox4.DataSource = null;
listBox4.DataSource = books;
listBox5.DataSource = null;
listBox5.DataSource = readers;
session.Close();
}
```
- Kliknąć dwukrotnie ma przycisk „Wypożycz” i dodać następującą obsługę zdarzenia:

```
private void button7_Click(object sender, EventArgs e)
{
Ksiazka book = listBox4.SelectedItem as Ksiazka;
Czytelnik reader = listBox5.SelectedItem as Czytelnik;
if (book.AktualnyCzytelnik != null) {
MessageBox.Show("Wybrana książka jest już wypożyczona");
return;
}
ISession session = _pm.SessionFactory.OpenSession();
session.SaveOrUpdate(book);
session.SaveOrUpdate(reader);
book.AktualnyCzytelnik = reader;
session.Flush();
session.Close();
}
```

- Kliknąć dwukrotnie ma przycisk „Przyjmij” i dodać następującą obsługę zdarzenia:

```
private void button8_Click(object sender, EventArgs e)
{
```



```
Ksiazka book = listBox4.SelectedItem as Ksiazka;
Czytelnik reader = listBox5.SelectedItem as Czytelnik;
ISession session = _pm.SessionFactory.OpenSession();
session.SaveOrUpdate(book);
session.SaveOrUpdate(reader);
if (reader.Wypozyczenia.Contains(book) == false){
MessageBox.Show("Wskazany czytelnik nie ma wypożyczonej wskazanej książki");
return;
}
book.AktualnyCzytelnik = null;
session.Flush();
session.Close();
}
```

- Skompilować uruchomić i projekt. Sprawdzić działanie zaimplementowanych funkcji poprzez następujące testy:
- kliknąć przycisk „Odśwież” i sprawdzić, czy obie listy zostaną wypełnione;
- korzystając z zakładki „Czytelnicy” odszukać numer czytelnika, który nie ma wypożyczonej żadnej książki. Następnie wybrać z listy tego użytkownika i wybrać dowolną książkę. Kliknąć „Wypożycz”. Jeżeli książka jest już wypożyczona to pojawi się komunikat, w przeciwnym wypadku książka zostanie przypisana do użytkownika. Teraz poprzez zakładkę „Czytelnicy” można sprawdzić, że wybrany czytelnik ma wypożyczoną nową książkę.
- ponownie zaznaczyć książkę, która została przed chwilą wypożyczona i zaznaczyć dowolnego czytelnika. Następnie kliknąć „Wypożycz”. Powinien się pojawić komunikat, że książka jest już wypożyczona.
- zaznaczyć czytelnika, któremu wypożyczono książkę i zaznaczyć tą książkę, następnie kliknąć przycisk „Przyjmij”. Jeżeli nie pojawił się komunikat, to znaczy, że książka została przyjęta i jest wolna. Jeżeli pojawił się komunikat, to znaczy, że zaznaczony czytelnik nie ma wypożyczonej zaznaczonej książki.

3. Tworzenie systemu trójwarstwowego

W tym rozdziale zaprezentowana jest koncepcja wykorzystania biblioteki NHibernate w systemach trójwarstwowych.

3.1. Wstęp teoretyczny

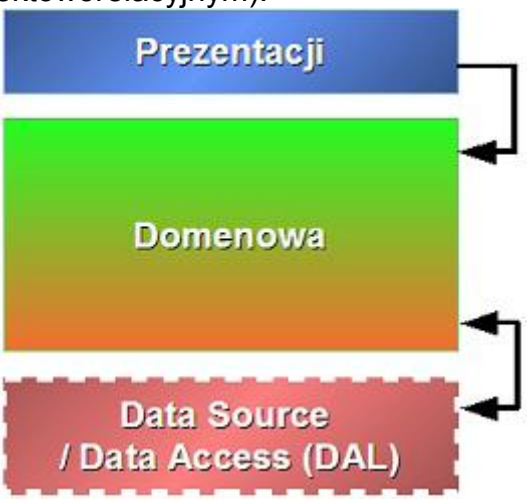
W tym rozdziale opisane jest tworzenie prostego systemu opartego o architekturę trójwarstwową przy wykorzystaniu biblioteki NHibernate w warstwie dostępu do danych. Dla ułatwienia cześć praktyczna opiera się na części kodów źródłowych stworzonych w poprzednim przykładzie („NHTutorial"). Aplikacja tworzona w tym rozdziale nazywa się „NHTutorial2" i jest bardzo uproszczona w porównaniu do poprzedniego przykładu, jednak dzięki temu wyraźniej widać różnice między architekturą dwuwarstwową a trójwarstwową.

Architektura twójwarstwowa

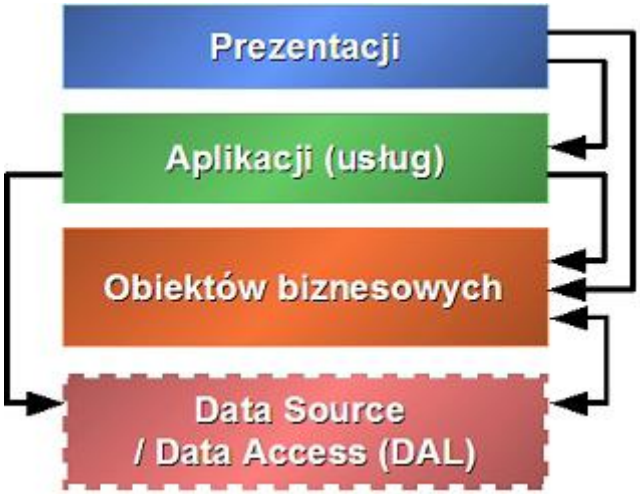
Wprowadzanie architektury trójwarstwowej do systemu ma na celu poprawienie jego struktury poprzez jasne sprecyzowanie zadań (odpowiedzialności) klas znajdujących się w odpowiedniej warstwie.

W standardowej strukturze trójwarstwowej (rys. 3.1) wyróżnia się:

- warstwę prezentacji – odpowiedzialna jest za wyświetlanie danych użytkownikowi oraz pobieranie od niego informacji zwrotnej. Dla inne warstwy nie powinny polegać na warstwie prezentacji, dlatego nie ma znaczenia czy jest ona wykonana w technologiach internetowych (np ASP.NET) czy lokalnych (WinForms);
- warstwę domenową, zwana także warstwą logiki biznesowej – odpowiedzialna jest za przetwarzanie wszelkiej logiki programu (reguły, procesy biznesowe, itp). Powinna to być warstwa jak najbardziej niezależna od innych, w szczególności nigdy nie powinna być zależna od warstwy prezentacji (np. poprzez referencje, otwieranie okien, itp);
- warstwa dostępu do danych (DAL) – odpowiedzialna jest za dostarczanie do systemu danych ze źródła danych (najczęściej bazy danych, pośredniczy wtedy także w mapowaniu obiektoworelacyjnym).



Rys. 3.1. Standardowe warstwy systemu trójwarstwowego



Rys. 3.2. System z wyodrębnioną warstwą usług

Warto zauważyć połączenia pomiędzy warstwami (rys. 3.1). Mogą się one zmieniać w zależności od systemu i jego skomplikowania. Dla bardzo złożonych projektów stosuje się rozdzielenie warstwy domenowej na warstwę aplikacji (zwaną też warstwą usług) oraz warstwę obiektów biznesowych.

Warstwa usług zawiera logikę operującą na wielu obiektach biznesowych na raz albo jest używana jako „punkt wejścia" dla warstwy prezentacji. Natomiast obiekty biznesowe to są encje, często uzupełnione o metody biznesowe (zwłaszcza jeżeli warstwa usług jest cienka).

Uwaga: dla ułatwienia w przykładowym projekcie „NHTutorial2" warstwa prezentacji „GUI" ma oprócz dostępu do warstwy „Domain" także dostęp do do warstwy danych „DAL".

Budowa warstwy DAL (Data Access Layer)

Warstwa dostępu do danych musi być dopasowana ściśle do źródła danych, na którym pracuje.

Najczęściej odpowiedzialna jest za mapowanie relacyjne, ale w systemach, które wykorzystują bibliotekę NHibernatem, zadanie to spada na tę właśnie bibliotekę.

Warstwa DAL składa się najczęściej z klas pomocniczych (typu PersistenceManager) oraz klas opartych o wzorzec aplikacyjny Data Access Object). Klasy DAO odpowiedzialne są za pośrednictwo pomiędzy bazą danych a modelem obiektowych dla konkretnej klasy (np: BooksDAO, z metodami NewBook() czy GetBookByAuthor()).

Uwaga: należy zauważyć, że dostęp (referencję) do biblioteki NHibernate w każdej architekturze powinna mieć tylko warstwa Data Access Layer!

Literatura dotycząca architektury systemów

- „Patterns of Enterprise Application Architecture", Fowler;
- „Applying UML and Patterns", Larman, 3rd Edition;
- http://java.sun.com/blueprints/corej2eepatterns/

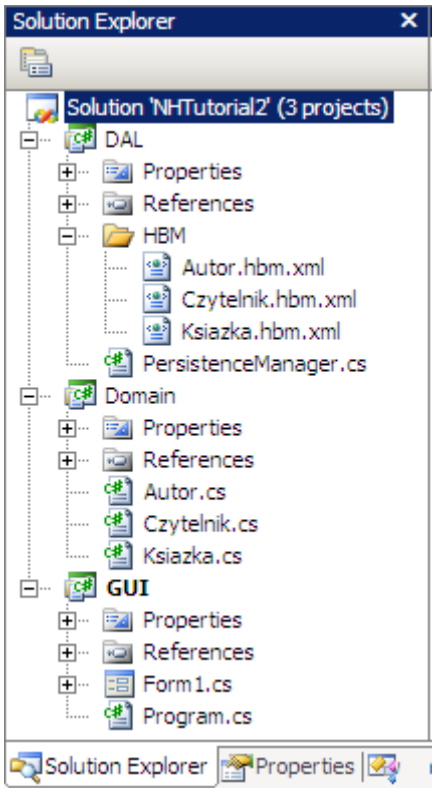
3.2. Część praktyczna

a) Tworzenie solution dla nowego projektu

- zamknąć poprzedni projekt;
- stworzyć nowy projekt aplikacji Windows Application z opcją „Create directory for solution" o nazwie „NHTutorial2" (jeżeli jako IDE używa się VC# 2005 Express to należy zapisać projekt, aby powyższa opcja była dostępna);

b) Tworzenie struktury projektów

- w oknie „Solution Explorer” zaznaczyć projekt „NHTutorial2” i zmienić jego nazwę na „GUI”;
- w tym samym oknie zaznaczyć „Solution” (element na samej górze) i dodać nowy projekt (prawym przyciskiem myszy Add->New Project). Jako typ nowego projektu wybrać „Class Library” a nazwę „DAL”. Ze stworzonego projektu usunąć plik „Class1.cs”;
- poprzedni punkt powtórzyć tworząc projekt „Domain”;
- uwaga: projekt GUI reprezentuje warstwę prezentacji w projekcie, a projekty DAL i Domain kolejno warstwę dostępu do danych (Data Acces Layer) i warstwę domenową (zawierającą logikę biznesową);
c) Przenoszenie kodu z poprzedniego solution „NHTutoria” do „NHTutorial2”
- uwaga: w rozdziale 2 instrukcji wykonano system dwuwarstwowy, klasy zawarte w nim zostaną wykorzystane w tym podpunkcie;
- z projektu „NHTutorial” skopiować do podprojektu „Domain” następujące pliki: Autor.cs, Czytelnik.cs i Ksiazka.cs. W widoku „Solution Explorer” zaznaczyć projekt „Domain” i dodać do niego wszystkie 3 skopiowane pliki (prawym przyciskiem myszy Add -> Existing Item);
- z projektu „NHTutorial” skopiować do podprojektu „DAL” katalog „HBM”. Następnie dodać ten katalog do projektu (np. poprzez „przeciągnij i upuść” z okna Windows Explorera do „Solution Explorer” na nazwę projektu DAL). Następnie koniecznie trzeba ustawić tryb „Build Action” na „Embedded Resource” (tak jak w poprzednim projekcie);
- z projektu „NHTutorial” skopiować plik „PersistenceManager.cs” do podprojektu „DAL” i poprzez Solution Explorer dodać go do projektu „DAL;
d) Referencja w projektach
- dodać referencję do biblioteki NHibernate.dll dla projektu „DAL”;
- w projekcie „GUI” dodać referencję do projektu „DAL” oraz „Domain” (w oknie „Add references” wybrać zakładkę „Projects” i wybrać oba projekty);
- w projekcie „DAL” dodać referencję do projektu „Domain” (analogicznie jak w poprzednim punkcie);



Rys. 3.3. Wygląd solution po przygotowaniu struktury

e) Sprawdzenie struktury projektu

- wygląd struktury projektu w oknie Solution Explorer powinien być identyczny jak na rys. 3.3;
- projekt powinien się kompilować bez błędów;

f) Poprawki w plikach HBM oraz PersistenceManager

- we wszystkich trzech plikach HBM.XML należy zmienić domyślne Assembly na „Domain". Poprawka jest potrzebna, ponieważ pliki HBM zostały przeniesione do projektu o innej nazwie. Zatem początek plików HBM powinnien wyglądać następująco:

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

- w pliku PersistenceManager.cs trzeba poprawić stałą HBM_ENTITY_FILES tak, aby określała nowe położenie plików HBM:


```
private readonly string[] HBM_ENTITY_FILES = {
"NHTutorial2.DAL.HBM.Autor.hbm.xml",
"NHTutorial2.DAL.HBM.Ksiazka.hbm.xml",
"NHTutorial2.DAL.HBM.Czytelnik.hbm.xml",
};
```

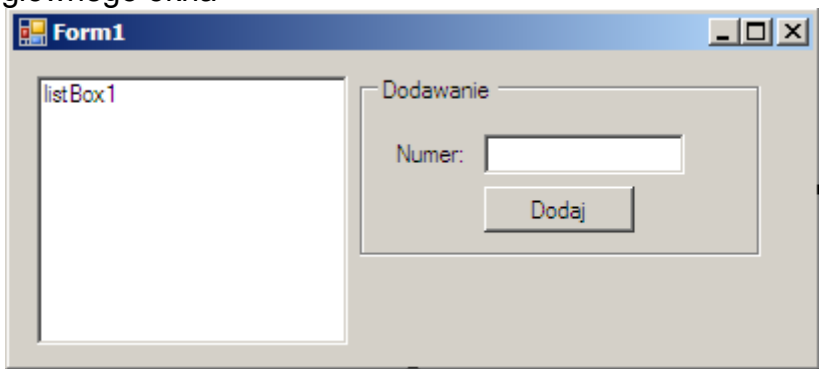
g) Tworzenie pliku DAO

- do projektu „DAL" dodać nową klasę i nazwać ją „CzytelnikDAO". Plik uzupełnić według przykładu:

```
using System.Collections.Generic;
using NHTutorial.Domain;
using NHibernate;
using NHTutorial;
namespace DAL {
public class CzytelnikDAO
{
private PersistenceManager _pm = PersistenceManager.Instance;
```

```
public IList<Czytelnik> GetAllReaders() {
ISession session = _pm.SessionFactory.OpenSession();
IList<Czytelnik> readers = session
.CreateCriteria(typeof(Czytelnik))
.List<Czytelnik>();
session.Close();
return readers;
}
public void CreateReader(int numer) {
Czytelnik reader = new Czytelnik();
reader.Numer = numer;
ISession session = _pm.SessionFactory.OpenSession();
session.Save(reader);
session.Flush();
session.Close();
}
}
}
```

h) Tworzenie głównego okna



Rys. 3.4. Wygląd okna głównego

- otworzyć „Form1” w trybie edycji wizualnej. Ustawić na niej kontrolki tak jak na rys. 3.4 (jedna kontrolka listBox1 do wypisywania listy czytelników i kontrolki textBox1 oraz button1 do dodawania nowych);
- otworzyć okno Form1 w trybie edycji kodu. Do klasy Form1 dodać pole _czytelnikDAO:

```
public partial class Form1 : Form
{
private DAL.CzytelnikDAO _czytelnikDAO = new DAL.CzytelnikDAO(); //<<<
...
```

- dodać nowe zdarzenie dla Form1 o nazwie „Load” (aby dodać to zdarzenie wystarczy dwukrotnie kliknąć w dowolne puste miejsce na formie w trybie edycji wizualnej). Następnie uzupełnić je według poniższego listingu:

```
private void Form1_Load(object sender, EventArgs e)
{
listBox1.DataSource = _czytelnikDAO.GetAllReaders();
}
```

- dodać obsługę zdarzenia OnClick dla przycisku button1 i uzupełnić jego kod następująco:

```
private void button1_Click(object sender, EventArgs e)
{
int numer;
if (Int32.TryParse(textBox1.Text, out numer) && numer > 0)
{
_czytelnikDAO.CreateReader(numer);
textBox1.Text = String.Empty;
```

```
}
else
MessageBox.Show("Podany numer nie jest prawidłowy");
listBox1.DataSource = _czytelnikDAO.GetAllReaders();
}
```

i) Testowanie stworzonej aplikacji

- Skompilować uruchomić i projekt. Sprawdzić działanie zaimplementowanych funkcji poprzez następujące testy:
- po uruchomieniu aplikacji na kontrolce listBox1 automatycznie powinna pojawić się lista czytelników;
- próba dodania czytelnika z numerem mniejszym niż 1 lub z numerem nie będącym liczbą całkowitą nie powiedzie się;
- po poprawnym dodaniu nowego użytkownika powinien on od razu pojawić się na liście użytkowników;

4. Podsumowanie

Przedstawione sprawozdanie pokazało jak łatwe jest wykorzystanie biblioteki NHibernate zarówno w prostych aplikacjach, jak i w większych systemach trójwarstwowych. Po zaprezentowanych przykładach widać, że tworzenie warstwy dostępu do danych (DAL) przy wykorzystaniu systemu ORM jest o wiele szybsze niż standardowe podejście, bazujące na ADO.NET.

5. Zadanie Indywidualne

Stwórz model aplikacji bazodanowej z wykorzystaniem technologii NHibernate realizującej funkcje manipulowania z obiektami klas poprzedniego projektu z kursu ASP.NET.