Uvod u ADO.NET

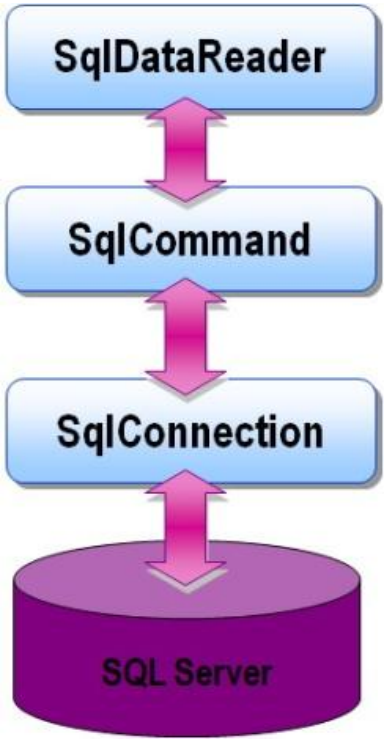
ADO.NET

- ADO.NET je **skup klasa za rad sa podacima**
- .NET snabdevači podataka su klase koje obezbeđuju mogućnost povezivanja sa izvorima podataka:
  - SQL Server snabdevač podataka
  - OLE DB snabdevač podataka
  - ostali snabdevači podataka
- **Prostori imena** za rad sa podacima su:
  - System.Data
  - System.Data.SqlClient
  - System.Data.OleDb
  - System.Data.SqlTypes
  - System.Xml

Konektovani scenario

- Resursi se uzimaju sa servera **sve dok se konekcija ne zatvori**
- Korisnik je konstantno povezan sa izvorom podataka
- Podaci su **ažurni**
- Konkurentnost se lakše kontroliše
- Mora postojati **konstantna mrežna konekacija**

Korišćenje ADO.NET klasa u konektovanom scenariju



- Otvaranje konekcije
- Izvršavanje komande
- Zatvaranje konekcije

Konekcije

- Pre bilo kakvog rada sa bazom podataka potrebno je **kreirati** a zatim **otvoriti** konekciju
- U ADO.NET se kreira objekat klase XxxConnection
- System.Data.SqlClient.**SqlConnection** omogućava kreiranje konekcije ka **SQL Server** bazi podataka
- System.Data.OleDb.**OleDbConnection** omogućava kreiranje konekcije ka svakom izvoru podataka koji ima pridruženi **OLE DB** provajder

SqlConnection klasa

```
public abstract class DbConnection : Component, IDbConnection, IDisposable
```

- Sql**Connection** klasa je izvedena iz klase **DbConnection** koja se nalazi u prostoru imena System.Data.Common
- Klasa DbConnection predstavlja **apstrakciju konekcije ka bazi podataka**
- Konkretne implementacije klase DbConnection:
  - SqlConnection – konekcija ka SQL Server bazi podataka
  - OleDbConnection – konekcija ka OLE DB izvorima podataka
  - OdbcConnection – konekcija ka ODBC izvorima podataka

Svojstva klase SqlConnection

- **ConnectionString**
  - Sadrži informacije potrebne za povezivanje sa bazom podataka (server, baza, autentifikacija, itd.)
- **State**
  - Prikazuje trenutno stanje konekcije (Closed, Open, Connecting, Executing, Fetching)
- **Database**
  - Naziv baze podataka na koju je uspostavljena konekcija
- **DataSource**
  - Naziv servera (ili instance) na koji se konekcija ostvaruje

Metode klase SqlConnection

- **Open**()
  - Otvara konekciju ka bazi podataka
- **Close**()
  - Zatvara konekciju ka bazi podataka
- **Dispose**()
  - Oslobađa resurse koje konekcija koristi
  - Poziva se automatski pri korišćenju using bloka
- **CreateCommand**()
  - Kreira objekat klase SqlCommand vezan za trenutnu konekciju

Baza Magacin

```
CREATE DATABASE Magacin
COLLATE Serbian_Latin_100_CI_AI
GO

USE Magacin
GO

CREATE TABLE Kategorija
(
    KategorijaId INT IDENTITY(1,1) PRIMARY KEY,
    Naziv NVARCHAR(70) NOT NULL UNIQUE,
    Opis NVARCHAR(120) NULL
)
GO

CREATE TABLE Proizvod
(
    ProizvodId INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
    KategorijaId INT NOT NULL
        FOREIGN KEY REFERENCES Kategorija(KategorijaId),
    Naziv NVARCHAR(120) NOT NULL UNIQUE,
    Cena DECIMAL(10,2) NOT NULL,
    Opis NVARCHAR(120) NULL
)
GO
```

Konekcija kod Windows autentifikacije

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

```
SqlConnection con = new SqlConnection();
con.ConnectionString = "konekcioni string";

SqlConnection con = new SqlConnection("konekcioni string");
```

Windows autentifikacija:

```
string konekcioniString = @"Data Source=(local)\SqlExpress;
Initial Catalog=Magacin;Integrated Security=true";
```

Data Source – ime SQL Servera (ili instance)
Initial Catalog – ime baze podataka sa kojom se radi
Integrated Security=true – koristi Windows autentifikaciju

Konekcija kod SQL Server autentifikacije

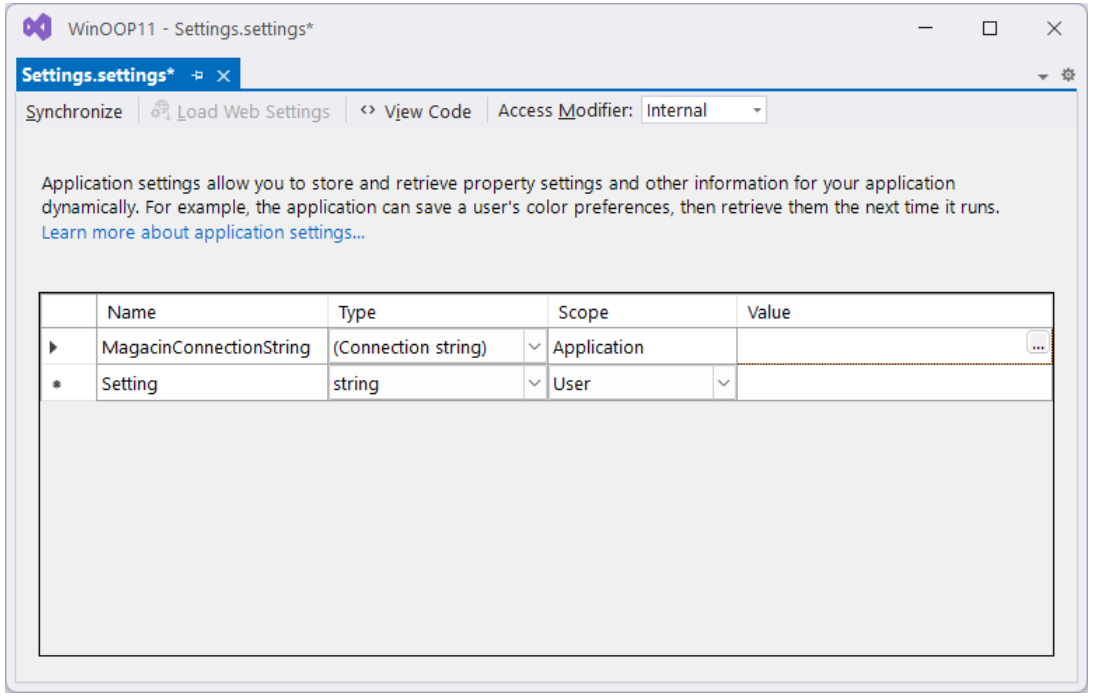
```
//SQL Server autentifikacija
string konekcioniString = @"Data Source=(local)\SqlExpress;
Initial Catalog=Magacin;User ID=sa;Password=****";
```

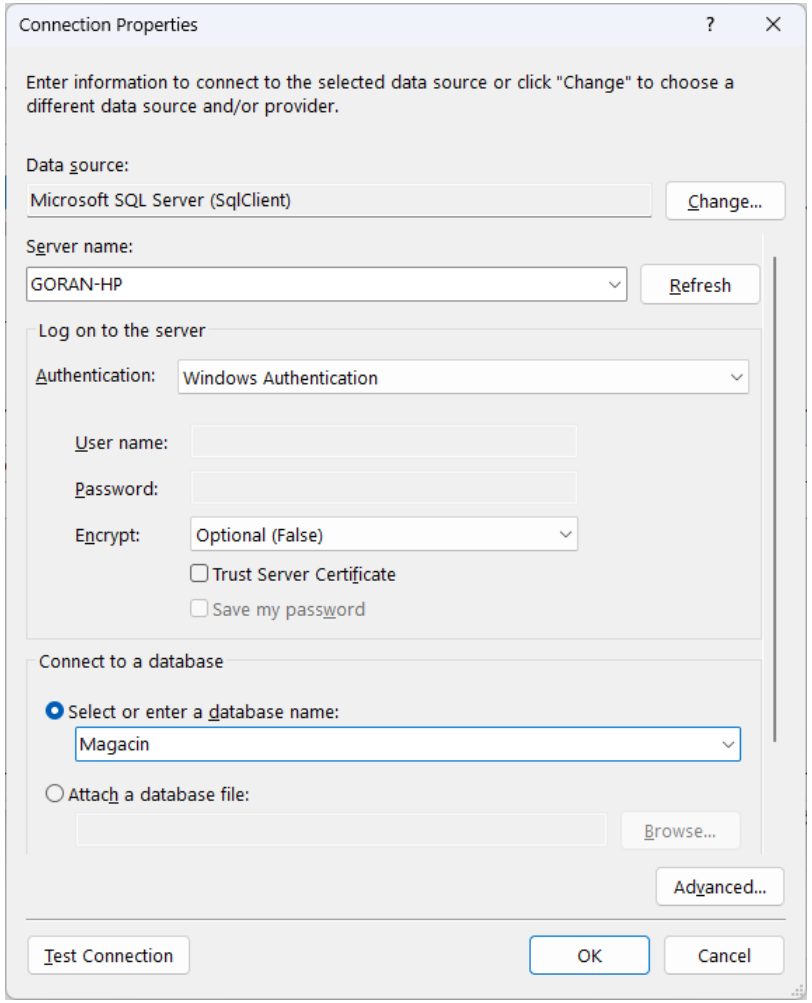
Rad u konektovanom okruženju

Upis konekcionog stringa u aplikaciona setovanja

- Konekcioni string se može sačuvati u Application Settings
- Na ovaj način se izbegava hard-kodiranje konekcionog stringa u kodu
- Podešavanje se vrši u fajlu **Settings.settings**
- Tip podešavanja: Connection string
- Scope: Application
- Prednosti:
  - Lakša promena konekcionog stringa
  - Centralizovano upravljanje podešavanjima
  - Veća bezbednost i čitljivost koda

Upis konekcionog stringa u aplikaciona setovanja





Klasa Konekcija

```
static class Konekcija
{

    public static string CnnMagacin
    {
        get
        {
            return Properties.Settings.Default.MagacinConnectionString;
        }
    }
}
```

```
static class Konekcija
{

    public static string CnnMagacin =>
        Properties.Settings.Default.MagacinConnectionString;
}
```

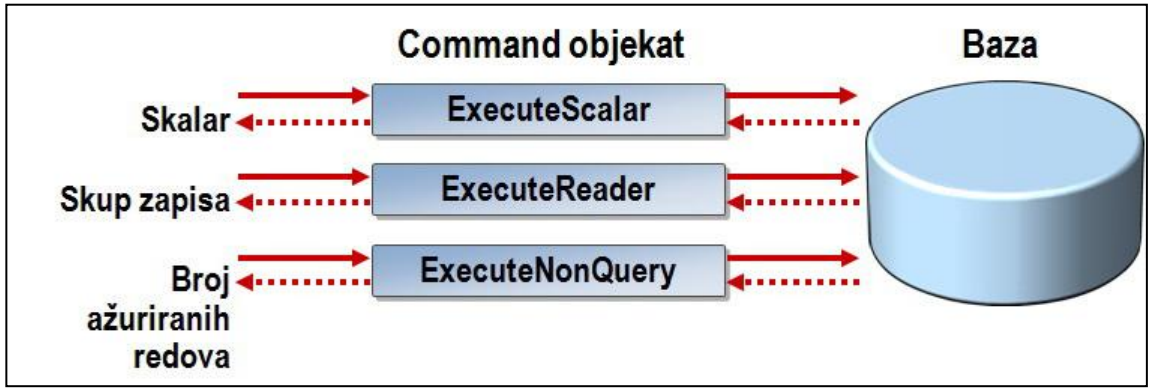
Klasa SqlCommand

- Koristi se za izvršavanje SQL komandi i uskladištenih procedura nad bazom podataka
- Objekat SqlCommand omogućava direktan pristup podacima u bazi u **konektovanom** okruženju
- Pomoću nje moguće je izvršavati upite:
  - SELECT
  - INSERT
  - UPDATE
  - DELETE

Svojstva klase SqlCommand

- **CommandText**
  - SQL naredba ili naziv uskladištene procedure koja se izvršava
- **Connection**
  - Referenca na objekat klase SqlConnection nad kojim se komanda izvršava
- **CommandType**
  - Text -SQL komanda (podrazumevana vrednost)
  - StoredProcedure
- **Parameters**
  - Kolekcija parametara koji se prosleđuju SQL komandi ili proceduru
- **Transaction**
  - Transakcija u okviru koje se komanda izvršava
- **CommandTimeout**
  - Maksimalno vreme (u sekundama) čekanja na izvršavanje komande

Metode SqlCommand klase



Metode SqlCommand klase

- **ExecuteNonQuery**() – izvršava SQL komande i uskladištene procedure koje ne vraćaju podatke (INSERT, UPDATE, DELETE)
- **ExecuteReader**() – izvršava komande koje vraćaju tabelarne podatke (SELECT) i vraća objekat tipa SqlDataReader
- **ExecuteScalar**() – izvršava SQL komande i uskladištene procedure koje vraćaju jednu skalarnu vrednost (COUNT, SUM, MAX)

Korišćenje using bloka

- **using** blok se koristi za automatsko oslobađanje resursa
- Najčešće se koristi sa objektima koji implementiraju interfejs **IDisposable**
- Nakon izlaska iz using bloka automatski se poziva metoda **Dispose**()
- Kod rada sa bazom podataka obezbeđuje pravilno zatvaranje:
  - konekcije
  - komand
  - ičitača podataka
- using ne mora da ima sopstveni blok { }
  - Ako se više using naredbi napiše jedna ispod druge,svi dele isti blok koda
  - Resursi se oslobađaju obrnutim redosledom kreiranja

Klasa DBNull i svojstvo Value

- Klasa DBNull predstavlja SQL NULL vrednost u .NET okruženju
- Koristi se pri radu sa bazama podataka (ADO.NET)
- Razlikuje se od null u C# jeziku
- **DBNull.Value** je jedina instanca klase DBNull
- Predstavlja konkretnu **SQL NULL** vrednost
- Uvek se koristi za proveru da li je vrednost iz baze NULL

Kreiranje i izvršavanje komande koja vraća skalarnu vrednost

```
private void button1_Click(object sender, EventArgs e)
{
    decimal cena = 0m;
    string poruka = "";
    try
    {
        using (SqlConnection konekcija = new SqlConnection(Konekcija.CnnMagacin))
        using (SqlCommand komanda = new SqlCommand("SELECT AVG(Cena) FROM Proizvod", konekcija))
        {
            konekcija.Open();
            object rezultat = komanda.ExecuteScalar();

            if (rezultat == DBNull.Value)
            {
                poruka = "Nema podataka u tabeli Proizvod.";
            }
            else
            {
                cena = Convert.ToDecimal(rezultat);
            }
        }
    }
    catch (Exception xcp)
    {
        poruka = xcp.Message;
    }
    ....
```

Kreiranje i izvršavanje komande koja vraća skalarnu vrednost

```
 if (poruka != "")
    {
        MessageBox.Show(poruka);
    }
    else
    {
        MessageBox.Show("Prosečna cena je: " + cena);
    }
}
```

Ažuriranje baze podataka

```
private void button2_Click(object sender, EventArgs e)
{
    string upit = @"UPDATE Kategorija
            SET Naziv = 'Sokovi'
            WHERE KategorijaId = 1";
    string greska = "";

    using (SqlConnection konekcija = new SqlConnection(Konekcija.CnnMagacin))

    using (SqlCommand komanda = new SqlCommand(upit, konekcija))
    {
        try
        {
            konekcija.Open();
            komanda.ExecuteNonQuery();
        }
        catch (Exception xcp)
        {
            greska = xcp.Message;
        }
    }
    if (greska != "")
    {
        MessageBox.Show(greska);
        return;
    }
    MessageBox.Show("Promenjena kategorija");
}
```

SqlDataReader objekat

SqlDataReader

- SqlDataReader je brz, read-only i forward-only kursor koji se kreće kroz skup zapisa
- Pristup podacima korišćenjem objekta SqlDataReader sastoji se od sledećih koraka:
  - Kreira se objekat SqlConnection
  - Kreira se objekat SqlCommand sa odgovarajućim SELECT upitom
  - Otvara se konekcija
  - Poziva se metoda SqlCommand.ExecuteReader() koja vraća objekat SqlDataReader
  - Korišćenjem metode Read() prolazi se kroz sve redove rezultata
  - Kada metoda Read() vrati false, završava se čitanje podataka i zatvara konekcija

Svojstva i metode klase SqlDataReader

- **Read**() metodaVrši pozicioniranje na sledeći red rezultata
  - Vraća true ako postoji još redova za čitanje
  - Vraća false ako je dostignut kraj skupa zapisa
- **Podrazumevana pozicija** SqlDataReader-a je ispred prvog reda tabele
- Pristup vrednostima kolona u redu na koji SqlDataReader trenutno pokazuje:
  - Vrednosti su u izvornom formatu tipa object
  - Potrebno je izvršiti kastovanje ili konverziju da bi se koristile
  - Pristup po imenu ili po indeksu kolone:
    - dr["ImeKolone"]
    - dr[pozicijaKolone]
  - Tipizirane metode za čitanje vrednosti:
  - dr.GetDateTime(0)
  - dr.GetDouble(0)
  - dr.GetInt32(1)
  - itd.

Entitetska klasa

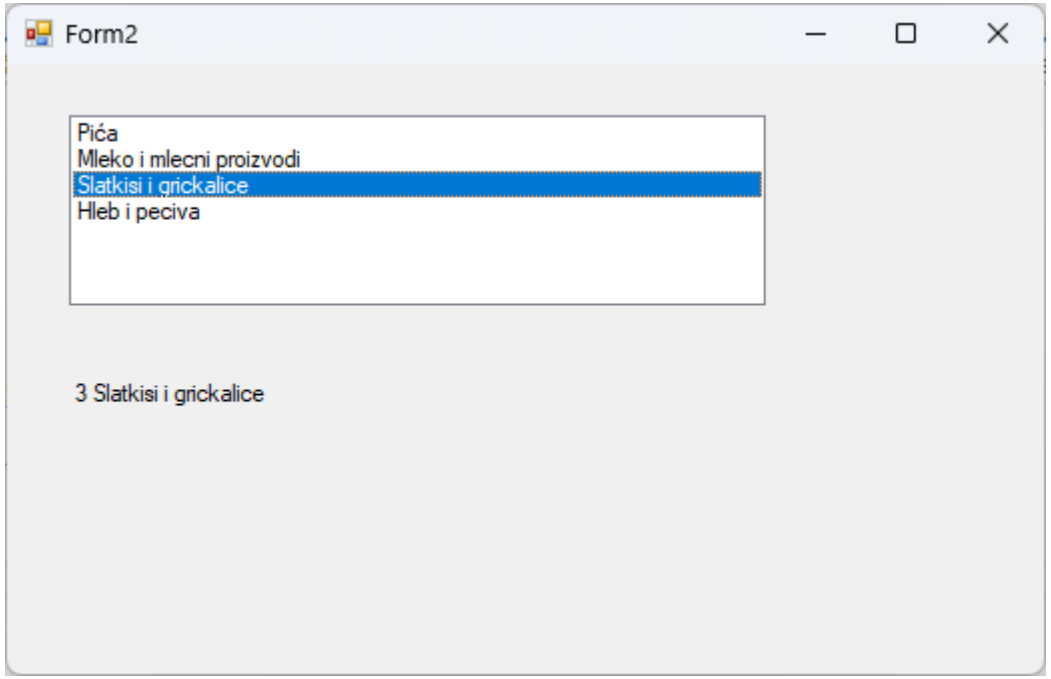
```
internal class Kategorija
{
    public int KategorijaId { get; set; }
    public string Naziv { get; set; }
    public string Opis { get; set; }

    public override string ToString()
    {
        return Naziv;
    }
}
```

Metoda VratiKategorije()

```
private List<Kategorija> VratiKategorije()
{
    List<Kategorija> lista = new List<Kategorija>();

    using (SqlConnection konekcija = new SqlConnection(Konekcija.CnnMagacin))
    using (SqlCommand komanda = new SqlCommand("SELECT * FROM Kategorija", konekcija))
    {
        konekcija.Open();
        SqlDataReader dr = komanda.ExecuteReader();
        while (dr.Read())
        {
            lista.Add(new Kategorija
            {
                KategorijaId = dr.GetInt32(0),
                Naziv = dr.GetString(1),
                Opis = dr.IsDBNull(2) ? "" : dr.GetString(2)
            });
        }

        dr.Close();
    }

    return lista;
}
```

Prikaz podataka u ListBox kontroli

```
private void Form2_Load(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    listBox1.Items.AddRange(VratiKategorije().ToArray());
}
```



SelectedIndexChanged događaj

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (listBox1.SelectedIndex > -1)
    {
        Kategorija k = listBox1.SelectedItem as Kategorija;
        label1.Text = k.KategorijaId + " " + k.Naziv;
    }
}
```

Rad sa parametrizovanim SQL upitima

Parametri u SQL komandama

- Parametri se mogu posmatrati kao promenljive koje se koriste za razmenu podataka između aplikacije i baze podataka
- Koriste se za prosleđivanje vrednosti u SQL komande na bezbedan i kontrolisan način
- Ime parametra počinje znakom @
- Parametri važe samo u okviru SQL komande u kojoj su definisani
- Tip parametra se definiše korišćenjem standardne enumeracijeSystem.Data.SqlDbType
- Tipičan primer korišćenja parametara je WHERE klauzula u SQL upitu

Izvršavanje parametarskog SQL upita

```
--parametarski SELECT upit
SELECT  ProizvodId, Naziv, Cena
FROM Proizvod
WHERE KategorijaId = @KategorijaId
```

```
DECLARE @KategorijaId INT
SET @KategorijaId = 1;


--parametarski SELECT upit
SELECT  ProizvodId, Naziv, Cena
FROM Proizvod
WHERE KategorijaId = @KategorijaId
```

Tipovi parametara

- Input
  - Ulazni parametri
  - Koriste se za slanje podataka iz aplikacije ka bazi podataka
  - Najčešće korišćen tip parametra
- Izlazni parametri
  - Koriste se za prihvatanje podataka iz baze nakon izvršavanja komande
  - Često se koriste u uskladištenim procedurama
- InputOutput
  - Ulazno/izlazni parametri
  - Omogućavaju slanje početne vrednosti i prijem izmenjene vrednosti iz baze

Kreiranje ulaznih parametara

- Ulazni parametri se mogu kreirati na dva načina
- Prvi način koristi eksplicitno definisan objekat SqlParameter
- Drugi način koristi skraćenu metodu AddWithValue
- Oba načina sprečavaju SQL Injection napade
- U praksi se oba načina sreću, izbor zavisi od potrebe i konteksta

Kreiranje ulaznih parametara

```
 SqlCommand komanda = new SqlCommand(
  "INSERT INTO Kategorija VALUES (@Naziv, @Opis)", konekcija);

//parameter naziv
SqlParameter p1 = new SqlParameter("@Naziv", SqlDbType.NVarChar, 70);
komanda.Parameters.Add(p1);
p1.Value = textBox1.Text;
```

```
komanda.Parameters.AddWithValue("@Naziv", textBox1.Text);
```

Kreiranje izlaznog parametra

- Izlazni parametar se koristi za vraćanje vrednosti iz SQL komande ili procedure
- Najčešće se koristi kod uskladištenih procedura
- Tip parametra se postavlja kao **Output**
- Vrednost izlaznog parametra se može pročitati nakon izvršenja SQL komande
- Izlazni parametri se koriste za:
  - vraćanje generisanih ID vrednosti
  - vraćanje statusa izvršenja
  - vraćanje jedne konkretne vrednosti iz baze

Kreiranje izlaznog parametra

```
string upit =
@"INSERT INTO Kategorija (Naziv, Opis)
    VALUES (@Naziv, @Opis);

    SET @KategorijaId = CAST(SCOPE_IDENTITY() AS int);";

SqlConnection konekcija = new SqlConnection(Konekcija.CnnMagacin);
SqlCommand cmd = new SqlCommand( upit,konekcija);

// izlazni parametar
SqlParameter param = new SqlParameter("@KategorijaId", SqlDbType.Int);
param.Direction = ParameterDirection.Output;
cmd.Parameters.Add(param);
```

Parametarska INSERT komanda

```
 static int UbaciKategoriju(Kategorija k)
{
    const string upit =
    @"INSERT INTO Kategorija (Naziv, Opis)
    VALUES (@Naziv, @Opis);

    SET @KategorijaId = CAST(SCOPE_IDENTITY() AS int);";

    try
    {
        using (SqlConnection konekcija = new SqlConnection(Konekcija.CnnMagacin))
        using (SqlCommand komanda = new SqlCommand(upit, konekcija))
        {
            // ulazni parametri
            komanda.Parameters.AddWithValue("@Naziv", k.Naziv);
            komanda.Parameters.AddWithValue("@Opis", k.Opis ?? (object)DBNull.Value);

            // izlazni parametar
            SqlParameter izlazniId = new SqlParameter("@KategorijaId", SqlDbType.Int);
            izlazniId.Direction = ParameterDirection.Output;
            komanda.Parameters.Add(izlazniId);

            konekcija.Open();
            komanda.ExecuteNonQuery();

            return (int)komanda.Parameters["@NoviId"].Value;
        }
    }
    catch (Exception)
    {
        return -1;
    }
}
```

Parametarska INSERT komanda

```
public static int UbaciKategoriju(Kategorija k)
{
    const string upit =
        @"INSERT INTO Kategorija (Naziv, Opis)
        VALUES (@Naziv, @Opis);
        SELECT CAST(SCOPE_IDENTITY() AS int);";

    try
    {
        using (SqlConnection konekcija = new SqlConnection(Konekcija.CnnMagacin))
        using (SqlCommand komanda = new SqlCommand(upit, konekcija))
        {
            komanda.Parameters.AddWithValue("@Naziv", k.Naziv);
            komanda.Parameters.AddWithValue("@Opis", k.Opis ?? (object)DBNull.Value);


            konekcija.Open();
            return (int)komanda.ExecuteScalar();
        }
    }
    catch (Exception)
    {
        return -1;
    }
}
```

Parametarska UPDATE komanda

```
public static int PromeniKategoriju(Kategorija k)
{
    const string upit =
        @"UPDATE Kategorija
    SET Naziv = @Naziv, Opis = @Opis
    WHERE KategorijaId = @KategorijaId";

    try
    {
        using (SqlConnection konekcija = new SqlConnection(Konekcija.cnnMagacin))
        using (SqlCommand komanda = new SqlCommand(upit, konekcija))
        {
            komanda.Parameters.AddWithValue("@Naziv", k.Naziv);
            komanda.Parameters.AddWithValue("@Opis", (object)k.Opis ?? DBNull.Value);
            komanda.Parameters.AddWithValue("@KategorijaId", k.KategorijaId);

            konekcija.Open();
            komanda.ExecuteNonQuery();
            return 0;
        }
    }
    catch (Exception)
    {
        return -1;
    }
}
```

Parametarska DELETE komanda

```
public static int ObrisiKategoriju(int id)
{
    const string upit =
        @"DELETE FROM Kategorija
        WHERE KategorijaId = @KategorijaId";

    try
    {
        using (SqlConnection konekcija = new SqlConnection(Konekcija.cnnMagacin))
        using (SqlCommand komanda = new SqlCommand(upit, konekcija))
        {
            komanda.Parameters.AddWithValue("@KategorijaId", id);

            konekcija.Open();
            komanda.ExecuteNonQuery();
            return 0;
        }
    }
    catch (Exception)
    {
        return -1;
    }
}
```

Pitanje 1

U konektovanom scenariju

a. resursi se uzimaju a servera tek kada se konekcija zatvori
b. resursi se uzimaju sa servera dok je konekcija otvorena
c. kreira se lokalna kopija podataka iz baze

Odgovor: b

Pitanje 2

Za uspostavljanje konekcije sa SQL Server bazom podataka koristi se objekat klase:

a. SqlConnection
b. OleDbConnection
c. SqlServerConnection

Odgovor: a

Pitanje 3

Atribut Data Source u konekcionom stringu definiše:

a. bazu podataka sa kojom se uspostavlja konekcija
b. tabelu u bazi sa kojom se uspostavlja konekcija
c. database server sa kojim se uspostavlja konekcija

Odgovor: c

Pitanje 4

Da bi se koristio SQL Server.NET snabdevač podataka potrebno je ulkjučiti prostor imena:

a. System.Data.SqlTypes
b. System.Data.SqlClient
c. System.Data.SqlServer

Odgovor: b

Pitanje 5

SqlDataReader objekat se kreira:

a. pozivom konstruktora SqlDataReader
b. pozivom metode CreateReader() objekta SqlConnection
c. pozivom metode ExecuteReader() objekta SqlConnection
d. pozivom metode ExecuteReader() objekta SqlCommand

Odgovor: d

Pitanje 6

Za izvršavanje upita nad bazom podataka koji treba da vrati jednu numeričku vrednost koristimo metodu objekta SqlCommand:

a. ExecutDataSet()
b. ExecuteReader()
c. ExecuteScalar()
d. ExecuteNonQuery()

Odgovor: c

Pitanje 7

Ukoliko se drugačije ne naznači, podrazumevana vrednost svojstva CommandType objekta SqlCommand je:

a. CommandType.Text
b. Text
c. CommandType.StoredProcedure
d. StoredProcedute

Odgovor: a

Pitanje 8

Izlaznom parametru objekta SqlCommand:

a. ne dodeljuje se vrednost pre izvršavanja odgovarajuće komande
b. dodeljuje se vrednost pre izvršavanja odgovarajuće komande
c. dodeljuje se vrednost pre izvršavanja odgovarajuće komande samo ako postoje I ulazni parametri

Odgovor: a

Pitanje 9

Ulaznom parametru objekta SqlCommand:

a. ne dodeljuje se vrednost pre izvršavanja odgovarajuće komande
b. mora se dodeliti vrednost pre izvršavanja odgovarajuće komande
c. vrednost se može ali ne mora dodeliti

Odgovor: b

Pitanje 10

Dat je sledeći C# kod:

```
SqlParameter CityParameter = new SqlParameter("@City", SqlDbType.NVarChar, 15);
cmdKorisnikIzGrada.Parameters.Add(CityParameter);
CityParameter.Value = textBox1.Text;
```

Ovim kodom kreira se:

a. ulazni parametar
b. izlazni
c. parametar koji može biti i ulazni i izlazni

Odgovor: a