1. ADO.NET

Ciljevi lekcije

- Upoznati se sa osnovama ADO.NET-a
- Upoznati se sa ADO.NET Data Provider-ima
- Upoznati se sa direktnim pristupom podacima korišćenjem ADO.NET-a (kreiranje konekcije ka bazi, kreiranje SQL komande, korišćenje DataReader objekta, parametrizacija upita)
- Upoznati se sa korišćenjem beskonekcionog sloja ADO.NET-a (korišćenje DataAdapter, DataSet, DataTable, DataColumn i DataRow klasa)

Neophodno predznanje

- Rad u Windows okruženju
- Relacione baze podataka
- Osnove programskog jezika C#

1.1. Uvod

Većina trenutno prisutnih aplikacija vrši obradu podataka. Najčešći način skladištenja i korišćenja uskladištenih podataka je upotreba baza podataka. U najjednostavnijim slučajevima, aplikacije koje pristupaju bazi podataka omogućavaju svojim korisnicima da pretražuju podatke i prikazuju ih u tabelarnom obliku. Za pristup podacima projektanti aplikacija mogu koristiti različite tehnologije.

U prošlosti je razvijeno više standardnih interfejsa za pristup bazama podataka. Svaki sistem za upravljanje bazama podataka (*eng. Database Management System – DBMS*) poseduje sopstveni programski interfejs (*eng. application programming interface-API*) čijim korišćenjem je moguće iz programskog koda odnosno iz aplikacije, vršiti manipulaciju podacima u bazi podataka. Programski interfejs (API) predstavlja kolekciju objekata i metoda koji omogućavaju pozivanje funkcija sistema za upravljanje bazama podataka (DBMS-a) iz programskog koda. Svaki DBMS poseduje svoj programski interfejs pa je bilo neophodno razviti standarde za pristup bazama podataka kako projektanti aplikacija ne bi morali da koriste različite interfejse u zavisnosti od konkretnog DBMS-a koji koriste.

Open Database Connectivity (ODBC) standard je razvijen sa ciljem da obezbedi načine za manipulaciju podacima u relacionim bazama podataka koji bi bili nezavisni od konkretnog DBMS-a. Microsoft je razvio OLE DB, objektno-orijentisani interfejs koji enkapsulira funkcionalnosti servera baza podataka. OLE DB je razvijen ne samo za relacione baze podataka već ima i mogućnost korišćenje drugih tipova podataka. Ovaj interfejs nisu mogli koristiti projektanti koji su svoje aplikacije razvijali korišćenjem Visual Basic-a i script jezika pa je Microsoft razvio Active Data Object (ADO) interfejs. ADO koristi funkcionalnosti OLE DB interfejsa i može biti korišćen iz bilo kog programskog jezika.

ADO.NET je naslednik ADO-a i deo je Microsoft-ove .NET platforme.

Funkcionalnosti ADO.NET-a baziraju se na korišćenju novog objekta pod nazivom DataSet. **DataSet predstavlja lokalnu kopiju podataka pribavljenih iz baze podataka i može sadržati više od jedne tabele**. Verovatno najbitnija karakteristika ovog objekta je činjenica da pruža mogućnost manipulacije nad podacima bez potrebe da veza sa bazom podataka bude u svakom trenutku otvorena. Prethodne tehnologije

za pristup podacima su pretpostavljale da je veza sa bazom podataka aktivna za vreme izvršenja koda koji vrši obradu podataka. Stalno aktivna konekcija dozvoljavala je trenutne izmene podataka i nadgledanje promena za vreme izvršenja koda. Problem je predstavljao ograničeni broj konekcija koje je server baze podataka mogao da pruži korisnicima pa su nakon zauzeća svih dostupnih konekcija ostali korisnici morali da čekaju da se neka od konekcija oslobodi.

ADO.NET ima u potpunosti drugačiji pristup u odnosu na prethodnike. Konekcija sa bazom podataka se i dalje kreira ali je moguće mnogo ranije osloboditi konekciju i učiniti je dostupnom ostalim korisnicima. Razlog je mogućnost pribavljanja kopije podataka iz baze i skladištenja ovih podataka u DataSet objektu. Nakon pribavljanja podataka, moguće je zatvoriti konekciju pre početka obrade podataka. Naravno, nakon završetka obrade podataka izmenjena je jedino lokalna kopija podataka pa je neophodno ponovo otvoriti konekciju ka bazi podataka kako bi bilo moguće snimiti izmene.

1.2. ADO.NET data provider-i

ADO.NET ne poseduje jedinstveni skup tipova objekata koji komuniciraju sa različitim sistemima za upravljanje bazama podataka (DBMS). **ADO.NET poseduje više skupova tipova objekata koji komuniciraju sa DBMS-om**. Ove skupove tipova objekata nazivamo data provider-ima. **Svaki od data provider-a optimizovan je za interakciju sa konkretnim DBMS-om**. Prednost ovakvog pristupa je mogućnost pojedinačnih data provider-a da imaju mogućnost manipulacije objektima koji su specifični za posmatrani DBMS. Još jedna od prednosti je način komunikacije između data provider-a i DBMS-a. Naime, s obzirom da je svaki data provider optimizovan za rad sa konkretnim DBMS-om, on komunicira direktno sa DBMS-om tj ne postoji međusloj koji bi prilagodio zahteve korisnika konkretnom DBMS-u.

Data provider je najlakše posmatrati kao skup tipova objekata definisanih u određenom prostoru imena (eng. namespace) koji imaju mogućnost direktne komunikacije sa konkretnim DBMS-om. Svaki od data provider-a poseduje skup klasa koje omogućavaju izvršenje osnovnih funkcionalnosti. Sve klase konkretnih data provider-a imaju zajedničke roditeljske klase tj izvedene su iz istog skupa klasa i interfejsa, koje se nalaze u System.Data.Common prostoru imena odnosno u System.Data prostoru imena. Osnovni objekti ADO.NET Data Provider tipa objekta prikazani su u tabeli 1.

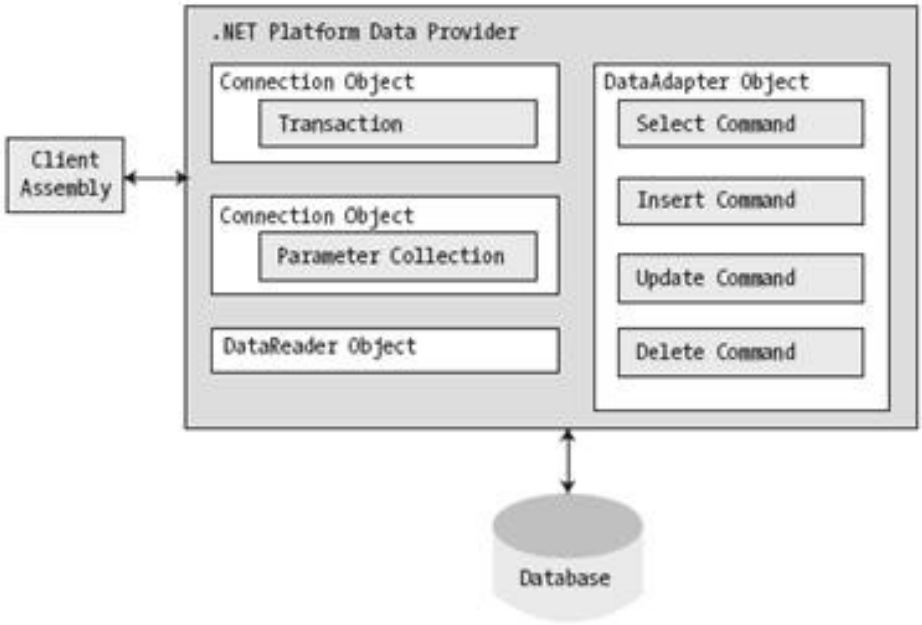
Objekat	Roditeljska klasa	Implementira interfejs	Značenje
Connection	DbConnection	IDbConnection	Omogućava otvaranje i zatvaranje konekcije ka bazi podataka
Command	DbCommand	IDbCommand	Predstavlja SQL upit ili uskladištenu proceduru. Omogućava pristup DataReader objektu konkretnog data provider-a
DataReader	DbDataReader	IDataReader, IDataRecord	Omogućava čitanje podataka korišćenjem kursora na serverskoj strani

Objekat	Roditeljska klasa	Implementira interfejs	Značenje
DataAdapter	DbDataAdapter	IDataAdapter, IDbDataAdapter	Prenosi DataSet objekte između klijenta i izvora podataka. Poseduje konekciju i skup od četiri osnovne operacije za selektovanje, dodavanje, izmenu i brisanje podataka u izvoru podataka
Parameter	DbParameter	IDataParameter,	Predstavlja imenovani parametar u

Objekat	Roditeljska klasa	Implementira interfejs	Značenje
		IDbDataParameter	parametrizovanom upitu
Transaction	DbTransaction	IDbTransaction	Enkapsulira transakciju baze podataka

Tabela 1: Osnovni objekti ADO.NET Data Provider-a

Iako će se imena konkretnih objekata svakog od konkretnih Data Provider objekata razlikovati (npr. SqlConnection, OracleConnection, OdbcConnection ili MySqlConnection), svi su izvedeni iz iste klase i implementiraju iste interfejse pa je nakon savladavanja korišćenja jednog od data provider-a relativno jednostavno koristiti sve ostale. Struktura data provider-a .NET platforme prikazana je na slici.



Microsoft .NET platforma poseduje niz ugrađenih data provider-a za različite DBMS-ove. Spisak ugrađenih data provider-a prikazan je u tabeli 2.

Data Provider	Prostor imena (namespace)	Biblioteka
OLE DB	System.Data.MySql	System.Data.dll
Microsoft SQL Server	System.Data.SqlClient	System.Data.dll
Microsoft SQL Server Mobile	System.Data.SqlServerCe	System.Data.SqlServerCe.dll
ODBC	System.Data.Odbc	System.Data.dll
Oracle	System.Data.OracleClient	System.Data.OracleClient.dll

Tabela 2: Microsoft ADO.NET Data Provider-i

Pored data provider-a koji su ugrađeni u .NET platformu, moguće je koristiti data provider-e koje obezbeđuju pojedinačni proizvođači besplatnih ili komercijalnih DBMS-ova poput SQLite, DB2, MySQL, PostgreSQL ili Sybase.

Pored predstavljenih prostora imena, .NET platforma poseduje skup dodatnih prostora imena koji svojim funkcionalnostima pripadaju skupu ADO.NET prostora imena. Neki od ovih prostora imena prikazani su u tabeli 3.

Prostor imena (namespace)	Značenje
System.Data	Definiše osnovne ADO.NET tipove koje koriste svi data provider-i

System.Data.Common	Sadrži tipove koje dele svi ADO.NET data provider-i
System.Data.Sql	Sadrži tipove koji omogućavaju otkrivanje instanci MS SQL Server-a u lokalnoj mreži
System.Data.SqlTypes	Sadrži tipove podataka koje koristi Microsoft SQL Server

Tabela 3: Spisak nekih od ADO.NET prostora imena

Od svih ADO.NET prostora imena (namespace-a), System.Data je najopštiji. Svaka aplikacija koja želi da pristupa podacima korišćenjem ADO.NET-a mora koristiti klase definisane ovim prostorom imena. System.Data sadrži klase koje su zajedničke za sve data provider-e. U tabeli 4 prikazani su osnovni članovi System.Data prostora imena.

Klasa	**Značenje**
Constraint	Predstavlja ograničenje primenjeno na DataColumn objekat
DataColumn	Predstavlja jednu kolonu DataTable objekta
DataRelation	Predstavlja roditelj/dete odnos između dva DataTable objekta
DataRow	Predstavlja jedan red u DataTable objektu
DataSet	Predstavlja lokalnu kopiju podataka u memoriji klijentskog računara koji se sastoji od niza povezanih DataTable objekata
DataTable	Predstavlja lokalnu kopiju tabele baze podataka
DataTableReader	Omogućava čitanje podataka iz DataTable objekta red po red
DataView	Predstavlja pogled na tabelu baze podataka i koirsti se za sortiranje, filtriranje, pretraživanje i izmenu podataka

Tabela 4: Osnovni članovi System.Data prostora imena

1.3. Direktan pristup podacima korišćenjem ADO.NET-a

Najlakši način izvršenja svih operacija nad bazom podataka je direktno izvršenje svih operacija pri čemu se ne vodi računa o lokalnim kopijama podataka. Ovakav model je najbliži tradicionalnom ADO programiranju i otklanja probleme konkurentnog izvršenja operacija nad bazom podataka koji se dešavaju kada više korisnika istovremeno izvršava operacije nad istim podskupom podataka. Ovakav način izvršenja operacija nad bazom podataka je dobro rešenje **kada je potrebno pročitati podatke ili izmeniti podake u jednom redu neke od tabela relacione baze podataka**. Ovakav pristup nije efikasan ukoliko je potrebno modifikovati više različitih redova iz jedne ili više tabela.

Uobičajeni redosled operacija prilikom pribavljanja podataka korišćenjem ovakvog pristupa sastoji se iz sledećih koraka:

1. Kreirati **Connection**, **Command** i **DataReader** objekte
2. Otvoriti konekciju
3. Koristiti **DataReader** za čitanje podataka iz baze podataka
4. Zatvoriti konekciju

1.3.1. Kreiranje konekcije ka bazi podataka

Pre pribavljanja ili izmene podataka neophodno je kreirati konekciju ka izvoru podataka. Broj raspoloživih konekcija je ograničen pa je potrebno držati konekciju otvorenom što je kraće moguće. Konekcija ka izvoru podataka u ADO.NET-u enkapsulirana je klasom **Connection**. Prilikom kreiranja

instance klase Connection neophodno je definisati **ConnectionString** atribut ove klase. **ConnectionString** atribut predstavlja formatirani niz karaktera sastavljen od niza ime/vrednost parova odvojenih međusobno karakterom ';'. Ovaj atribut najčešće sadrži informacije o imenu mašine kojoj pristupamo, načinu autentifikacije korisnika, imenu baze kojoj pristupamo i sl. U nastavku je dat primer kreiranja konekcije korišćenjem Microsoft SQL Server data provider-a , Oracle data provider-a i MySQL data provider-a.

Microsoft SQL Server:

```
SqlConnection conn = new SqlConnection();
conn.ConnectionString = "Data Source=localhost;Initial Catalog=TestDB;Integrated Security=SSPI");
```

Oracle

```
OrecleConnection conn = new OracleConnection();
conn.ConnectionString = "Data Source=192.168.1.190/lab.skola.edu.rs;User Id=21101;Password=21101";
```

Data Source ukazuje na ime servera na kome se nalazi baza podataka kojoj pristupamo. **Initial Catalog** predstavlja ime baze podataka kojoj ćemo pristupati korišćenjem kreirane konekcije. **Integrated Security** ukazuje na način autentifikacije korisnika. U posmatranom slučaju pristupa se korišćenjem Windows korisničkog naloga. Alternativno, moguće je proslediti username i password naloga koji želimo da koristimo za pristup bazi podataka.

MySQL

Microsoft Visual Studio ne dolazi sa ugrađenom podrškom za MySQL !
Prvo treba biti svestan da za MySQL treba instalirati MySqlConnector – preuzima se sa sajta proizvođača – da bi Visual Studio mogao da radi sa MySQL.
Drugo, od verzije Visual Studio 2015 pa nadalje, postoji i dodatak „MySQL for Visual Studio" koji omogućava lakši rad (pristup vizuelnim alatima)
I jedno i drugo se mogu postaviti na računar ako se koristi web installer.

Tek kada na računaru postoji instaliran MySqlConnector, može da se kreira konekcija korišćenjem MySql Data provider-a:

1.način:

```
MySqlConnection konekcija = new MySqlConnection("server=localhost;user id=ucenik;password=test;database=proba");
```

2.način:

```
MySqlConnection konekcija = new MySqlConnection();
konekcija.ConnectionString = "server=localhost;user id=ucenik; password=test; database=proba";
```

3.način:

```
string konekcioniString =  "server=localhost;user id=ucenik;password=test;database=proba";
MySqlConnection konekcija = new MySqlConnection(konekcioniString);
```

Server ukazuje na ime servera na kome se nalazi baza podataka kojoj pristupamo, dok **database** predstavlja ime baze podataka kojoj ćemo pristupati korišćenjem kreirane konekcije. U posmatranom

slučaju pristupa se korišćenjem username i password naloga koji želimo da koristimo za pristup bazi podataka.

Pre korišćenja konekcije neophodno je eksplicitno otvoriti konekciju:

Otvaranje konekcije

```
conn.Open();
```

Nakon eksplicitnog poziva funkcije za otvaranje konekcije, otvorena je i aktivna konekcija ka bazi podataka. Nakon završetka obrade podataka neophodno je zatvoriti konekciju pozivom funkcije za zatvaranje konekcije:

Zatvaranje konekcije

```
conn.Close();
```

Objekti klase **Connection** poseduju skup atributa kojima je moguće upravljati tokom izvršenja transakcija nad bazom podataka i pribaviti informacije vezane za izvor podataka kome se pristupa.

1.3.2. Kreiranje SQL komande

Nakon kreiranja konekcija ka bazi podataka, neophodno je izvršiti SQL naredbe za manipulaciju nad podacima. Ukoliko koristimo Microsoft SQL Server data provider, SQL naredbe se izvršavaju korišćenjem instanci klase SqlCommand, tj korišćenjem SqlCommand objekta. Ukoliko koristimo Oracle data provider, SQL naredbe se izvršavaju korišćenjem instanci klase OracleCommand, tj korišćanjam OracleCommand objekta. Ukoliko koristimo MySQL data provider, SQL naredbe se izvršavaju korišćenjem instanci klase MySqlCommand, tj korišćenjem MySqlCommand objekta, …

SQL naredbe se izvršavaju korišćenjem instanci klase ***Command** tj korišćenjem ***Command** objekta. (**SqlCommand / OleDbCommand / OracleCommand / MySqlCommand …** zavisi od verzije)

(u nastavku se neće naglašavati koja je varijanta – umesto * u nastavku će se koristiti MySql, dok za druge varijante taj nastavak treba promeniti u Sql za MS SQL odnosno u Oracle za Oracle SQL.)

Objekti klase **MySqlCommand** predstavljaju objektno-orijentisanu reprezentaciju SQL upita, imena tabela ili uskladištenih procedura. Dakle, komande mogu biti različitog tipa. Tip komande specificiran je **CommandType** atributom objekta klase **MySqlCommand** i može imati neku od vrednosti iz **ComandType** enumeracije.

Enumeracija tipova komandi

```
public enum CommandType
{
        StoredProcedure,
        TableDirect,
        Text
}
```

Prilikom kreiranja komande moguće je navesti SQL upit kao argument konstruktora objekta ili korišćenjem **CommandText** atributa **MySqlCommand** objekta. Prilikom kreiranja komande neophodno je navesti konekciju koja će se koristiti za izvršenje komande. Konekciju je moguće navesti kao argument konstruktora objekta ili korišćenjem **Connection** atributa **MySqlCommand** objekta.

Kreiranje komande kod MySQL:

```
MySqlConnection conn = new MySqlConnection();


conn.ConnectionString = "server=localhost;user id=ucenik;password=test;database=proba";

String strSQL = "Select * from RADNIK";


MySqlCommand newComm = new MySqlCommand();

newComm.Connection = conn;
newComm.CommandText = strSQL;
```

Prosto kreiranje MySqlCommand objekta, ili bilo koje druge komande, ne znači da je SQL upit sadržan u komandi automatski prosleđen na izvršenje. Objekat koji predstavlja komandu je nakon kreiranja samo pripremljen za dalju upotrebu. Najbitnije metode članice MySqlCommand objekta prikazane su u tabeli.

Metoda	Značenje
Cancel()	Poništava izvršenje komande
ExecuteReader()	Povratna vrednost funkcije je DataReader objekat izabranog data provider-a
ExecuteNonQuery()	Izvršava komandu od koje se ne očekuje da kao povratne vrednosti daje podatke
ExecuteScalar()	Varijanta ExecuteNonQuery() koja kao povratnu vrednost vraća jedan podatak

1.3.3. Korišćenje DataReader objekta

Nakon uspostavljanja konekcije ka bazi podataka i kreiranja SQL upita, neophodno je izvršiti kreirani upit kako bi se pribavili podaci. Jedan način za pribavljanje podataka je korišćenje **DataReader** objekata. **DataReader** objekti mogu samo da čitaju podatke i to red po red unapred. Imajući ovo u vidu, jasno je da je upotreba **DataReader** objekata korisna samo u situacijama kada je kreirani upit SELECT SQL naredba. **DataReader** objekti posebno se koriste kada je potrebno brzo iterirati kroz veliku količinu podataka pri čemu nije potrebno posedovati lokalne kopije jednom očitanih podataka. **DataReader** objekti kreiraju se pozivom **ExecuteReader()** metode kreirane komande. Nakon kreiranja **DataReader** objekta, jedan red rezultujuće tabele podataka pribavlja se korišćenjem **Read()** metode.

Ukoliko pretpostavimo da je kreirana konekcija ka bazi, da bi se očitali podaci iz baze podataka neophodno je napisati SQL upit koji selektuje potrebne informacije, kreirati komandu koja će izvršiti upit i kreirati **DataReader** objekat koji će pribaviti podatke. U nastavku će biti prikazan deo programskog koda koji izvršava selektovanje svih redova iz tabele RADNIK u bazi podataka PROBA.

Selektovanje podataka

```
MySqlConnection conn = new MySqlConnection();

conn.ConnectionString = "server=localhost;user id=ucenik;password=test;database=proba";
```

```
String strSQL = "Select * from RADNIK";

MySqlCommand comm = new MySqlCommand(strSQL, conn);

MySqlDataReader reader;


conn.Open();


reader = comm.ExecuteReader();

while (reader.Read())
{
    //programski kod koji vrši obradu pribavljenih podataka
}

conn.Close();
```

Pored izvršenja SQL upita koji vrše selektovanje podataka, moguće je izvršiti SQL upite koji vrše dodavanje, izmenu ili brisanje podataka iz baze podataka. U nastavku će biti prikazan deo programskog koda koji vrši dodavanje novog radnika, izmenu podataka o radniku i briše podatke o radniku u tabeli RADNIK baze podataka PROBA.

Dodavanje, izmena i brisanje podataka

```
MySqlConnection conn = new MySqlConnection();

conn.ConnectionString = "server=localhost;user id=ucenik;password=test;database=proba";

try
{

conn.Open();

//dodavanje novog radnika

String strSQL1 = "Insert into RADNIK values (123456781,'Marko', 'J', 'Petrovic', '2/9/1965',
'Obilićev Venac', 'M', '30000',333445555, 5)";

MySqlCommand comm1 = new MySqlCommand(strSQL1, conn);

comm1.ExecuteNonQuery();

//izmena podataka o radniku

String strSQL2 = "Update RADNIK set Plata=50000 where MatBr=123456781";

MySqlCommand comm2 = new MySqlCommand(strSQL2, conn);

comm2.ExecuteNonQuery();

//brisanje podataka o radniku
```

```
String strSQL3 = "Delete from RADNIK where MatBr=123456781";

MySqlCommand comm3 = new MySqlCommand(strSQL3, conn);

comm3.ExecuteNonQuery();

}

catch (Exception exc)
{
//obrada izuzetka
}

finally
{
conn.Close();
}
```

PREPORUKA: dobra je praksa koristiti **try – catch – finally** blok. Ako dođe do izuzetka (nešto ne valja – ne radi) da se izuzetak obradi, i u svakom slučaju, da se na kraju zatvori konekcija.

Često je neophodno parametrizovati upite koji se prosleđuju bazi podataka na osnovu podataka koje su korisnici uneli u aplikaciju. Kako bi prosleđivanje podataka koje su korisnici uneli bilo što sigurnije, ADO.NET poseduje mogućnost kreiranja parametrizovanih komandi. Svaka ADO.NET komanda poseduje kolekciju individualnih parametara. Svaki od **parametara predstavlja nezavisni objekat koji je moguće kreirati i dodati u kolekciju ADO.NET Command objekta**. **Parametri zauzimaju određeno mesto u SQL upitu koji komanda izvršava**. Kako bi se ukazala pozicija u SQL upitu na kojoj će se naći određeni parametar, koristi se prefix @ praćen imenom odgovarajućeg parametra. U nastavku će biti prikazan primer kreiranja i izvršenja parametrizovane komande pri čemu se koristi **MySql** data provider. U slučaju korišćenja **MySql** data provider-a, parametri su instance klase **MySqlParameter**.

Izvršenje parametrizovane komande

```
MySqlConnection conn = new MySqlConnection();

conn.ConnectionString = "server=localhost;user id=ucenik;password=test;database=proba";

//kreiranje parametra

MySqlParameter param = new MySqlParameter();

param.ParameterName = "@parameter";

param.MySqlDbType = MySqlDbType.VarChar;

param.Value = "Marko";

//kreiranje parametrizovanog upita:

String strSQL = "Select * from RADNIK where Ime=@parameter";

MySqlCommand comm = new MySqlCommand(strSQL, conn);

//dodavanje parametra u kolekciju parametara komande

comm.Parameters.Add(param);

try
```

```
{
   conn.Open();

   MySqlDataReader dr = comm.ExecuteReader();

   while (dr.Read())
   {
      //obrada podataka
   }
}

catch (Exception exc)
{

}

finally
{
   conn.Close();
}
```

1.4. Korišćenje beskonekcionog sloja ADO.NET-a za pristup podacima

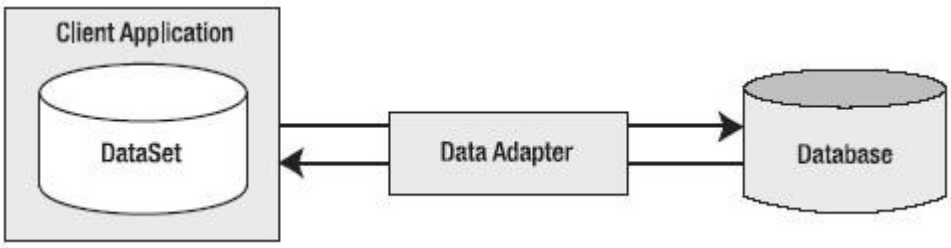
(drugim rečima, korišćenjem sloja sa raskinutom vezom)

Korišćenje **Connection**, **Command** i **DataReader** objekata podrazumeva da je veza ka izvoru informacija (bazi podataka) otvorena za sve vreme trajanja obrade podataka. Prednosti i mane ovakvog načina pristupanja podacima opisane su u prethodnim poglavljima. ADO.NET pored ovakvog načina pristupa podacima poseduje i drugačiji mehanizam, tzv. beskonekcioni pristup podacima (korišćenjem sloja sa raskinutom vezom.)

Korišćenje beskonekcionog načina pristupa podacima zasnovano je na kreiranju lokalne kopije podataka koji se čuvaju u operativnoj memoriji klijentskog računara. Korišćenjem tipova objekata iz **System.Data** ADO.NET prostora imena moguće je kreirati lokalni model podataka koji će pored sirovih podataka posedovati veze između tabela, ograničenja primenjena na kolone, primarne ključeve, poglede i sve druge karakteristike modela podataka korišćenog za kreiranje baze podataka. Lokalni model podataka dozvoljava korisnicima kreiranje i izvršavanje upita nad lokalnim podacima, njihovo filtriranje, sortiranje i snimanje u bazu podataka.

U većini slučajeva neophodno je kreirati objekte koji predstavljaju konekciju i komandu. Za pribavljanje i ažuriranje podataka se u slučaju korišćenja sloja sa raskinutom vezom za pristup podacima koristi tzv. data adapter objekat, koji predstavlja instancu **DataAdapter** klase. Za razliku od direktnog pristupa podacima, podaci pribavljeni korišćenjem data adapter objekta se ne pribavljaju korišćenjem **DataReader** objekata. Data adapter objekti koriste **DataSet** objekte za prenos podataka od i ka bazi podataka. Svaki **DataSet** objekat može biti sastavljen od većeg broja **DataTable** (tabela podataka) objekata koji poseduju kolekcije **DataRow** (red tabele) i **DataColumn** (kolona tabele) objekata.

Data adapter objekat koji se koristi upravlja konekcijom ka bazi podataka automatski. Kako bi se poboljšala funkcionalnost, data adapter objekat će konekciju ka bazi držati otvorenom minimalni period vremena. Kada se na klijentskoj strani kreira DataSet objekat, data adapter će prekinuti konekciju sa bazom podataka ostavljajući potpuno samostalnu kopiju podataka na klijentskom računaru. Klijentska aplikacija može nakon toga nesmetano vršiti izmene nad pribavljenom kopijom podataka. Ovaj proces prikazan je na slici.



Da bi kreirali lokalnu kopiju podataka korišćenjem data adapter-a i **DataSet** objekata, neophodno je najpre kreirati konekciju ka bazi podataka i komandu na osnovu koje će biti kreirana lokalna kopija podataka. U nastavku je prikazan primer programskog koda koji vrši učitavanje svih radnika iz tabele radnik korišćenjem MySql data adapter-a.

Korišćenje DataAdapter objekta

```
MySqlConnection conn = new MySqlConnection();

conn.ConnectionString = "server=localhost;user id=ucenik;password=test;database=proba";

String strSQL = "Select * from RADNIK";

MySqlCommand comm = new MySqlCommand(strSQL, conn);

MySqlDataAdapter adapter = new MySqlDataAdapter(comm);

DataSet ds = new DataSet();

try
{

   conn.Open();

   adapter.Fill(ds, "Radnici");

}

catch (Exception exc)
{
   //obrada izuzetka
}

finally
{
   conn.Close();
}
```

Nakon kreiranja lokalne kopije podataka tj **DataSet** objekta, moguće je ažurirati pribavljene podatke. U nastavku je prikazan primer programskog koda koji vrši ažuriranje podataka o sivm radnicima čije je ime Milos postavljajući im platu na vrednost 30000.

Ažuriranje podataka

```
foreach (DataRow dr in ds.Tables["Radnici"].Rows)
{
```

```
    if (dr["Ime"].ToString() == "Milos")
    {

        dr["Plata"] = 30000;

     }
}
```

Moguće je takođe i obrisati neki od redova u lokalnoj kopiji podataka ili dodati novi red.

Brisanje i dodavanje podataka

```
foreach(DataRow dr in ds.Tables["Radnici"].Rows)
{

    if(dr["Ime"].ToString() == "Milos")

    {

        dr["Prezime"] = "Bogdanović";

        dr["SSlovo"] = "M";

        dr["DatRodj"] = "1/1/2011";

        dr["Adresa"] = "Bulevar Nemanjića 5/11";

        dr["Plata"] = 20000;


        DataRow row = ds.Tables["Radnici"].NewRow();

        row["MatBr"] = 223456789;

        row["Ime"] = "Milos";

        row["Prezime"] = "Obilic";

        row["SSlovo"] = "M";

        row["DatRodj"] = "1/1/2011";

        row["Adresa"] = "Bulevar Nemanjića 8/11";

        row["Plata"] = 30000;

        ds.Tables["Radnici"].Rows.Add(row);

    }
}
```

Sve izmene učinjene na lokalnoj kopiji podataka mogu se snimiti u bazu podataka korišćenjem ***Update()*** metode **DataAdapter** objekta.

Snimanje podataka

```
MySqlCommandBuilder myMySqlCommandBuilder = new MySqlCommandBuilder(adapter);

adapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;

int izmenjeniRedovi = adapter.Update(ds,"Radnici");
```

ADO.Net održava informacije o originalnim i trenutnim vrednostima svih podataka koji se nalaze u **DataSet** objektu. Prilikom ažuriranja podataka, ADO.NET traži redove koji u potpunosti odgovaraju originalnim vrednostima podataka u redovima DataSet objekta i po pronalaženju ih zamenjuje novim redovima tj podacima iz redova lokalne kopije. U ovom trenutku lako je uočiti da će problem nastati ukoliko neki drugi korisnik izmeni podatke pre nego što naša aplikacija snimi izmenjene podatke. Naime, drugi korisnici mogu izmeniti podatke u bazi pa prilikom upoređivanja originalnih vrednosti lokalnih kopija podataka i podataka u bazi neće doći do poklapanja. U ovakvim situacijama desiće se izuzetak. Ovakve situacije moguće je predupredi korišćenjem **DataAdapter.RowUpdated** događaja. Ovaj događaj dešava se prilikom izvršenja svake insert, update ili delete operacije ali pre nego štio se desi izuzetak pa nam daje mogućnost da sprečimo dešavanje izuzetka. **DataAdapter.RowUpdated** događaj dešava se u trenucima dok aplikacija radi sa DataSet objektom i otvara novu konekciju ka bazi podataka pa je preporučljivo da programski kod koji se izvršava u ovim situacijama bude što manji.

1.5. Pitanja

Pokušajte da odgovorite na sledeća pitanja. Nakon toga pogledajte ponovo materijal u ovoj lekciji. Za svaki tačan odgovor dodelite sebi 2 poena, za delimično tačan 1, a za netačan 0. Pogledajte ponovo one delove lekcije za koje ste imali 0 poena.

1. Na kom novom objektu se bazira najveći deo funkcionalnosti ADO.NET-a?
2. Šta je data provider?
3. U kojim prostorima imena se nalaze roditeljske klase svih ADO.NET data provider-a?
4. Navesti osnovne članove System.Data prostora imena.
5. U kojim situacijama je dobro koristiti direktan pristup podacima korišćenjem ADO.NET-a?
6. Koji je uobičajeni redosled operacija prilikom direktnog pristupa podacima korišćenjem ADO.NET-a?
7. Koja klasa enkapsulira konekciju ka izvoru podataka u ADO.NET-u?
8. Šta predstavlja ConnectionString atribut klase Connection?
9. Ukoliko se koristi OLE DB data provider, koja klasa predstavlja objektno-orijentisanu reprezentaciju SQL komande koju je neophodno izvršiti?
10. Čemu služe instance klase DataReader i na koji način je moguće instancirati objekte ove klase?
11. Šta predstavljaju parametri SQL komande?
12. Ukoliko se koristi OLE DB data provider, na koji način je moguće ukazati na poziciju koju parametar zauzima u SQL upitu?
13. Na čemu se bazira korišćenje beskonekcionog pristupa podacima korišćenjem ADO.NET-a?
14. Koja se klasa koristi za instanciranje objekata koji omogućavaju pribavljanje i ažuriranje objekata?
15. Koja klasa se koristi za instanciranje objekata koji vrše prenos podataka od i ka bazi podataka kod beskonekcionog pristupa ADO.NET-a?

1.6. Zadatak

ADO.NET

1. Korišćenjem ADO.NET direktnog pritupa podacima pronaći radnika čiji je matični broj 123456789 i izmeniti njegovo ime na “Aleksandar”. Za pristup podacima koristiti MySQL data provider. Podaci o radnicima čuvaju se u tabeli RADNIK baze podataka PROBA.

2. Korišćenjem beskonekcionog sloja ADO.NET-a kreirati lokalnu kopiju podataka o svim radnicima, povećati svim radnicima platu za 15% i snimiti izmenjene podatke. Podaci o radnicima čuvaju se u tabeli RADNIK baze podataka PROBA.

LITERATURA

Leonid Stoimenov, *UVOD U BAZE PODATAKA*, Univerzitet u Nišu, Elektronski fakultet, Niš, 2013
(https://edoc.pub/leonid-stoimenov-uvod-u-baze-podatakapdf-pdf-free.html)