UDK: 004.6 Strucni rad

INTEGRISANI UPITNI JEZIK – LINQ U OBJEKTNO ORIJENTISANOM OKRUŽENJU - C#

LANGUAGE INTEGRATED QUERY IN OBJECT ORIENTED ENVIRONMENT C#

Goran Miodragović[1], Selver Pepić[2], Slobodan Ivanović[3], Slobodan Aleksandrov[4], Snežana Gavrilović[5]
[1,2,3,4,5]*Visoka tehnička mašinska škola strukovnih studija Trstenik*
Radoja Krstića 19, 37240 Trstenik, Srbija
[1]goran.miodragovic@vtmsts.edu.rs, [2]selver.pepic@vtmsts.edu.rs,
[3]slobodan.ivanovic@vtmsts.edu.rs,[4]slobodan.aleksandrov@vtmsts.edu.rs,
[5]gavrilovicsnezana@yahoo.com

Abstrakt: *U radu je prikazana nova paradigma korišćenja ugrađenog SQL jezika. Iako je SQL jezik, od kraja 70tih godina pa do danas, podrazumevani jezik za rad sa serverima baza podataka, postavlja se pitanje da li je moguća optimizacija sintakse za formiranje upita nad bazom podataka. Ovo je naročito značajno kod složenijih upita. Kao odgovor na ovaj zahtev u okviru C#, odnosno kompletnog MS Visual Studio-a, implementiran je LINQ u cilju pojednostavljenja i unificiranja pristupu bilo kojoj vrsti podataka.*
Ključne reči: *LINQ, SQL, C#, baze podataka.*

Abstract:
This paper presents a new paradigm using the embedded SQL language. While the SQL language, since the end of the 70s until today, the default language for working with database servers, the question is whether it is possible to optimize the formation of the syntax for queries of the the database. This is especially significant for more complex queries. In response to this request in the context of C #, or the entire MS Visual Studio, LINQ is implemented with the aim of unifying and simplified access to any type of data.
Key words: *LINQ, SQL, C#, databases.*

1. UVOD

U savremenom programiranju, podaci kojima se programski upravlja, kroz generisanje, transformisanje i manipulisanje, pripadaju različitim domenima: niz, objekat, graf, XML dokument, baza podataka, tekstualna datoteka, registar ključeva, e-mail poruka, SOAP (Simple Object Access Protocol) sadržaj poruke, XLS datoteka (MS EXCEL) i slično. Svaki domen podataka ima svoj određeni protokol pristupa. Kada su u pitanju baze podataka, podrazumeva se da se koristi SQL metod, dok kod XML dokumenata, koristi se Xquery ili DOM (Document Object Model). Za čitanje grafova, koriste se protokoli za

formiranje algoritama kojim se podaci sa grafova smeštaju u nizove. Dalje, da bi se pristupilo različitim podacima koji pripadaju MS OFFICE-u, kao što je MS EXCEL, tabele ACCESS-ove baze podataka, e-mail porukama, koriste se aplikacije formirane u API-ju (Application Programming Interfaces). Jednom rečju, moraju se pisati različiti programski modeli za pristupanje i korišćenje različitih izvora podataka.

Postoje različiti pokušaji da se unificira pristup podacima u jedan sveobuhvatni model. Jedan od pokušaja je i ODBC (Open Database Connectivity) metod, koji omogućava transformaciju i protok podataka iz različitih oblika baza podataka, korišćenjem sintakse programskog jezika koji se koristi za transformaciju. Na ovaj način moguće je izvršiti transfer podataka iz MS ACCESS baze podataka u tabele Visual FoxPro-a, Paradox tabele, ili pak u Excel i obrnuto. Mana ovakvog pristupa je to što korisnik mora da ima instalirane drajvere da bi povezao pomenute različite domene podataka, kao i to da se koriste upiti slični SQL – u za pristup podacima predstavljenim preko relacionog modela. Međutim, ponekad se podaci znatno efikasnije predstavljaju pomoću hijerarhijskog ili grafičkog modela, pa bi pomenuti koncept, zasnovan na relacionom modelu, mogao da dovede do grešaka, ili čak nemogućnosti pristupa podacima iz različitih sistema. Osim toga, ako model podataka nije direktno podržan softverom u kome se program piše, najverovatnije se mora predvideti upravljanje sa različitim tipovima strukture podataka.

Svi ovi problemi prouzrukuju neusklađenost između sistema podataka i programskog koda. Sistem koji pokušava da reši ove probleme je integrisani jezik upita (engl. Language INtegrateg Query) – LINQ. LINQ nudi jedinstveni način za pristup i upravljanje podacima bez forsiranja modela usvajanja "jedna veličina odgovara svima" [1]. LINQ usklađuje sličnosti između operacija u ovim modelima podataka umesto da izjednačuje njihove strukture.

2. LINQ NOVI PRISTUP PISANJA UPITA ZA BAZE PODATAKA

LINQ to SQL, je deo .NET-a 3.5, napravljen u svrhu upravljanja podacima, iz relacionih baza podataka, korišćenjem objekata, uz mogućnost postavljanja upita na drugačijoj osnovi. Naime, LINQ je zasnovan na tome da se upiti pisani u LINQ-u prevode u SQL za izvršenje nad bazom podataka, a rezultati upita se prevode u objekte [2].

LINQ predstavlja model programiranja koji uvodi upite kao jedinstveni koncept u bilo koji Microsoft – ov .NET programski jezik. Da bi koncept LINQ bio efikasan u svakom programskom jeziku izvršena su određena proširenja i prilagođavanja sintakse. Ova proširenja treba da obezbede veću produktivnost koja se ogleda u jednostavnijoj, moćnijoj i izražajnijoj sintaksi za manipulisanje podacima različitih domena [3].

Pristup podacima u LINQ to SQL-u se obavlja na objektno-orijentisani način korišćenjem objekata mapiranih na relacionu bazu podataka. Umesto, da se preko stranih ključeva objekti povezuju, oni su povezani referencama, tako da neki od objekata mogu sadržati kolekcije drugih objekata, čineći takozvani "graf objekata". Pošto graf objekata može biti veliki, LINQ to SQL odlaže učitavanje (deferred loading) – učitava podatke (obavlja dodatne upite) samo onda kada su zatraženi. Drugačije rečeno, kako se LINQ izrazi prevode u SQL, moguće ga je koristiti s različitim SUBP-ima bez potrebe za

prilagođavanjem koda upita [4].

U upitu Q.1., je dat primer jednostavnog LINQ upita, kao tipičnog softverskog rešenja, koji vraća ime, prezime i broj indeksa studenta koji su na smeru Informacione tehnologije.

```
var query =
        from s in Studenti
        where s.Smer == "Infromacione tehnologije"
        select s.Prezime+s.Ime+s.BrojIndeksa As Rezultat;                    [Q.1.]
```

Rezultat upita je spisak (lista) studenata i svaki od redova u rezultatu predstavlja tekstualni tip podatka – string. U programskom jeziku C#, rezultat upita Q.1, može se predstaviti sintaksom kao što je prikazano u izrazu I.1.

```
foreach (Rezultat)
{
        Console.WriteLine( red );
}                                                                            [I.1.]
```

I upit Q.1., i izraz I.1., predstavljaju regularnu sintaksu programskog jezika C#, tako da je opravdano pitanje da li je upit Q.1., novi oblik ugrađenog SQL –a. Naravno, to nije slučaj, jer dati upit (sa kodom datim u izrazu I.1.) se može primeniti i na SQL bazu podataka, na DataSet, na nizove objekata, odnosno na sve ostale tipove podataka. To znači da se Studenti mogu posmatrati kao kolekcija objekata, I.2., odnosno konceptualni model podataka preslikan u relacionu bazu podataka, I.3..

```
EvidencijaStudenata[] Studenti;
        Studenti mogu biti DataTable u DataSet:
        DataSet ds = GetDataSet();
        DataTable Studenti = ds.Tables["Studenti"];
DataContext db = new DataContext( ConnectionString );
Table<Studenti> EvidencijaStudenata = db.GetTable<Studenti>();               [I.2.]
```

```
EvidencijaStudenata dataModel = new EvidencijaStudenata();
ObjectQuery<Student> Studenti = dataModel.Studenti;                          [I.3.]
```

Sintaksa, slična SQL – u, koja se koristi u LINQ se naziva upitni izraz, [1]. Softveri koji primenjuju ugrađeni SQL, definišu sintaksu za postavljanje SQL iskaza u tom softveru. Međutim, ti iskazi nisu integrisani u prirodnu sintaksu i tip sistema datog softvera. Na primer, kada se koriste ugrađeni SQL, ne može se pozvati, izvorna, funkcija za ispisivanje rezultata u sredini SQL izraza, dok je primenom LINQ – a to moguće. Za razliku od SQL – a, LINQ nije ograničen samo na upite nad bazama podataka.

3. NAČIN RADA LINQ

Da bi se shvatio način izvršenja koncepta integracije upita u softver, posmatraće se primer dat u kodu Q.2. Kada se napiše kod, Q.2., kompajler generiše kod kao što je prikazano u Q.3.

```
Student[] Studenti = GetStudenti();
var query =
from s in Studenti
where s.Smer== " Infromacione tehnologije "
select s;                                                    [Q.2.]
```

```
Student[] Studenti = GetStudenti();
IEnumerable<Student> query =
        Studenti
        .Where( s => s.Smer == " Infromacione tehnologije " );   [Q.3.]
```

U slučaju da je upit proširen, Q.4, kompajler generiše kod Q.5.

```
var query =
        from s in Studenti
        where s. Smer== " Infromacione tehnologije "
        orderby s.BrojIndeksa
        select new { s. Prezime, s.Ime, s.Mesto };           [Q.4.]
```

```
var query =
        Studenti
        .Where( s => s. Smer== " Infromacione tehnologije " );
        .OrderBy( s => s.BrojIndeksa )
        .Select( s=> new { c.Prezime, c.Ime, c.Mesto } );    [Q.5.]
```

Kao što može da se primeti iz izraza Q.2. – Q.5., kod očigledno poziva instance članova objekta vraćenog iz prethodnog poziva. Može se videti da je rezultat koda regulisan dodatom sintaksom softvera u kome se piše program, u ovom slučaju sintaksom C#. Implementacija sintakse *Where*, *OrderBy* i *Select*, zavisi od toga kog je tipa objekat Studenti, i od prostora podataka koji je prethodno određen. Ove dodate metode su osnovna sintaksa koja se koristi u LINQ, što omogućava rad sa istom sintaksom, a za različite domene podataka.

Drugi važan koncept je vreme izvršenja operacije nad podacima. Generalno posmatrano, LINQ upit se ne izvršava sve dok se pristupa rezultatima upita, zato što opisuje skup operacija koje će se izvesti kada bude potrebno.

Kada se LINQ upit, izvodi nad relacionom bazom podataka, generiše stanje ekvivalentno SQL – u, umesto da kopije podataka, iz tabela baze podataka, smešta u memoriju.

4. LINQ i relacioni model podataka

Na prvi pogled, LINQ se može učiniti kao jedna od modifikacija SQL upitnog jezika. Sličnost proističe iz sintakse LINQ – a, kojima se opisuje odnos između entiteta, a koji podseća na SQL komandu *JOIN*, Q.6.

```
var query =
        from s in Student
        join o in Ocene
         on s. BrojIndeksa equals o. BrojIndeksa
        select  {s. BrojIndeksa, s.Prezime, s.Ime, o.PredmetID, o.Ocena };        [Q. 6]
```

Upit predstavljen Q.6., je sličan postavljanju upita u relacionom modelu. Međutim, LINQ nije limitiran na pojedinačne domene podataka kao što je to slučaj kod relacionog modela. U relacionom modelu svaki student može, za svaki ispitni rok, da prijavi više ispita, i da u svakom prijavi više predmeta koje polaže. U LINQ – u, lista predmeta koji se polažu u jednom ispitnom roku može se generisati kao što je prikazano u upitu Q.7.

```
var query =
        from s in Student
        from i_r in s.Ispitni_rok
        select new { s. BrojIndeksa, s.Ime, s.Prezime, i_r.IspitniRok,
                     i_r.Predmet.Naziv };                                        [Q.7]
```

Upit, Q.7., ne sadrži klauzulu *JOIN*. Relacije između entiteta Student i Ispitni_rok se izražava u drugoj *from* klauzuli, gde sintaksa *s.Ispitni_rok* znači « preuzeti sve predmete iz ispitnog roka za *s Student*». Relacija između entiteta Ispitni_rok i Predmet se izražava preko stranog ključa Predmet u entitetu Ispitni_rok. Rezultat je naziv predmeta za svaki red iz tabele Ispitni_rok korišćenjem *i_r.Predmet.Naziv.*

Ukoliko postoje uspostavljene relacije između entiteta, LINQ – u, takođe, dopušta korišćenje eksplicitnih (unapred definisanih) relacija iz modela podataka. Na primer, ako želimo da pronadjemo studente i profesore koji su iz istog grada, a relacioni model podataka nije predvideo eksplicitnu vezu između tih atributa, LINQ – u dopušta uspostavljanje veza (bez posledica po relacioni model podataka). Naravno, postoji mogućnost pisanja upita, koji će za posledicu imati grupisanje podataka po izabranom kriterijumu, Q.8.

```
var query =
        from s in Studenti
        join p in Profesori
         on s.Mesto equals p.Mesto
        into StudentProfesor
        select new { s.Mesto, s.ImePrezime, StudentProfesor };                   [Q.8]
```

Poslednji upit za rezultat ima redove koji prikazuju listu profesora za svakog studenta,

koji su iz istog mesta, T1.

T.2. Rezultat izvršenja upita Q.8.

Mesto: Trstenik	*ImePrezime*: S_im_pr_1	
	Profesor: P_im_pr_1	*Mesto*: Trstenik
	Profesor: P_im_pr_2	*Mesto*: Trstenik
Mesto: Čačak	*ImePrezime*: S_im_pr_2	
	Profesor: P_im_pr_3	*Mesto*: Čačak
	Profesor: P_im_pr_4	*Mesto*: Čačak

Pri formiranju LINQ upita, podaci koji se izdvajaju iz baze podataka, se opisuju na potpuno isti način koji zahteva sintaksa jezika u koji je ugrađen LINQ (u ovom slučaju C#).

5. ZAKLJUČAK

LINQ predstavlja tehnologiju upita ugrađenih u određeni softver, transformišući sintaksu tog softvera u SQL. LINQ manipuliše relacionim modelima podataka bez gubitka mogućnosti postavljanja upita. Zasnovan je na tome da se upiti pisani LINQ–om prevode u SQL, a rezultati upita se prevode u objekte. Infrastruktura LINQ–a omogućuje dinamičku izgradnju složenih upita, koristeći sve pogodnosti Visual Studio razvojnog okruženja.

LITERATURA

[1] Pialorsi, P., Russo, M., Introducing Microsoft LINQ, Microsoft Corporation, 2007.

[2] MSDN, .NET Language-Integrated Query for Relational Data, URL: https://msdn.microsoft.com/en-us/library/bb425822.aspx (13.02.2017)

[3] MSDN, Language-Integrated Query, URL: https://msdn.microsoft.com/en-us/library/bb397926(v=vs.110).aspx (01.02.2017)

[4] CQL++ Database Management System, URL: http://www.cql.com/literat.html (01.03.2017)

[5] Stroustrup B. (2013). *The C++ Programming Language (4th Edition)*, Addison-Wesley Professional.

Tutorials Point, LINQ stands for Language Integrated Query, Copyright 2015 by Tutorials Point (I) Pvt. Ltd.