MICROSOFT LINQ

Ognjen Borovina, *Elektrotehnički fakultet, Istočno Sarajevo*

Sadržaj: *U ovom radu uveli smo LINQ i pokazali osnove njegovog rada. Pokazali smo kako možemo postavljati upite nad različitim tipovima podataka koristeći jedinstvenu sintaksu koja je ugrađena u osnovne programske jezike kao što su C# i Viusal Basic. Razmotrili smo prednosti koje nudi jezik integracije, uključujući i deklarativno programiranje, type checking, te transparentnost kroz različite vrste sistema. Ukratko smo predstavili LINQ implementaciju dostupnu u in .NET 3.5—LINQ to Objects, LINQ to ADO.NET and LINQ to XML. Navodili smo primjere za LINQ iz konkretne aplikacije Dnevnik nastavne aktivnosti. Ova aplikacija je pisana za Elektrotehnički fakultet u Istočnom Sarajevu. Lako je prilagodljiva i za ostale fakultete univerziteta.*

Abstract: *In this page, we introduced LINQ and discussed how it works. We also examined how different data domains can be queried and manipulated by using a uniform syntax that is integrated into current mainstream programming languages such as C# and Visual Basic. We took a look at the benefits offered by language integration, including declarative programming, type checking, and transparency across different type systems. We briefly presented the LINQ implementations available in .NET 3.5—LINQ to Objects, LINQ to ADO.NET and LINQ to XML. Finally, we made some concrete application example of LINQ to daily teaching activities. The application is written for Faculty of Electrical Engineering in East Sarajevo. It is easly adaptable for university faculties and beyond.*

Uvod

Language Integrated Query (LINQ) dostupan je u nekoliko različitih koncepta. Unutrašnja proširljivost ga čini korisnim u mnogim situacijama kada je manipulacija sa podacima potrebna. Pomenuti koncepti su LINQ to Object LINQ to ADO.NET i LINQ to XML.

LiINQ to ADO.NET uključuje nekoliko LINQ implmentacija koje funkcionišu sa Microsoft ADO.NET – om:

- LINQ to SQL omogućava upite nad relacionom strukturom pretvaranjem LINQ upita u izvorni SQL upit
- LINQ to DataSet interaguje sa podacima koji se već nalaze u ADO.NET DataSet strukturama
- LINQ to Entities omogućava postavljanje upita nad logičkim modelima koji su na većem nivou

LINQ to SQL

LINQ to SQL je koncept za pristup pravim bazama podataka tj. onome za šta SQL jezik najviše i služi. U početku je bio koncept namijenjen samo za pristup bazi podataka SQL Server dok je u trenutku pisanja ovog rada dostupan i LINQ to Oracle provajder koji omogućcava postavljanje upita nad Oracle bazama podataka. Bilo koji vanjski podaci moraju biti opisani odgovarajućim meta podacima na osnovu kojih se definiše klasa. Svaka tablela mora imati odgovarajuću klasu. Klasa može potpuno ili djelimično da opisuje fizičku tabelu, pogled ili rezultat snimljene procedure (engl. stored procedures). U LINQ upitima možemo koristiti samo ona polja koja su opisana pomoću navedenih tabela.

Entitet tabele

Da bi koristili LINQ nad nekom bazom podataka prvo što treba uraditi je napraviti tzv. entitet tabele. Entitet tabele su neka vrsta preslikavanja tabela iz baze podataka u klase u memoriji. U nekim od narednih primjera koristićemo elemente aplikacije Dnevnik nastavne aktivnosti Elektrotehničkog fakulteta u I. Sarajevu koja je realizovana upotrebom ove tehnologije. Klasa tj. entitet tabela “ocjene”:

```
[Table(Name="dbo.ocjene")]
public partial class ocjene{

private string _id_stud;
private int _id_predmet;
private int _id_ispit;
private int _ocjena;
private int _id_prof;
private string _datum;
            ...
[Column(Storage="_ocjena",
DbType="Int NOT NULL")]
```

```
public int ocjena
{
get
{
      return this._ocjena;
}
set
{
if ((this._ocjena != value))
{
this.OnocjenaChanging(value);
this.SendPropertyChanging();
this._ocjena = value;
this.SendPropertyChanged("ocjena")
;
this.OnocjenaChanged();
}
}
}
            ...
}
```

nije prikazana u cjelosti zbog preglednosti rada. Navedena entitet klasa je obimna zbog toga što su joj varijable tipa *private* i svaka od njih ima svoje svojstvo pomoću koga joj se pristupa. Prikazani su samo njeni dijelovi koji su bitni za razumijevanje pojma entitet tabela tj. entitet klasa. U entitet klasi možemo primijetiti atribute ispred deklaracije klase i varijabli. Atribut je obavezno navesti a svojstva su proizvoljna. Tako npr. svojstvo Name ispred deklaracije klase ne mora se navesti ukoliko je naziv klase isti kao i naziv tabele u bazi podataka. Isto tako svojstvo Name nije potrebno navoditi za atribut Column ako se nazivi varijable članice ne razlikuje od naziva polja u tabeli. Ukoliko postoji potreba da ta dva naziva ne budu ista onda je potrebno navesti svojstvo Name. Ukoliko je neka varijabla primarni ključ potrebno je navesti svojstvo koje se zove IsPrimaryKey ispred deklaracije varijable, ali samo ako se vrše izmjene u tabeli inače ga nije obavezno navesti. Kao primjer prepravićemo prethodnu varijablu „ocjena”:

```
[Column(IsPrimaryKey = true)].
```

Kod kreiranja klase, koja će predstavljati preslikavanje tabele, treba obratiti pažnju i na tipove varijabli. Iako je riječ o skoro istim tipovima varijabli, oni imaju različite nazive u relacijskoj bazi podataka i .NET Frameworku. Navedene razlike date su u sledećoj tabeli:

RDB	.NET
tinyint, smallint, int, bigint	Byte, Int16, UInt16, Int32, UInt32, Int64, UInt64
bit	Boolean
decimal, numeric, smallmoney, money	Decimal
real, float	Single, Double
char, varchar, text, nchar, nvarchar, ntext	String
datetime, smalldatetime	DateTime
timestamp, binary, varbinary	Byte[], Binary

DataContext

Skupu entitetskih klasa koje LINQ to SQL zahtjeva potreban je sloj apstrakcije nad relacionim modelom. Svaka entitetska tabela definiše pristupne tabele podataka kojima se mogu modifikovati podaci i postavljati upiti. Entitetske instance promjene podataka mogu da primjene na podatke sadržane u relacionoj bazi podataka. DataContext klase sadrže komunikaciju između LINQ-a i eksternih relacionih izvora podataka. Svaka instanca je pojedinačna konekcija na bazu podataka i njen tip je IDbConnection. DataContext koristi meta podatke da bi preslikala fizičku strukturu relacionih podataka, na kojima se temelje generacije SQL koda. DataContext klasa takođe može biti korištena za poziv stornih procedura kao i podataka iz relacione baze koji su promijenjeni od strane instance entitetske klase. Umjesto klase DataContext možemo, a i preporučljivo je koristiti strogo tipski određenu klasu koja će naslijediti klasu DataContext i čije će članice biti sve tabele baze podataka.

LINQ to SQL stvara SQL naredbe na osnovu LINQ upita onako kako on misli da je najbolje. Ukoliko niste zadovoljni sa ovim generisanjem, čija složenost zavisi od složnosti LINQ izraza, možete sami direktno postaviti SQL upit. Naravno to mozete raditi ne samo ako mislite da možete napraviti SQL naredbu bolje od LINQ to SQL nego i u slučaju da on uopšte ne može generisati stablo izraza na osnovu koga bi se stvorio SQL upit. Da bi realizovali pomenuti upit imamo na raspolaganju metodu ExecuteQuery klase DataContext ili klase naslijeđene od nje. Ograničenje ovog pristupa je u samim razlozima uvođenja LINQ koncepta jer ako koristimo metodu ExecuteQuery tada nema više tipske određenosti, a nemamo ni pomoć prevodioca koji bi u vremenu prevođenja otkrio ne pravilnosti u upitu, pa bismo kao rezultat ne ispravno napisane SQL naredbe dobili grešku generisanu u vremenu izvođenja.

Storne Procedure

Objekti klase DataContext i klasa naslijeđenih od nje mogu pozivati i sve storne procedure pojedine baze podataka. Storne procedure su u klasi metode koje ne moraju imati isto ime kao storne procedure što se označava atributom Function i svojstvom Name koje će sadržati naziv procedure zapisan u bazi podataka. Storne procedure se pozivaju na isti način kao i druge . LINQ to SQL je povezao ne objektni sistem relacionih baza podataka sa objektno orijentisanim programiranjem. Dobila se stroga tipska određenost pri radu sa bazama podataka i bolja ispravnost koda u vremenu prevođenja (engl. Compile time checking). U nastavku ćemo vidjeti da se isto postiglo i sa XML dokumentima koji takođe nisu objektno orijentisani.

Klasa LinqDataSource

Serverski kontrolni objekat LinqDataSource je poseban po tome što sam sebi nije svrha, odnosno uvijek dolazi u kombinaciji s nekim drugim objektom koji će na kraju da prikaže podatke. Takav je npr. objekat klase GridView a objekat LinqDataSource je tu samo da ostvari vezu između tabele relacione baze podataka i objekta za prikaz podataka. LinqDataSource mnogo olakšava komunikaciju između web stranice i relacijske baze podataka. Veliki dio posla svodi se na par klikova miša. U pozadini se dešava sve ono što je vezano za izvođenje LINQ upita. Poznavanje procesa koji se dešava u pozadini može samo pomoći u razumijevanju ovih gotovih objekata. Nakon što kreiramo entitet klasu za bazu iz našeg primjera i postavimo objekte LinqdataSource i GridView na formu objekti će biti instancirani i u aspx fajlu će biti kreiran sledeći kod:

```
<asp: LinqDataSource
ID = “LinqdataSource1” runat =
“server”
ContextTypeName =
“EvidencijaStudenata”
          TableName =
”NastavaPrisustvo”>
</asp: GridView>

<asp: GridView
      ID = “GridView1” runat =
“server”
      DataSourceID =
“LinqdataSource1”>
<asp: GridView>
```

Svojstvo ContextTypeName sadrži naziv klase koja nasleđuje klasu DataContext, dok je u svojstvu TableName naziv tabele čiji će se podaci prikazati u GridView objektu. Isti taj GridView objekat preko svojsta DataSourceID uspostavlja vezu sa LinqDataSource objektom i to je kraj priče ukoliko želimo samo prikazati podatke iz baze podataka u GridView objektu. Objekat LinqDataSource ima svojstva Select, Where i OrderBy koja predstavljaju Linq operatore select, where i orderby s kojima se može izvesti željeni upit nad tabelom iz baze podataka. GridView objekat može se postaviti u stanje u kojem je moguće uređivati sadržaj, a preko objekta LinqdataSource takve je promjene moguće zapisivatinazad u bazu podataka. Potrebno je samo postaviti svojstva koja to određuju na vrijednost true:

```
LinqdataSource1.EnableInsert =
true;
LinqdataSource1.EnableUpdate =
true;
LinqdataSource1.EnableDelete =
true;
```

LINQ to DataSet

ADO.NET tehnologija koja se pojavila kao dio cjelokupne .NET platforme donijela je jedan nov, različit način rada sa bazama podataka. Ovu tehnologiju najviše je isticala mogućnost da se podaci iz relacijske baze podataka predstave kao objekti u memoriji bez obzira na različitost relacijske baze podataka i objekata u memoriji po pitanju objekte orijentisanosti.

Ključnu ulogu u svemu ovome ima klasa DataSet čiji objekti predstavljaju tabele iz baze podataka. Naročito je značajno što ova klasa u kombinaciji sa klasama Sql – DataAdapter i OleDbAdapter (zavisno da li je riječ o Sql bazi podataka ili nekoj drugoj bazi podataka) omogućuje da se sa preslikanim podacima radi kao i sa svim drugim promjenljivim tj. varijablama u memoriji bez obzira da li su to naredbe C# - a ili nekod drugog .NET podržanog programskog jezika. Takođe, postoji mogućnost da se podaci koji su eventualno promijenjeni zapišu nazad u bazu podataka pozivom jedne ili dvije metode klase DataSet.

Klasa DataSet ima veliku ulogu u ADO.NET koji opet ima značajnu ulogu u cjelokupnom razvoju softvera (u Windows svijetu) . Tako nam postaje očigledno da klasa DataSet ima globalno veliku ulogu u izradi programskih rješenja.

DataSet pristupa bazi podataka preko objekta jedne od dvije spomenute klase Adaptera tako da objekat klase DataSet nema ništasa bazom podataka i zato i ne postoje dvije verzije klase DataSeet (za SQL

Server i za ostale). Cijeli posao u komunikaciji sa bazom podataka obavlja Adapter koji:

- otvara bazu podataka pomoću proslijeđenog znakovnog niza za spajanje
- izdvaja podatke na osnovu proslijeđene SQL naredbe
- zatvara bazu podataka

Sledeći primjer pokazuje nam kako rade klase DataSet i SQLDataAdapter. Zbog jednostavnosti i uštede prostora izuzetke nećemo obuhvatiti iako su oni sastavni dio rada sa bazama podataka.

```
using System;
using System.Data;
using System.Data.Sql;
using System.Data.SqlClient;

public class Test
{
public static void Main()
{
SqlDataAdapter adapter = new
SqlDataAdapter ("SELECT * FROM
Customers",
"Server=.\\SQLExpress;AttachDbFile
name=c:\\DataBases\\ Northwnd.mdf;
Database=Northwind;Trusted_Connect
ion=Yes;");
SqlCommandBuilder builder = new
SqlCommandBuilder(adapter);
DataSet ds = new DataSet();
adapter.Fill(ds, "Customers");
DataTable table =
ds.Tables["Customers"];
foreach(DataRow row in table.Rows)
if(((string)row["CustomerID"])[0]
== 'S')
row["Country"] = "Srpska";
adapter.Update(ds, "Customers");
ds.AcceptChanges();
}
}
```

Objekat adapter je taj koji obavlja najveći posao i nalazi se između objekta ds klase DataSet i relacione baze pdoataka. Njegova metoda Fill preslikava tabelu Customers iz baze podataka u memoriju zajedno sa svim podacima tako da je ostvarena sledeća analogija:

- objekat klase DataTable predstavlja tabelu u memoriji
- objekat klase DataRow predstavlja slog (engl. Record) u memoriji
- objekat klase DataColumn predstavlja polje (engl. field) u slogu

Iako smo ostali vezani za SQL sintaksu u konstruktoru klase SqlDataAdapter, s podacima radimo isključivo i jedino naredbama C# programskog jezika. To ima velikih prednosti kao što je npr. provjera sintakse u vremenu prevođenja, umjesto u vremenu izvođenja što je za sve slučajeve a naročito za rad sa bazama podataka ne uporediva razlika. Kad je rad sa izmjenama podataka završen sve što treba uraditi je pozvati metode Update i AcceptChanges koje će promjene nad objektom ds zapisati nazad u bazu podataka.

LINQ upiti nad DataSet

Mnogo je programskog koda već napisano korištenjem klase DataSet i to je bio razlog zašto LINQ nije mogao ostati postrani kad je rijč o pretraživanju podataka koji se nalaze u objektima takve jedne klase. Tako se dobila dvostruka korist, programski kod ostaje, a naprednije tehnologije se mogu primijeniti. LINQ je za razliku od klase DataSet potpuno tipski određen zahvaljujući generičkim tipovima koji su uvedeni u C# 2.0. Klasa DataSet ne nasleđuje interfejs Ienumerable<T> pa je zato njen objekat potrebno pretvoriti u takav tip.
Nakon toga je sve spremno za ivođenje LINQ upita.

LINQ to DataSet operatori

LINQ to DataSet ima i neke specifične operatore koji se mogu koristiti isključivo za DataSet objekte. To su operatori CopyToDataTable, AsDataView i DataRowComparer

- Operator CopyToDataTable stvara kopiju tipa DataTable zajedno sa svim podacima.
- Operator AsDataView definisan je nad klasom EnumerableRowCollection i kao povratnu vrijednost vraća tip DataView, dobro poznatu klasu na čiji objekat možemo povezati neki kontrolni objekat za prikaz podataka kao npr. objekat klase GridView.
- DataRowComparer je operator koji se može koristiti jedino u LINQ to DataSet. Operator upoređujedva sloga iz dva različita objekta tipa DataTable i vraća true ako su sve vrijednosti u slogu jednake ili false ako bar jedna od vrijednosti nije jednaka.

LINQ to XML

LINQ to SQL je najkompleksniji od svih dosad nabrojanih provajdera zato što uključuje mnogo

funkcionalnosti. LINQ to XML pruža cijeli niz mogućnosti za obradu XML-a ali ga je bolje gledati kao specijalni način za interakciju sa XML dokumentima nego kao objektni model. XML dokument može se pojaviti u nekoliko oblika i LINQ to XML mora ih sve obraditi. Programeri koji ga koriste imaju potrebu da vrše unos. brisanje i ažuriranje XML elemenata i atributa. Takodje mogu da vrše transformaciju XML dokumenata u neke druge forme kao i obrnuto tj. transformisanje nekih drugih formi dokumenata u XML dokumente. Sve ovo čini XML nekom vrstom središnjeg tla za transformaciju podataka. Obrada XML dokumenata unosi složenost u LINQ to XML provajder. Možete vršiti parsiranje cijelog dokumenta, dijela dokumenta ili pojedine nizove u dokumentu. To je prostor gdje uvijek dolazi do grešaka pa iz tih razloga LINQ to XML provajder mora da obezbijedi funkcionalno rukovanje tim greškama.
Sledeća lista opisuje kako možete koristiti LINQ to XML u interakciji sa XML-om:

- učitavanje XML dokumenta iz datoteke u memoriju
- definisanje vlastitih XML dokumenata korištenjem XML upita
- kreiranje XML upita koji izlgedaju kao I svaki drugi LINQ upiti
- kreiranje složenih XML upita koji koriste XPath za lokaciju određenih dijelova podataka
- izmjena XML stable predstavljenog u stablu metodama kao što su: Add, ReplaceWith, Remove i SetValue
- transformacije podataka iz jednog formata u drugi pomoću XML-a kao polazne tačke, kao krajnje tačke ili kao posrednika između dva različita formata

Pod konceptom LINQ to XML možemo podrazumijevati dvije tehnologije koje su dosta povezane ali ipak i različite:

- skup klasa za rad sa XML dokumentima
- izradu LINQ upita nad sekvencom koja će predstavljati XML document

Osnovna klasa za rad sa XML dokumentima je XElement i nalazi se u System.Xml.Linq. Neki podaci pokazuju da se trećina programskog koda utroši na rad sa podacima kojima pristupamo a koji mogu biti objekti u memoriji, podaci u relacijskoj bazi podataka ili XML dokumenti. Kako LINQ omogoućava da sa svima njima radimo na isti način to znači da bi ušteda u vremenu razvijanja bila 22%. Time ova tehnologija potpunjo opravdava svoje postojanje.

LINQ to Object

Objekti (iz LINQ perspektive) mogu da upućuju na bilo koji element u vašoj aplikaciji koji implementira IEnumerable interfejs. Npr. Objekat može da sadrži polja, DataSet, baze pdoataka ili čak kontrole. Možete kreirati klase koje podržavaju IEnumerable interfejs I komunicirati sa njima kao objektima u LINQ. LINQ to Object je generički provajder koga ćete koristiti za obavljenje bilo kojeg zadatka sa LINQ. Ostali provajderi samo dodaju funkcionalnosti u Linq to Object provajder.

LITERATURA

[1] Paolo Pialorsi, Marco Russo, *INTRODUCING MICROSOFT LINQ*