Laboratorium 12 – MS SQL Server 2008

Temat: Obiektowo-relacyjne odwzorowanie, LINQ

Opracowanie: A.Dydejczyk

Tematyka realizowana w trakcie laboratorium:

1. Technologia LINQ
2. Składnia zapytania LINQ
3. LINQ to Objects
4. LINQ to SQL
5. LINQ to SQL w Visual Studio
6. LINQ wspiera operatory SQL dla zapytań tworzonych na dokumentach XML.
7. Przetwarzanie danych w bazie AdvantureWorks 2008.

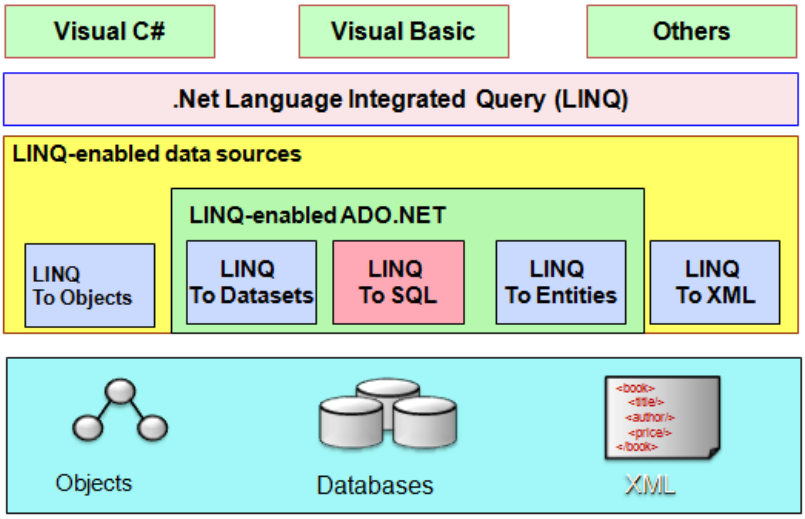
1. Czym jest LINQ?

LINQ (Language Integrated Query – zapytania zintegrowane) to technologia zapytań zintegrowanych z językiem programowania, pozwalająca na opracowanie zapytań dotyczących lokalnych kolekcji obiektów i zdalnych źródeł danych, a przy tym podlegających kontroli typów. Podstawowymi jednostkami danych w LINQ są kolekcje i elementy. Kolekcja danych odpytywana przez LINQ musi implementować interfejs IEnumerable<>, a element to dowolny pojedynczy element takiej kolekcji. Dostęp do obiektów wewnątrz programu oraz do obiektów baz danych jest zrealizowany w jednolity sposób. Składnia zapytań LINQ umożliwia odpytywanie różnych źródeł danych: obiektów, baz danych SQL, dokumentów XML, encji itd. LINQ zawiera pełne wsparcie dla transakcji, widoków oraz procedur przechowanych. Zapytania LINQ stają się częścią języka wspierającego .NET (C#, VB itd.).Wszystkie operatory zapytań są implementowane jako statyczne metody rozszerzające klasy Enumerable. Poprawne działanie LINQ wymaga znajomości całej mapy bazy danych. Zapytanie LINQ zwraca kolekcję z przestrzeni nazw typów ogólnych. Kolekcja ta może być modyfikowana, a następnie zwrócona do źródła. Dzięki temu zachowywana jest pełna kontrola typów danych i ich konwersji w poszczególnych mechanizmach pośredniczących w pobieraniu danych. Dane mogą pochodzić z:

- **zwykłych obiektów**
  LINQ to Objects – umożliwia pracę na zwykłych obiektach nie mających swojej reprezentacji w bazie danych. Przetwarzanie dotyczy obiektów implementujących interfejs IEnumerable<T>.
- **bazy danych**
  LINQ to SQL – pozwala na wykorzystaniu technologii LINQ jako ORM i podpięcia się do źródeł bazodanowych, w pełni wspiera m.in. procedury składowane i wyzwalacze.
- **dokumentów XML**
  LINQ to XML

W ramach LINQ można stosować operacje takie jak: projekcje lub złączenia. LINQ zapożyczył z

języka SQL sposób budowania zapytań.



Rys.1. Architektura LINQ

2. Składnia zapytania LINQ

```
  from id in source
{ from id in source |
  join id in source on expr equals expr [ into id ] |
  let id = expr |
  where condition |
  orderby ordering, ordering, … }
  select expr | group expr by key
[ into id query ]
```

Rys. 2 Składnia polecenia LINQ

3. LINQ to Objects

Tworzymy projekt w MS Visual Studio 2008 aplikacji konsolowej:
New Project → Visual C# → ConsoleApplication

W projekcie utworzymy klasę „Samochod” (Add → New Item → Class):

```
class Samochod
{
   public int ID { get; set; }
```

```
    public int IDMarka { get; set; }
    public String Kolor { get; set; }
}
```

Następnie w metodzie Main klasy Program dopisujemy inicjalizację listy, która jest klasą generyczną List<T> implementującą IEnumerable<T>:

```
List<Samochod> samochody = new List<Samochod> {
    new Samochod() { ID = 1, IDMarka = 1, Kolor = "Czarny"},
    new Samochod() { ID = 2, IDMarka = 2, Kolor = "Niebieski"}
};
```

Dalej w metodzie Main tworzymy poniższe zapytanie, gdzie **From** określa źródło danych, **s** reprezentuje obiekt w kolekcji **samochody, where** następuje warunek, zaś po **select** wskazujemy które dane nas interesują do przetworzenia:

```
var query = from s in samochody where s.ID == 1 select s.IDMarka + ", " + s.Kolor ;

foreach (string q in query)
{
    Console.WriteLine(q);
}
```

Możemy przystąpić do uruchomienia naszego programu:
Debug → Start Without Debugging (Ctrl + F5).

Alternatywnie w przypadku braku opcji „Start Without Debugging” dopisujemy na końcu metody Main:

```
Console.ReadKey();
```

Alternatywnie zamiast poprzedniego zapytania, moglibyśmy skorzystać z poniższego zapytania wykorzystującego typ anonimowy:

```
var query = from s in samochody where s.ID == 1 select new {s.IDMarka, s.Kolor} ;

Console.WriteLine(query.First().IDMarka + ", " + query.First().Kolor);
```

Zapytanie tego typu przypomina dosyć wyrażenia znane z SQL. Gdy kompilator napotka zapytanie przetwarza je do wywołań metod C#.

LINQ udostępnia API znane jako Standard Query Operators (**SQO**s).
Metody te należą do klasy statycznej System.Linq.Enumerable.
Poniżej lista dostępnych metod w formie tabeli – opis poszczególnych dostępny jest na stronie firmy Microsoft: https://msdn.microsoft.com/pl-pl/library/system.linq.enumerable(v=vs.110).aspx

Typ operacji:	Nazwa operacji:
Aggregation	Aggregate, Average, Count, LongCount, Max, Min, Sum
Conversion	Cast, OfType, ToArray, ToDictionary, ToList, ToLookup, ToSequence
Element	DefaultIfEmpty, ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault

Equality	EqualAll
Generation	Empty, Range, Repeat
Grouping	GroupBy
Joining	GroupJoin, Join
Ordering	OrderBy, ThenBy, OrderByDescending, ThenByDescending, Reverse
Partitioning	Skip, SkipWhile, Take, TakeWhile
Quantifiers	All, Any, Contains
Restriction	Where
Selection	Select, SelectMany
Set	Concat, Distinct, Except, Intersect, Union

Tabela 1. Lista metod dostępna w ramach LINQ

Przykład użycia operacji JOIN:

Dopisujemy klasę Marka.

```
class Marka
{
   public int ID { get; set; }
   public String Nazwa { get; set; }
 }
```

Inicjalizujemy listę marek.

```
List<Marka> marki = new List<Marka> {
   new Marka { ID = 1, Nazwa = "Fiat"},
   new Marka { ID = 2, Nazwa = "BMW"}
};
```

W metodzie Main budujemy zapytanie:

```
var query2 = from s in samochody join m in marki on s.IDMarka equals m.ID select m.Nazwa +
" " + s.Kolor;

foreach (string q2 in query2)
{
   Console.WriteLine(q2);
}
```

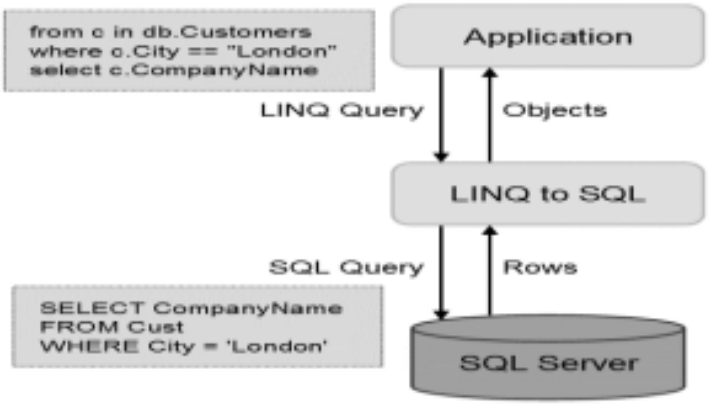
Zadanie 1

Należy zbudować 3 zapytania przy pomocy nie użytych do tej pory operatorów (Tabela 1).

4. LINQ to SQL

Trzy kroki wymagane do pracy z bazą danych.

1. Stworzenie klas dla tabel w bazie danych, udekorowanie ich odpowiednimi atrybutami (klasy te zwykle nazywane są encjami).
2. Udekorowanie pól w klasie w celu wyjaśnienia LINQ jak ich użyć.
3. Stworzenie obiektu DataContext pośredniczącego pomiędzy tabelami bazy danych a klasami mapowanymi na te tabele.



Rys.3. Przetwarzanie zapytania LINQ to SQL

Utworzenie klas mapowanych.

Tworzymy nowy projekt (New Project → Visual C# → ConsoleApplication) zawierający klasy mapujące dla encji „Marka” i encji „Samochod” zawierającej informację o relacji względem Marki (Association).

Aby móc korzystać z atrybutów Table i Column w MS VS 2008 należy do projektu dodać referencję System.Data.Linq (References → Add Reference).

Klasa „Marka”:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Linq.Mapping;
using System.Data.Linq;

[Table(Name="Marka")]
class Marka
{
   private int _ID;
   private String _Nazwa;
```

```
    [Column(Name="ID", Storage="_ID", DbType="int NOT NULL IDENTITY",
IsPrimaryKey=true, IsDbGenerated=true)]

    public int ID
    {
        get { return _ID; }
        set { _ID = value; }
    }

    [Column(Name = "Nazwa", Storage = "_Nazwa", DbType = "nvarchar(30)")]
    public String Nazwa
    {
        get { return _Nazwa; }
        set { _Nazwa = value; }
    }

    public override string ToString()
    {
        return "[" + ID + "] " + Nazwa;
    }
}
```

Klasa „Samochod”:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Linq.Mapping;
using System.Data.Linq;

[Table(Name = "Samochod")]
class Samochod
{
    private int _ID;
    private int _IDMarka;
    private String _Kolor;
    private EntityRef<Marka> _Marka;

    [Association(Storage = "_Marka", ThisKey = "IDMarka", OtherKey = "ID")]
    public Marka Marka
    {
        set
        {
            _Marka.Entity = value;
        }
        get
        {
            return _Marka.Entity;
        }
    }

    [Column(Name = "ID", Storage = "_ID", DbType = "int NOT NULL IDENTITY", IsPrimaryKey
= true, IsDbGenerated = true)]
    public int ID
    {
        get { return _ID; }
        set { _ID = value; }
    }
```

```
    [Column(Name = "IDMarka", Storage = "_IDMarka", DbType = "int NOT NULL")]
    public int IDMarka
    {
        get { return _IDMarka; }
        set { _IDMarka = value; }
    }

    [Column(Name = "Kolor", Storage = "_Kolor", DbType = "nvarchar(30)")]
    public String Kolor
    {
        get { return _Kolor; }
        set { _Kolor = value; }
    }

    public override String ToString()
    {
        return "[" + ID + "] " + Kolor + " " + Marka.Nazwa;
    }
}
```

Tworzenie kontekstu

DataContext jest obiektem typu System.Data.Linq.DataContext, wspiera przetwarzanie i aktualizowanie obiektów dostępnych dla LINQ. Obsługuje połączenia z bazą danych i implementuje SQO dla dostępu bazodanowego.

Na początek zdefiniujemy konfigurację połączenia z bazą za którą będzie odpowiedzialny obiekt **connString**, gdzie **Data Source** określa serwer bazodanowy (w naszym przypadku lokalny (' **.** '), **Initial Catalog** określa bazę z którą będziemy pracować:

```
String connString = @"
    Data Source=.;
    Initial Catalog=Samochody;
    Integrated Security=True
";
```

Utworzenie kontekstu służącego do połączenia z bazą danych:

```
DataContext sDataContext = new DataContext(connString);
Table<Samochod> Samochody = sDataContext.GetTable<Samochod>();
```

Następnie sprawdzamy czy baza danych istnieje, jeżeli tak to ją usuwamy a następnie tworzymy od nowa.

```
if (sDataContext.DatabaseExists())
{
    Console.WriteLine("Deleting old database...");
    sDataContext.DeleteDatabase();
}
sDataContext.CreateDatabase();
```

Zadanie 2

Tworzymy instancje typu Samochod, które następnie wrzucamy do bazy, wywołujemy metodę obiektu odpowiadającego tabeli:

```
Samochody.InsertOnSubmit(samochod);
```

Aby zachować zmiany w bazie należy zatwierdzić zmiany w kontekście:

```
sDataContext.SubmitChanges();
```

Zadanie 3

Zapytania analogiczne do tych z LINQ to Objects wykonujemy na instancji obiektu reprezentującego tabelę. Na tym etapie instancja obiektu posiada już dostęp do danych z tabeli.

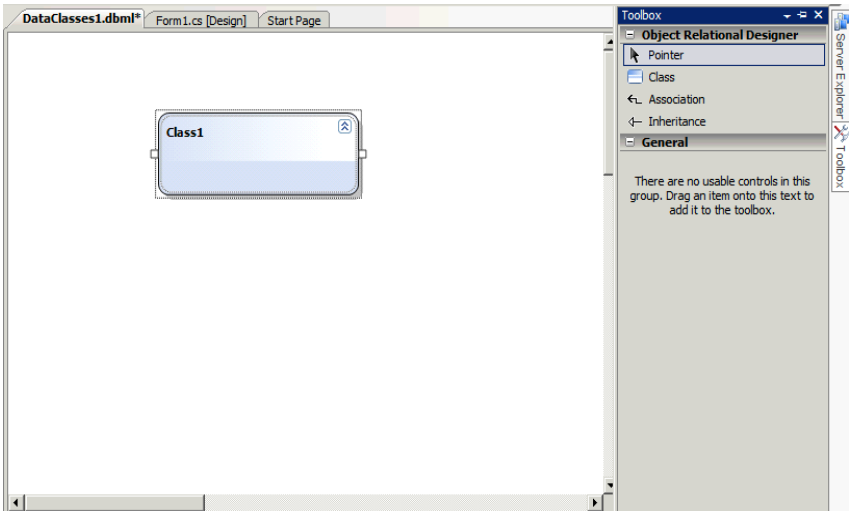
W tym momencie powinniśmy mieć już tabele wypełnione danymi.
Następnie należy skonstruować zapytanie wykorzystujące powiązanie klas.

Uwaga: Za pomocą „SQL Server Management Studio" możemy sprawdzić czy nasza baza się rzeczywiście utworzyła.

5. LINQ to SQL w Visual Studio 2008

1. Otwieramy tworzenie nowego projektu w Microsoft Visual Studio 2008 (File → New → Project).
2. Wybieramy rodzaj projektu (Visual C# → Windows → Windows Form Application) i nadajemy nazwę.
3. Dodajemy do projektu (Add → New Item) klasę zgodną z szablonem **LINQ to SQL classes** o nazwie **Employee**.

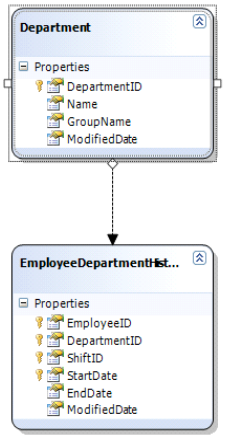
W ramach utworzonej klasy możemy wykorzystać dostępne narzędzia (zakładki) do tworzenia zawartości klasy: View → Toolbox - powinna pojawić się zakładka Toolbox umożliwiająca graficzne dodawanie klas modelu a następnie definiowanie pól (właściwości) klas (rys.4).



Rys.4. Widok: Linq to SQL classes designer

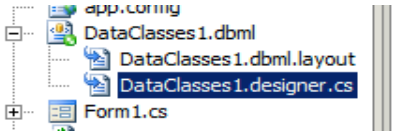
W ramach narzędzia można skorzystać z **Linq to SQL Classes Designer,** który umożliwia tworzenie modelu na podstawie bazy danych.
W tym celu należy otworzyć Server Explorer (View → Server Explorer). Następnie wybieramy połączenie z bazą danych (Connect to Database) Adventure Works. Po nawiązaniu połączenia przeciągamy do okna z zakładką pliku „dbml" tabelę Department, a następnie EmployeeDepartmentHistory.

W tym momencie powinniśmy mieć reprezentację klas powiązanych relacją w oparciu o zdefiniowane klucze (rys.5).



Rys.5. Powiązanie pomiędzy tabelami.

W celu obejrzenia wygenerowanego kodu klas oglądamy plik „cs" (rys.6) (należy zauważyć nazwę na kontekst udostępnianych klas modemu - **EmployeeDataContext)** .



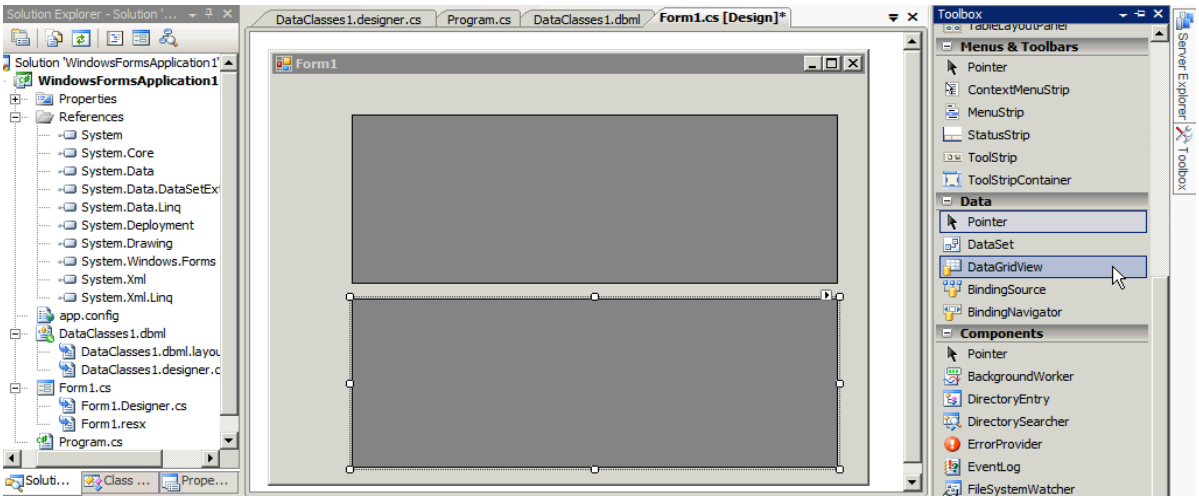
Rys.6. Plik „cs" z wygenerowanymi klasami.

Na koniec opracujemy formularz aplikacji do prezentacji danych pobranych z bazy danych. W tym celu do okna aplikacji (Form1) dodamy dwa obiekty typu DataGridView nazwane odpowiednio dgDepartment i dgEmpDepartmentHistory (rys.7). Po utworzeniu okien w ramach formularza otwieramy plik Form1.cs (klikamy dwa razy na graficznej reprezentacji formularza) i uzupełniamy kod klasy poniższymi liniami kodu.

W konstruktorze klasy Form1 tworzymy instancję obiektu EmployeeDataContext:

```
private EmployeeDataContext db;
public Form1()
{
        InitializeComponent();
        db = new EmployeeDataContext();
```

```
}
```



Rys.7. Dodanie obiektów typu DataGridView

W następnym kroku dodamy obsługę załadowania wyświetlanych tabelek.
Pod powyższym konstruktorem dopisujemy metodę Load – prawidłowo realizujemy to poprzez dwukrotne kliknięcie w obszar okna Form1 co spowoduje utworzenie pustej metody Form1_Load:

```
private void Form1_Load(object sender, EventArgs e)
{
        var query = from d in db.Departments select d;

        dgDepartment.DataSource = query;
        dgEmpDepartmentHistory.DataSource = dgDepartment.DataSource;
        dgEmpDepartmentHistory.DataMember = „EmployeeDepartmentHistories”;
}
```

Na koniec uruchamiamy aplikację (ctrl+F5).

6. LINQ wspiera operatory SQL dla zapytań tworzonych na dokumentach XML.

Pracujemy w nowym projekcie konsolowym Visual C#.

Tworzymy plik (Add New Item → Visual C# → XML File) **pracownicy.xml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<osoby>
        <osoba>
                <id>1</id>
                <imie>Andrzej</imie>
                <nazwisko>Zawada</nazwisko>
                <idrola>1</idrola>
        </osoba>
        <osoba>
                <id>2</id>
```

```
                <imie>Jan</imie>
                <nazwisko>Nowak</nazwisko>
                <idrola>2</idrola>
        </osoba>
        <osoba>
                <id>3</id>
                <imie>Monika</imie>
                <nazwisko>Kowalska</nazwisko>
                <idrola>2</idrola>
        </osoba>
        <osoba>
                <id>4</id>
                <imie>John</imie>
                <nazwisko>Smith</nazwisko>
                <idrola>1</idrola>
        </osoba>
 <!--Role-->
        <rola>
                <id>1</id>
                <nazwa>Szef</nazwa>
        </rola>
        <rola>
                <id>2</id>
                <nazwa>Pracownik</nazwa>
        </rola>
 <!--zarobki-->
        <zarobki>
                <idosoba id="1" rok="2004" zarobki_roczne="60000" />
                <idosoba id="1" rok="2005" zarobki_roczne="45000" />
                <idosoba id="2" rok="2004" zarobki_roczne="73000" />
        </zarobki>
</osoby>
```

W metodzie Main wczytujemy plik XML do obiektu (w miejsce @"pracownicy.xml" należy podać pełną ścieżkę do pliku xml, lub ścieżkę względem obecnego położenia np. ../../pracownicy.xml):

```
XDocument xml = XDocument.Load(@"pracownicy.xml");
```

Następnie tworzymy zapytanie:

```
var query = from o in xml.Elements("osoby").Elements("osoba")
            where (int)o.Element("id") == 1
            select o;

foreach(var record in query)
{
 Console.WriteLine("Osoba: {0} {1}",
 record.Element("imie").Value,
 record.Element("nazwisko").Value);
}
```

W celu przetworzenia bezpośrednio elementów osoba przy pominięciu elementu osoby użyjemy:

```
XElement xml2 = XElement.Load(@"pracownicy.xml");
var query2 = from o in xml2. Descendants ("osoba")
            where (int)o.Element("id") == 1
            select o;
```

```
foreach(var record in query2)
{
  Console.WriteLine("Osoba: {0} {1}",
  record.Element("imie"),
  record.Element("nazwisko"));
}
```

Zadanie 4

Stworzyć zapytanie wypisujące zarobki z atrybutem rok = 2004.

```
XElement xml3 = XElement.Load(@"pracownicy.xml");
var query3 = from o in xml3.Descendants("osoba")
                    join z in xml3.Descendants("idosoba")
                    on (int)o.Element("id") equals (int)z.Attribute("id")
                    select new {Imie=o.Element("imie").Value,
                                Nazwisko=o.Element("nazwisko").Value,
                                Kwota=z.Attribute("zarobki_roczne").Value};

foreach(var record in query3)
{
 Console.WriteLine("Osoba: {0} {1}, Salary {2}",
 record.Imie,
 record.Nazwisko,
 record.Kwota);
}
```

LINQ to XML wpiera także tworzenie dokumentów XML.
W celu utworzenia elementu XML używamy (zawartością może być inny zagnieżdżony element):

```
new XElement(”nazwa_elementu”,”zawartość_elementu”, ...)
```

Analogicznie tworzymy atrybuty:

```
new XAttribute(”nazwa_atrybutu”,”wartość_atrybutu”)
```

Zadanie 5

Napisać kod tworzący XML-a z obszerniejszą zawartością (jak np. pracownicy.xml).

Wskazówka w celu wypisania XML-a wykorzystamy:

```
Xdocument xml = new Xdocument(...);

System.IO.StringWriter sw = new System.IO.StringWriter();
xml.Save(sw);
Console.WriteLine(sw);
```

7. Przetwarzanie danych w bazie AdvantureWorks 2008.

Rozwijamy węzły:

AdventureWorks2008 -> Programability -> Types -> XML Schema Collection

Wykorzystując polecenie T-SQL w ramach konkretnej bazy danych:

```
SELECT * FROM sys.xml_schema_collections;

SELECT xml_schema_namespace(N'Person',N'IndividualSurveySchemaCollection')
GO
```

Znając schema znamy nazwy i typy danych umieszczonych w dokumentach xml. Znamy także relacje między danymi. Kilka przykładów wyrażeń LINQ.

Przykład 1.

Zaczynamy od przykładu z XPath.

```
SELECT Demographics.value('declare default element
namespace"http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/IndividualSurvey";
(/IndividualSurvey/YearlyIncome)[1]','varchar(250)')FROM Person.Person;
```

Dla porównania fragment kodu w C# z wykorzystaniem LINQ.

```
XNamespace nsp = "http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/IndividualSurvey";
        var queryp = from person in db.Persons
                     select new { demographics = person.Demographics };
        var val = new List<string>();
        foreach (var d in queryp)
        {
                var qry = from name in ( (XElement) d.demographics).Descendants(nsp +
        "YearlyIncome")
                select name.Value;
                foreach (var q in qry)
                {
                        val.Add(q);
                }
        }
        dgvEmployee.DataSource = val.ConvertAll(x => new { Value = x });
```

Przykład 2.

```
SELECT Demographics.value('declare default element namespace
"http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey";
(/IndividualSurvey/TotalChildren)[1][. > 1]','varchar(250)')
FROM Person.Person
```

Kod C# z LINQ

```
var queryp = from person in db.Persons
                     select new { demographics = person.Demographics };
```

```
var doc = new List<XElement>();
var val = new List<string>();
foreach (var d in queryp)
{
        var qry = from n in ((XElement)d.demographics).Elements(nsp + "TotalChil-
dren")
                where int.Parse(n.Value) > 1
                select n.Value;
        foreach (var q in qry)
        {
                val.Add(q);
        }
}
dgvEmployee.DataSource = val.ConvertAll(x => new { Value = x });
```

Przykład 3.

```
SELECT xml_schema_namespace(N'HumanResources',N'HRResumeSchemaCollection')
```

```
XNamespace ns = "http://schemas.microsoft.com/sqlserver/2004/07/adventure-
works/Resume";
var query = from candidate in db.JobCandidates
        select new { resume = candidate.Resume, beid = candidate.BusinessEntityID };
var val = new List<string>();
foreach (var v in query)
{
        XElement elem = v.resume;
        var qry = from name in elem.Elements(ns + "Name").Elements(ns + "Name.Last")
        select name.Value;
        foreach (var q in qry)
        {
                val.Add(q);
        }
}
dgvEmployee.DataSource = val.ConvertAll(x => new { Value = x });
```

Przykład 4.

```
var doc = new List<XElement>();
var val = new List<string>();
foreach (var v in query)
{
        XElement elem = v.resume;
        var qry = from name in elem.Elements(ns + "Employment")
                select name;
        foreach (var q in qry)
        {
                doc.Add(q.Elements(ns + "Emp.StartDate").First());
                string s = q.Elements(ns + "Emp.StartDate").First().Value;
                val.Add(s);
        }
}
dgvEmployee.DataSource = val.ConvertAll(x => new { Value = x });
```

Przykład 5. Modyfikacja zawartości.

```
XElement newAddr = new XElement(ns + "Address");
XElement newTel = new XElement(ns + "Addr.Telephone",
```

```
            new XElement(ns + "Telephone",
                  new XElement(ns + "Tel.IntlCode", "48"),
                  new XElement(ns + "Tel.AreaCode", "12"),
                  new XElement(ns + "Tel.Number", "601776655")
            )
);
newAddr.Add(new XElement(ns + "Addr.Type", "Local"));
newAddr.Add(new XElement(ns + "Addr.Street", "Reymonta"));
newAddr.Add(new XElement(ns + "Addr.Location",
            new XElement(ns + "Location",
                  new XElement(ns + "Loc.CountryRegion", "PL"),
                  new XElement(ns + "Loc.State", "MA"),
                  new XElement(ns + "Loc.City", "Cracow")
            )
            )
);
newAddr.Add(new XElement(ns + "Addr.PostalCode", "30-000"));
newAddr.Add(newTel);

var query = from candidate in db.JobCandidates
            select new { resume = candidate.Resume, beid = candi-date.JobCandidateID };
var queryNN = from q in query select new { elem = (XElement)q.resume, id = q.beid };
var list = queryNN.ToList();
var queryEmail = from em in list
            where ((string)em.elem.Element(ns + "EMail")) == "Stephen@example.com"
            select new { email = em.elem, id = em.id };
int idsel = (int)queryEmail.Single().id;
var queryUpdate = (from candidate in db.JobCandidates
            where candidate.JobCandidateID == idsel
            select candidate).Single(); //.Resume;

XElement src = queryUpdate.Resume;
src.Element(ns + "Address").AddBeforeSelf(newAddr); //Add(newAddr);
queryUpdate.Resume = new XElement(src);
try
{
            db.SubmitChanges();
}
catch (Exception ex)
{
            String s = ex.Message;
}
```

Zadanie 6

W tabeli [Person].[Person] bazy AdventureWorks2008 kolumna Demographics zawiera dokumenty XML. Skonstruować pytanie LINQ o wartości /IndividualSurvey/NumberChildrenAtHome. Pytanie powinno zwrócić nazwiska osób dla których wartość NumberChildrenAtHome wynosi 5.