

# Aplikacija nad bazom podataka

---

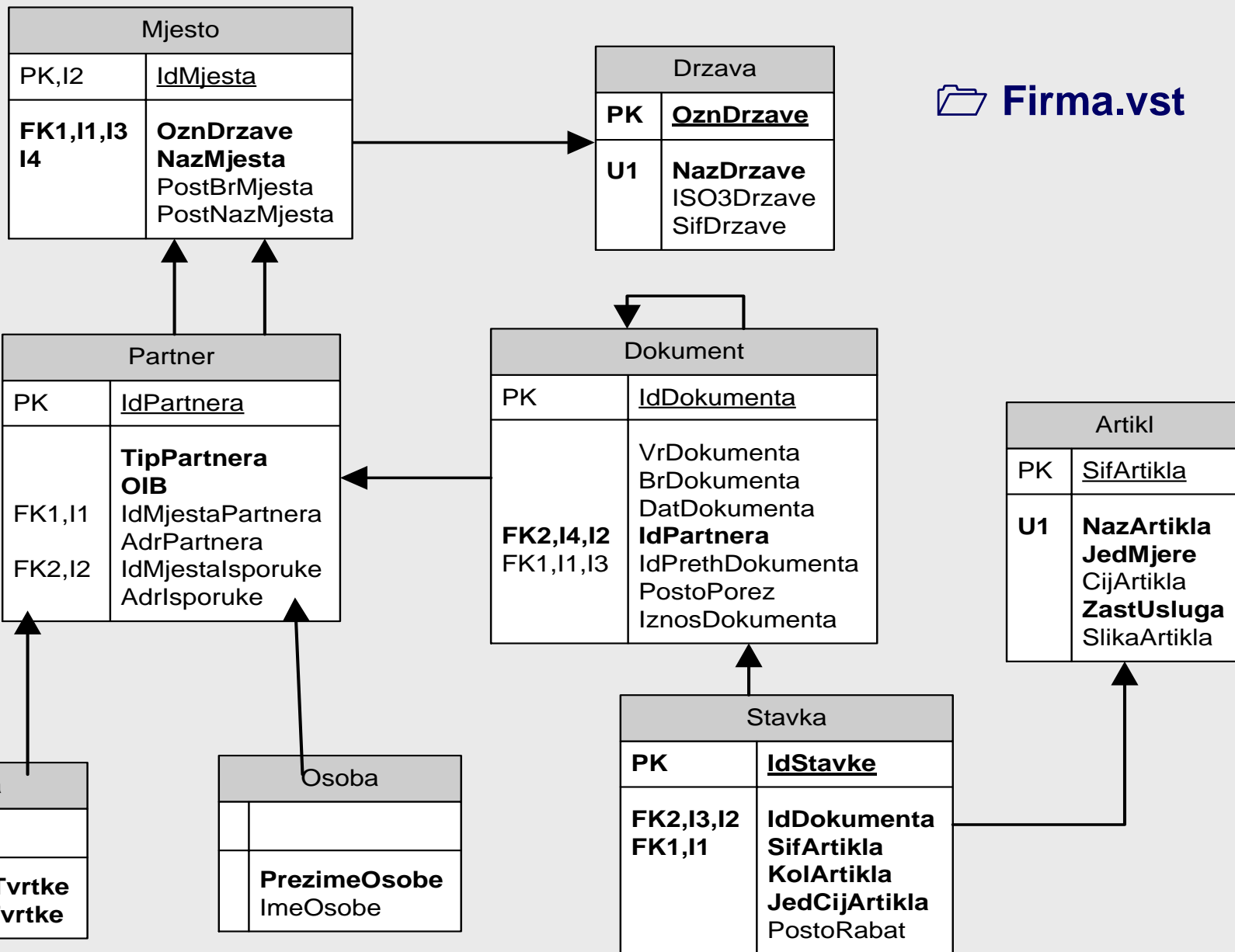
2014/15.05

# Ogledna baza podataka

- ❑ **SQL Server: rppp.fer.hr\SQL2012,3000**
- ❑ **Ogledna baza podataka: Firma**
  - SQL Server Authentication: rppp/zaporka se nalazi u primjerima na TFS-u
  - Moguće mijenjati podatke u svrhu testiranja
- ❑ **Datoteka BazePodataka.zip na TFS-u sadrži**
  - BazePodataka.txt – kratke upute
  - Firma.bak – backup baze podataka Firma
  - Firma.mdf + Firma.ldf – SQL server datoteke baze podataka Firma za prikapčanje na vlastiti server
  - Firma.mdb – MS Access baza podataka za demonstraciju OleDb
  - Firma.vst – MS Visio dijagram baze podataka
- ❑ **Baze podataka oblika RPPPxx:**
  - Baza podataka za grupu XX
  - Korisničko ime rpppXX , zaporka predana na predavanjima

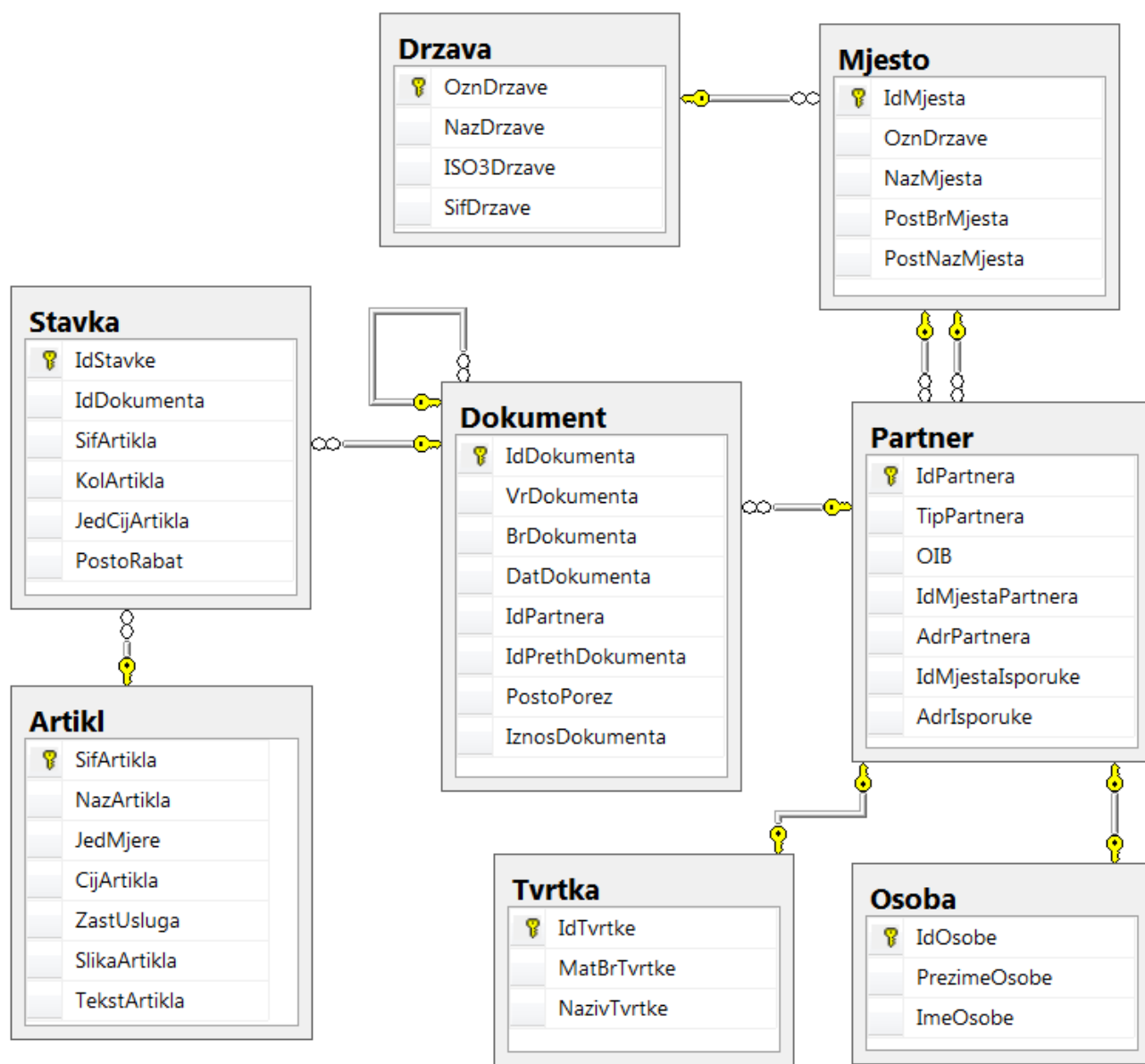
# Ogledna baza podataka (MS Visio)

 **Firma.vst**



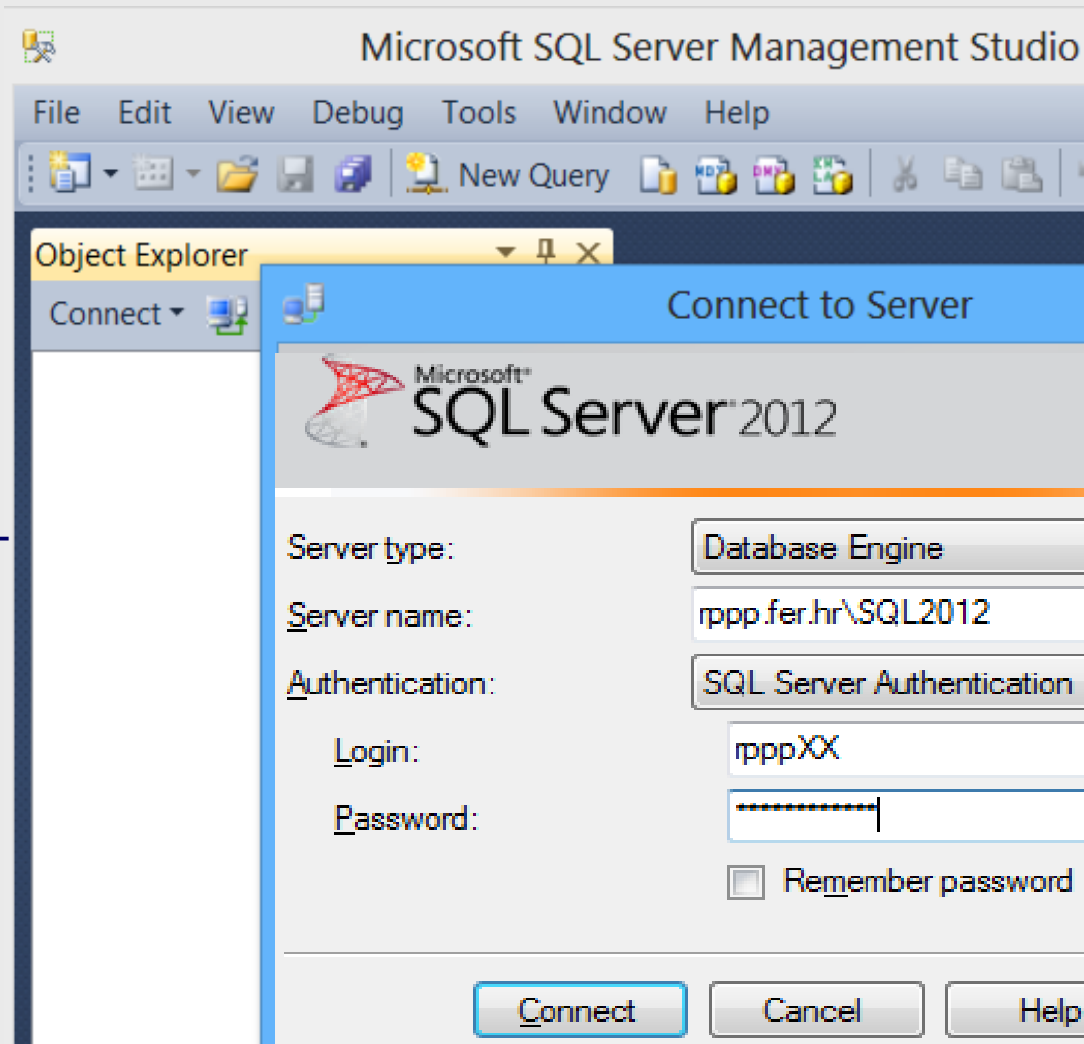
# Ogledna baza podataka (rppp.fer.hr\SQL2012,3000 - Firma)

 Firma.bak



# SQL Server Management Studio (sučelje SUBP)

- ❑ Ime servera oblika računalo\nnaziv instance (ako je naziv naveden pri instalaciji).
  - . ili (local) = "ovo" računalo
- ❑ Dva načina identificiranje korisnika
  - Windows Authentication – provjera korisnika domene oblika NazivDomene\Korisnik
  - SQL Server Authentication – provjera korisnika čije je ime definirano u SUBP
- ❑ Svaka baza na SQL Serveru ima svoje korisnike koji su povezani s korisnicima iz domene ili onima definiranim na razini SQL Servera



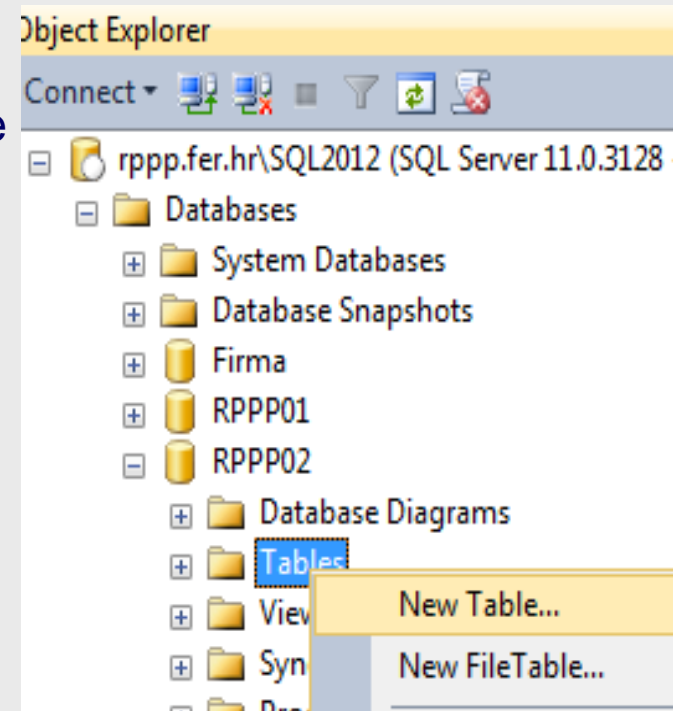
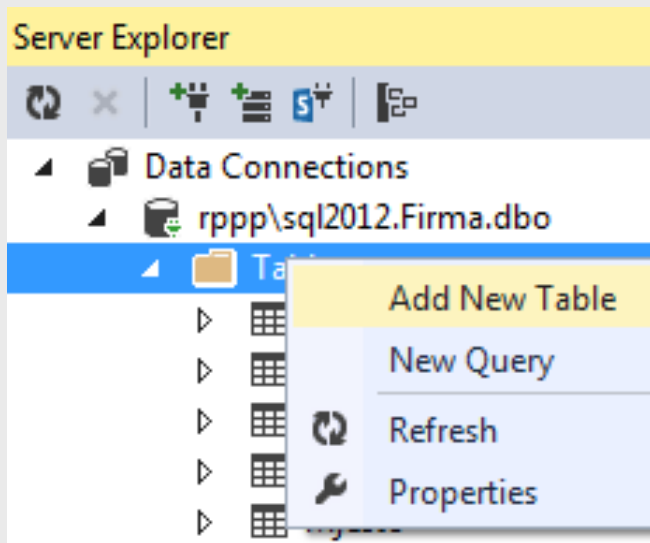
# Pristup bazi podataka iz razvojnog okruženja

## ❑ SQL Server Management Studio

- Tables \ desni klik \ New Table - kreiranje tablice
- tablica \ desni klik – dizajn i rukovanje podacima
- New Query ili File – Open + Execute

## ❑ Razvojno okruženje Visual Studio

- Server Explorer \ Data Connections \ desni klik \ Add Connection
- desni klik \ New Query ili Add New Table



# ADO.NET

---

# ADO.NET

## ❑ ActiveX Data Objects .NET (ADO.NET) je tehnologija za rukovanje podacima unutar .NET Frameworka

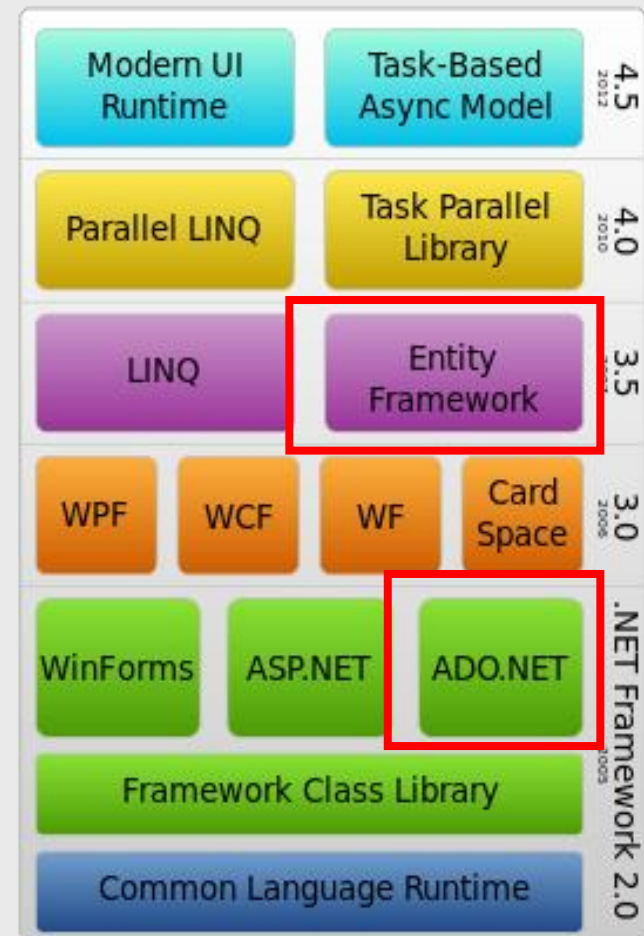
- Omogućuje pristup bazama podataka, ali i drugim spremištima podataka, za koje postoji odgovarajući opskrbljivač podacima (provider)
  - sinonimi za opskrbljivač: davatelj, pružatelj, poslužitelj

## ❑ Podrška različitim tipovima spremišta

- Strukturirani, nehijerarhijski podaci
  - Comma Separated Value (CSV) datoteke,
  - Microsoft Excel proračunske tablice, ...
- Hijerarhijski podaci
  - XML dokumenti
- Relacijske baze podataka
  - SQL Server, Oracle, MS Access, ...

## ❑ Entity Framework za objektno-relacijsko preslikavanje

- Izvorno dio .NET-a, kasnije *Open Source* paket





# Opskrbljivači (davatelji) podataka

- ❑ Postoje dvije osnovne kategorije davatelja prilagođene različitim tehnologijama i smještene u odgovarajuće prostore imena
- ❑ ***System.Data.SqlClient***
  - optimiran za rad s MS SQL Server-om
  - Razredi: *SqlCommand*, *SqlConnection*, *SqlDataReader*, *SqlDataAdapter*
- ❑ ***System.Data.OleDb***
  - generički davatelj za rad s bilo kojim OLE Database (OLE DB) izvorom
    - npr: MS JET, SQL OLE DB
  - Razredi: *OleDbCommand*, *OleDbConnection*, *OleDbDataReader*, *OleDbDataAdapter*
- ❑ Navedeni razredi implementiraju zajednička sučelja pa imaju članove jednakih naziva
  - čime se postiže neovisnost aplikacije o fizičkom smještaju podataka
  - Popis dodatnih davatelja podataka: <http://msdn.microsoft.com/en-us/data/dd363565.aspx>

# Davatelj podataka *.NET Data Provider*

## ☐ Connection

- Priključak (veza) s izvorom podataka

## ☐ Command

- naredba nad izvorom podataka

## ☐ DataReader

- Rezultat upita nad podacima (forward-only, read-only connected result set)

## ☐ ParameterCollection

- Parametri Command objekta

## ☐ Parameter

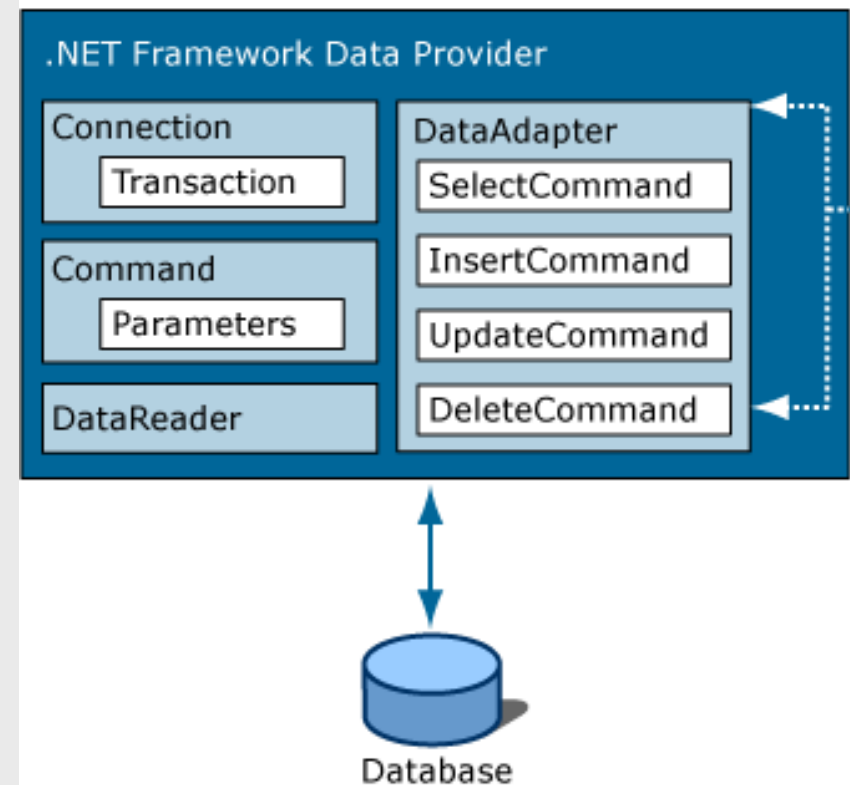
- Parametar parametrizirane SQL naredbe ili pohranjene procedure

## ☐ Transaction

- Nedjeljiva grupa naredbi nad podacima

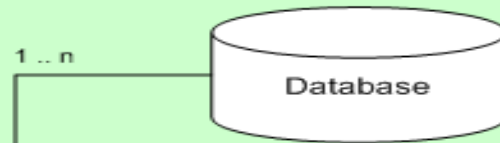
## ☐ DataAdapter

- Most između podataka na izvoru i kopije u memoriji (DataSet i DataTable)

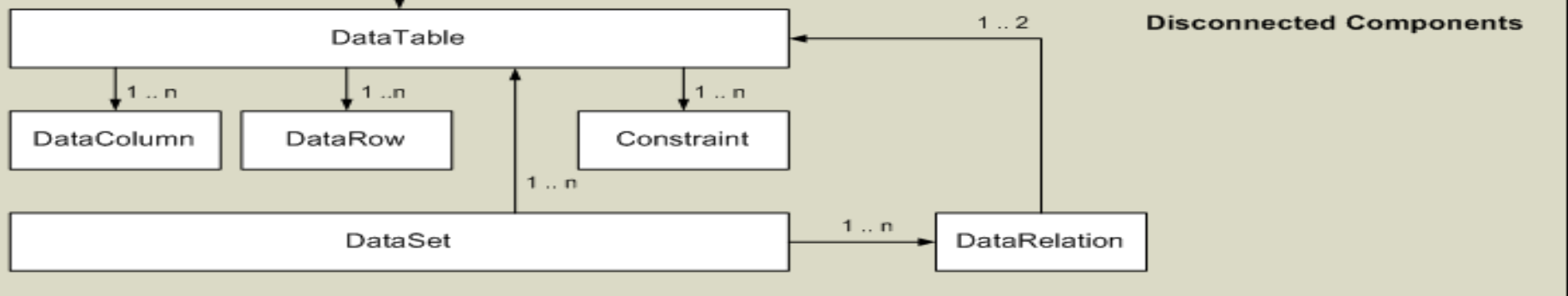
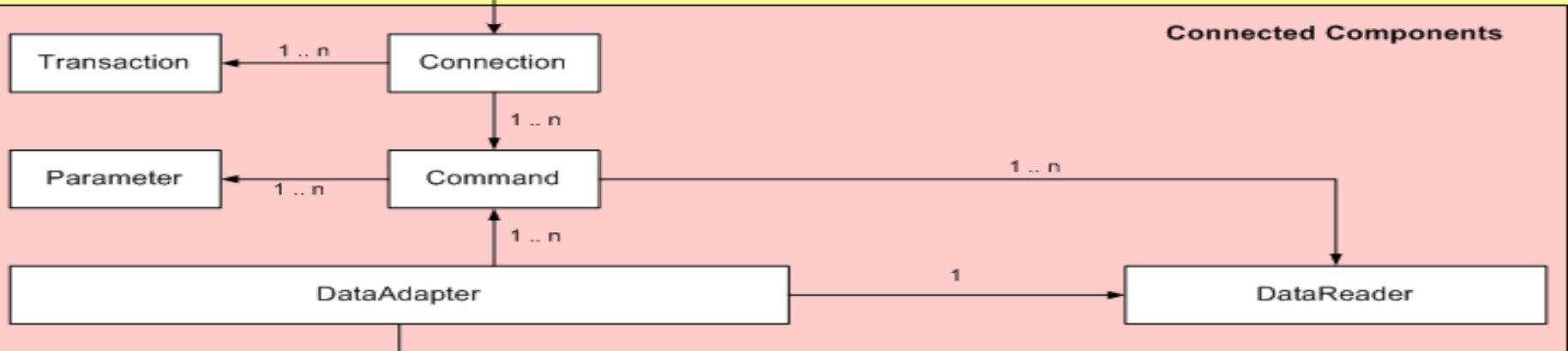


# Odnosi među elementima ADO.NET-a

## Physical Database Layer



## ADO.NET Layer

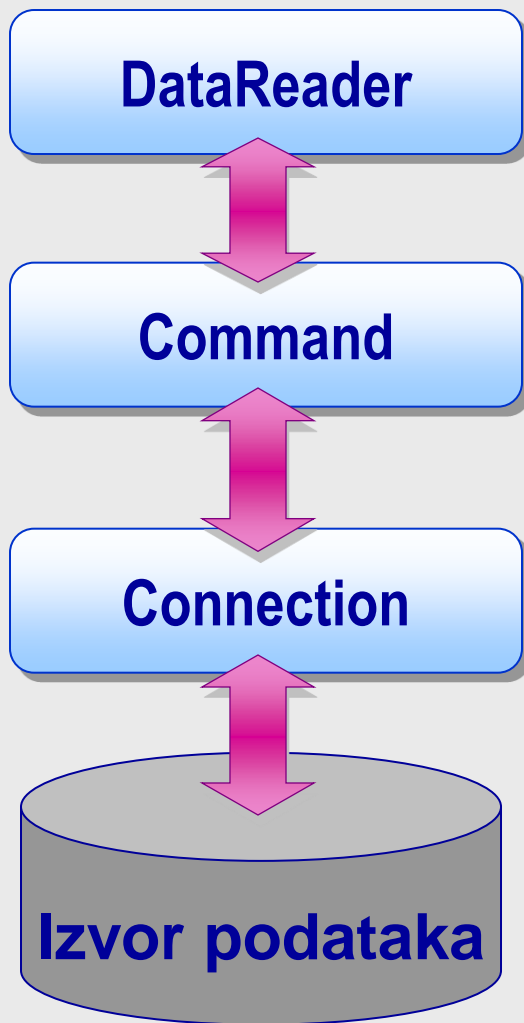


## Application Layer

# Izravna obrada podataka

---

# Izravna obrada podataka na poslužitelju



## ☐ Resursi na poslužitelju

1. Otvori priključak
2. Izvrši naredbu
3. Obradi podatke u čitaču
4. Zatvori čitač
5. Zatvori priključak

# Priključak na bazu podataka

## ❑ Priključak, veza (Connection)

- otvara i zatvara vezu s fizičkim izvorom podataka
- omogućuje transakcije

## ❑ Sučelje *System.Data.IDbConnection*

## ❑ Apstraktni razred *System.Data.DbConnection*

## ❑ Implementacije: *OleDbConnection*, *SqlConnection*, ...

## ❑ Važnija svojstva

- `ConnectionString` – string koji se sastoji od parova postavki oblika naziv=vrijednost odvojenih točka-zarezom
  - jedino promjenjivo svojstvo
- `State` – oznaka stanja priključka
  - `enum ConnectionState { Broken, Closed, Connecting, Executing, Fetching, Open }`

## ❑ Važniji postupci

- `Open` – prikapčanje na izvor podataka
- `Close` - otkapčanje s izvora podataka

# Elementi svojstva `ConnectionString`

`AttachDBFilename`

Koristi se kada se želi pristupiti bazi podataka koja nije registrirana u SUBP (npr. .MDF koji nije vezan na SQL Server). Uobičajeno se koristi parametar `Initial Catalog`.

`ConnectTimeout`

Vrijednost svojstva `Timeout` – čekanje do otvaranja ili iznimke

`Data Source`

Naziv samostojne baze podataka (npr. MS Access .MDB), naziv poslužitelja ili mrežna adresa poslužitelja baze podataka. Na lokalnom računalu može se koristiti naziv `local` ili `".."`.

`Initial Catalog /  
Database`

Naziv baze podataka.

`Integrated Security`

Postavlja se na `false` (default), `true` ili `SSPI` (Security Service Provider Interface – standardizirano sučelje za sigurnost distribuiranih aplikacija). Kada se postavi na `SSPI`, .NET se povezuje koristeći sustav zaštite OS Windows.

`Persist Security Info`

`True` ili `false` (default). Kad je postavljen na `false`, sigurnosno osjetljive postavke (npr. lozinka) se automatski uklanjaju iz `ConnectionStringa` nakon što je priključak otvoren.

`User ID / UID`

Identifikator korisnika (korisničko ime) u bazi podataka.

`Password/PWD`

Lozinka za korisničko ime.

# Primjeri postavki priključka na bazu

## ❑ Primjer: ADO\Upitnik – NekeKonekcije.txt

### ■ OleDb ConnectionString za MS Access

- `Provider=Microsoft.ACE.OLEDB.12.0;Data Source=c:\Projects\Firma.mdb`

### ■ OleDb ConnectionString za Excel

- `Provider=Microsoft.ACE.OLEDB.12.0;Data Source=c:\Projects\podaci.xls;Extended Properties='Excel 8.0;HDR=Yes '`

### ■ System.Data.SqlClient.SqlConnection

- `Data Source=.\SQL2008;Initial Catalog=Firma;Integrated Security=True`
- `Data Source=rppp.fer.hr\SQL2012,3000;Initial Catalog=Firma;User Id=rppp;Password=šifra`



# Postavke priključka na bazu podatka

- ❑ Izbjegavati pisanje postavki priključka unutar koda
  - Promjena postavki bi zahtijevala ponovnu izgradnju programa (nova verzija?)
- ❑ Postavke staviti u konfiguracijsku datoteku (app.config)

```
<configuration>
  <connectionStrings>
    <add name="Firma" connectionString="Data
      Source=rppp.fer.hr\SQL2012,3000;Initial Catalog=Firma;
      User Id=rppp;Password=šifra;" />
  </connectionStrings>
</configuration>
```

- ❑ Dohvatljivo iz koda pomoću razreda *ConfigurationManager*

```
string connString =
  ConfigurationManager.ConnectionStrings["Firma"].ConnectionString;
```

# Povezivanje s bazom podataka i postavljanje upita

## ❑ Primjer: ADO\UsingDataReader\PopisArtikala

```
string connString = "Data Source=rppp.fer.hr\\SQL2012,3000;  
    Initial Catalog=Firma;User Id=rppp;Password=šifra";  
SqlConnection conn = new SqlConnection(connString);  
SqlCommand command = new SqlCommand();  
command.CommandText = "SELECT TOP 3 * FROM Artikl";  
command.Connection = conn;  
conn.Open();  
SqlDataReader reader = command.ExecuteReader();  
while (reader.Read())  
{  
    object NazivArtikla = reader["NazArtikla"];  
    ...  
}  
reader.Close();  
conn.Close();
```

# Sučelje *IDbCommand*

- ❑ **Reprezentira SQL naredbe koje se obavljaju nad izvorom podataka**
  - upit može biti SQL naredba ili pohranjena procedura
- ❑ **Implementacija u .NET pružateljima koji pristupaju relacijskim BP**
  - *OleDbCommand* i *SqlCommand*
- ❑ **Važnija svojstva**
  - `Connection`: priključak na izvor podataka
  - `CommandText`: SQL naredba, ime pohranjene procedure ili ime tablice
  - `CommandType`: tumačenje teksta naredbe, standardno `Text`
    - `enum CommandType { Text, StoredProcedure, TableDirect }`
- ❑ **Važniji postupci**
  - `ExecuteReader` – izvršava naredbu i vraća `DataReader`
  - `ExecuteNonQuery` – izvršava naredbu koja vraća broj obrađenih zapisa, npr. neka od naredbi `UPDATE`, `DELETE` ili `INSERT`.
  - `ExecuteScalar` – izvršava naredbu koja vraća jednu vrijednost, npr. rezultat agregatne funkcije

# Ostali članovi *IDbCommand*

## ❑ Svojstva

- `CommandTimeout`: broj sekundi čekanja na izvršenje (standardno 30s)
- `Parameters` – kolekcija parametara (argumenata) naredbe
  - `Parameter` - parametar parametrizirane SQL naredbe ili pohranjene procedure, sa svojstvima: `DbType`, `IsNullable`, `OleDbType`, `ParameterName`, `Precision`, `Scale`, `Size`, `SourceColumn`
- `Transaction` – transakcija koje je naredba dio (o transakcijama kasnije)
- `UpdatedRowSource` – određuje način ažuriranja izvora podataka, kad se naredba koristi sa skupom podataka i prilagodnikom podataka

## ❑ Postupci

- `Cancel` – pokušaj prekida naredbe koja se izvršava
  - da bi prekid bio moguć, naredba mora biti pokrenuta na drugoj niti
  - u protivnom će kod biti blokiran, jer se naredbe izvršavaju sinkrono
- `CreateParameter` – kreira novi `Parameter` objekt, koji se dodaje u kolekciju `Command.Parameters`
  - primjer: `public DbParameter CreateParameter();`
- `Prepare` – postupak se koristi za pripremu (prekompilaciju) naredbe na izvoru podataka, s namjerom poboljšanja brzine njenog izvođenja

# Sučelje *IDataReader*

## ❑ Razredi *OleDbDataReader* i *SqlDataReader* – implementiraju `System.Data.IDataReader`

- Rezultat upita nad podacima (forward-only, read-only connected result set).

## ❑ Važnija svojstva

- `FieldCount` - broj stupaca u rezultatu upita
- `HasRows` – indikator da `DataReader` objekt sadrži zapise
- `IsClosed` – indikator da je `DataReader` objekt zatvoren
- `Item` – vrijednost stupca u izvornom obliku
  - `public virtual object this[int] {get;}`
  - `public virtual object this[string] {get;}`
- `RecordsAffected` - broj zapisa obrađenih naredbom koja mijenja podatke
  - 0 ako nije obrađen ni jedan zapis, -1 za `SELECT` naredbu

## ❑ Važniji postupci

- `Read` – čita sljedeći zapis u `DataReader`
  - vraća `true` ako postoji još zapisa
- `Close` – zatvara `DataReader` objekt (ne nužno i priključak s kojeg čita)

# Ostali članovi *IDataReader*

## ❑ Postupci

- **GetName** – vraća naziv za zadani redni broj stupca
- **GetOrdinal** – vraća redni broj za zadano ime stupca
- **GetValue** – dohvaća vrijednost zadanog stupca za aktualni redak
  - `public virtual object GetValue( int ordinal );`
- **GetValues** – dohvaća aktualni redak kao polje objekata
  - `public virtual int GetValues( object[] values );`
- **GetType** – dohvaća vrijednost zadanog stupca u određenom tipu, na primjer `GetChar` ili npr.
  - `DataReader rdrArtikl = cmdArtikl.ExecuteReader();`
  - `int sifraArtikla = readerArtikl.GetInt32(0);`
- **GetSchemaTable** – dobavlja `DataTable` objekt s opisom podataka
- **NextResult** – pomiče se na sljedeći rezultirajući skup, za naredbe koje vraćaju više skupova
  - vraća `true` ako postoji još rezultata

# Zatvaranje priključka

- ❑ Svaku otvorenu vezu prema bazi podataka treba zatvoriti!
- ❑ Što ako se dogodi iznimka u prethodnom primjeru?
  - `Conn.Close()` nije izvršen – veza ostaje otvorena i ne može se ponovo iskoristiti
    - Staviti `Conn.Close()` unutar *finally* blocka
      - 📁 ADO\UsingDataReader\PopisArtikalaTryCatch
  - Priključak implementira sučelje *IDisposable* → *Dispose* je (u ovom slučaju) ekvivalentan *Close* → koristiti *using*
    - 📁 ADO\UsingDataReader\PopisArtikalaUsing

```
using (SqlConnection conn = new SqlConnection(connString)) {  
    using (SqlCommand command = new SqlCommand()) {  
        ...  
        using (SqlDataReader reader = command.ExecuteReader()) {  
            ...  
        }  
    }  
}
```

- ❑ Napomena: rješenje s *using* nije uvijek moguće – npr. ako će povezivanje/čitanje podataka naknadno nastupiti
  - Automatsko zatvaranje priključka kad se zatvori čitač  
`command.ExecuteReader(System.Data.CommandBehavior.CloseConnection)`

# Povezivanje s različitim tipovima baza podataka i postavljanje upita

## ❑ Primjer: ADO\Upitnik

Postavke priključka

☐ MS SQL Server priključak  
☐ OLEDB Excel priključak  
☒ OLEDB Access priključak

ConnectionString:  
Provider=Microsoft.ACE.OLEDB.12.0;Data Source=..\..\Fima.mdb;

Otvori Obavi NonQuery Scalar Zatvori

SQL upit:  
SELECT \* FROM Artikl

Rezultat upita:

SifArtikla	NazArtikla	JedMjere	CijArtikla	ZastUsluga
1234	RAM, 512 MB, ...	kom	407,00	False
2184	SODIMM PC-13...	kom	263,00	False
2937	Knjiga "PC Škol...	kom	100,00	False
3194	Knjiga "PC Škol...	kom	120,00	False
3688	Knjiga "PC Škol...	kom	69,00	False
4243	Torba za notebo...	kom	283,00	False
4409	SODIMM SSSR ...	kom	366,00	False



# Uspostavljanje priključka na izvor podataka

- ❑ Konkretni priključak može se promatrati kroz sučelje *IDbConnection* ili apstraktni razred *DbConnection*
- ❑ Primjer:  ADO\Upitnik

```
string connString = "";

IDbConnection connection; // sučelje
//može i DbConnection connection; //apstraktni razred
...
if ((radioButtonSQLCON.Checked))
{
    connection = new SqlConnection(connString);
}
else
{
    connection = new OleDbConnection(connString);
}
connection.Open(); // višeobličje
```

# Primjer izvođenja naredbe za dohvat podataka

## ❑ Primjer: ADO\Upitnik (gumb Obavi)

```
// analogno za OleDb
SqlCommand sqlCommand = new SqlCommand(textBoxUpit.Text,
                                       sqlConnection);

SqlDataReader sqlReader = sqlCommand.ExecuteReader();

IDataReader reader = sqlReader;

using(reader) {
    //s using osiguravamo da se napravi .Close() nakon bloka

    // obrada čitača
    while (reader.Read()) // isto što i sqlReader.Read()
    {
        for (int i = 0; i < reader.FieldCount; i++)
            // čini nešto s reader[i] //.ToString()
        }
    //reader.Close(); nije potrebno zbog using
}
```

# Primjer izvođenja drugih upita

## ❑ Primjer: ADO\Upitnik

```
// analogno za OleDb

SqlCommand sqlCommand = new SqlCommand(textBoxUpit.Text,
                                         sqlConnection);

IDbCommand command = sqlCommand;

// naredba koja vraća broj obrađenih zapisa
int result = command.ExecuteNonQuery();

// naredba koja vraća jednu vrijednost
object o = command.ExecuteScalar();
```



# Zadatak za vježbu

- ❑ **Doraditi primjer obradom događaja i prikazom stanja priključka**
  - *Napomena: Priključak mora biti tipa `DbConnection`, umjesto `IDbConnection`*

```
...  
connection.StateChange += new  
StateChangeEventHandler(connection_StateChange);  
  
...  
private void connection_StateChange  
    (object sender, StateChangeEventArgs e) {  
    // e.OriginalState // .ToString()  
    // e.CurrentState // .ToString()  
}
```



# Zadaci za vježbu


- ❑ U tablicu *Artikl* u bazi podataka dodati sve artikle iz neke *Excel* datoteke. Za dodane zapise
  - Provjeriti ima li jedinicu mjere iz skupa { "h", "kom", "kg", "l", "pak" } .
  - Ukoliko nema, ažurirati jedinicu mjere vrijednošću "---".
  - Ažuriranje provesti kreiranjem odgovarajuće SQL naredbe za svaki zapis koji treba mijenjati.
- ❑ Prebaciti pojedini redak čitača u polje objekata.
  - Podatke iz polja objekata prikazati u *ListBox* kontroli koristeći naredbu *foreach* prema uzoru indeksiranja iz sljedećeg primjera.

```
while (reader.Read())
{
    Object [] cols = new Object[10] ;
    reader.GetValues( cols );

    Console.WriteLine( cols[0].ToString() + " | " + cols[1] );
}
```

# DbProviderFactory / DbProviderFactories


## ❑ Omogućava stvaranje priključaka i naredbi bez navođenja konkretnih implementacija

- Statički postupak `GetFactory` u razredu `DbProviderFactories` vraća instancu razreda `DbProviderFactory`
- Postupci `CreateConnection`, `CreateCommand`, ... kao rezultat vraćaju instance konkretnih implementacija, ali promatrane kroz odgovarajuće apstraktne razrede
- Primjer:  ADO\Upitnik

```
string factoryName = "System.Data.SqlClient";  
DbProviderFactory factory =  
    DbProviderFactories.GetFactory(factoryName);  
using (DbConnection conn = factory.CreateConnection())  
{  
    conn.ConnectionString = ...  
    using (DbCommand command = factory.CreateCommand()) {  
        command.Connection = conn;  
        ...  
    }  
}
```

# Parametrizirani upiti

## ❑ Dijelovi upita s parametrima oblika @NazivParametra

- Olakšava pisanje upita
- Brže izvršavanje u slučaju višestrukih izvršavanja
- Zaštita od SQL injection napada
- Primjer:  ADO\DataReaderParamProc\Program.cs – ParamUpiti
- Instancirati implementaciju apstraktnog razreda DbParameter (npr. new SqlParameter) ili pozvati postupak CreateParameter na nekoj naredbi

```
command.CommandText = "SELECT TOP 3 * FROM Artikl WHERE JedMjere =  
    @JedMjere ORDER BY CijArtikla DESC;" +  
    "SELECT TOP 3 * FROM Artikl WHERE JedMjere = @JedMjere AND  
    CijArtikla > @Cijena ORDER BY CijArtikla";  
  
DbParameter param = command.CreateParameter() ;  
param.ParameterName = "JedMjere"; param.DbType = DbType.String;  
param.Value = "kom"; command.Parameters.Add(param) ;  
  
param = command.CreateParameter();  
param.ParameterName = "Cijena"; param.DbType = DbType.Decimal;  
param.Value = 100m; command.Parameters.Add(param);
```

# Svojstva parametra

- ❑ ***DbType*** – vrijednost iz enumeracije *System.Data.DbType*
  - Predstavlja tip podatka koji se prenosi parametrom.
- ❑ ***Direction*** – vrijednost iz enumeracije *System.Data.ParameterDirection*
  - Određuje da li je parametar ulazni, izlazni, ulazno-izlazni ili rezultat poziva pohranjene procedure. Ako se ne navede, pretpostavlja se da je ulazni.
- ❑ ***IsNullable*** – Određuje može li parametar imati null vrijednost
- ❑ ***ParameterName*** – Naziv parametra
- ❑ ***Size*** – Maksimalna veličina parametra u bajtovima
  - Upotrebljava se kod prijenosa tekstualnih podataka.
- ❑ ***Value*** – Vrijednost parametra
  - Vrijednost izlaznog argumenta se može dobiti i preko instance naredbe `command.Parameters["Naziv parametra"].Value`



# Upit s više skupova rezultata

- ❑ U slučaju da rezultat upita vraća više skupova rezultata, svaki sljedeći dohvaća se postupkom *NextResult* na čitaču podataka
- ❑ Primjer:  ADO\DataReaderParamProc\Program.cs – ParamUpiti

```
command.CommandText = "SELECT TOP 3 * FROM Artikl WHERE JedMjere =  
    @JedMjere ORDER BY CijArtikla DESC;" +  
    "SELECT TOP 3 * FROM Artikl WHERE JedMjere = @JedMjere AND  
    CijArtikla > @Cijena ORDER BY CijArtikla";  
  
...  
using (DbDataReader reader = command.ExecuteReader()) {  
    do {  
        while (reader.Read()) {  
            ...  
        }  
    }  
    while (reader.NextResult());  
}
```

# Pozivi pohranjenih procedura

## ❑ Primjer: ADO\DataReaderParamProc\Program.cs – Procedura

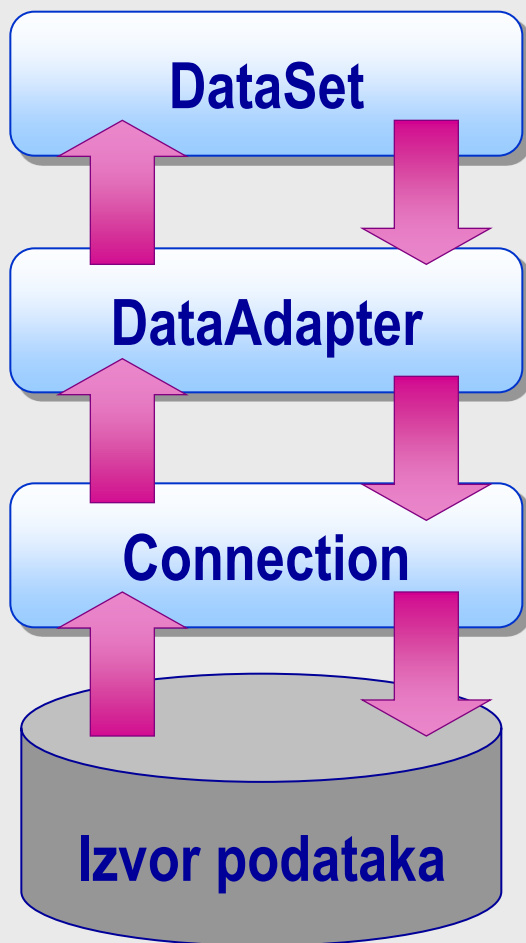
- Parametri procedure navode se kao i kod parametriziranih upita
- Svojstvo *CommandType* na naredbi potrebno je postaviti na *System.Data.CommandType.StoredProcedure*
- Ako procedura ne vraća skup podataka, koristi se postupak *ExecuteNonQuery*
  - Očekuje li se skup podataka kao rezultat koristi se *ExecuteReader*.
  - Vrijednosti izlaznih parametara mogu se dobiti tek po zatvaranju čitača

```
command.CommandText = "ap_ArtikliSkupljajOd";  
command.CommandType = System.Data.CommandType.StoredProcedure;  
...  
param = command.CreateParameter();  
param.ParameterName = "BrojJeftinijih"; param.DbType = DbType.Int32;  
param.Direction = System.Data.ParameterDirection.Output;  
command.Parameters.Add(param);  
...  
using (DbDataReader reader = command.ExecuteReader()) { ... }  
int brJef = command.Parameters["BrojJeftinijih"].Value
```

# Lokalna obrada podataka

---

# Lokalna obrada podataka



## ☐ Podaci se obrađuju lokalno

- DataSet reprezentira stvarne podatke pohranjene u memoriju

1. Otvori priključak
2. Napuni *DataSet*
3. Zatvori priključak
4. Obradi *DataSet*
5. Otvori priključak
6. Ažuriraj izvor podataka
7. Zatvori priključak

# Dinamički skup podataka (*DataSet*)

## ❑ DataSet

- skup(ovi) podataka u memoriji računala
- sadrži kolekciju `DataTable` objekata

## ❑ DataTable

- tablica podataka u memoriji računala
- `DataColumnCollection` – kolekcija atributa
- `DataRowCollection` – kolekcija zapisa
- `ConstraintCollection` – kolekcija ograničenja nad tablicom

## ❑ DataRow

- rukovanje retkom u `DataTable`

## ❑ DataColumn

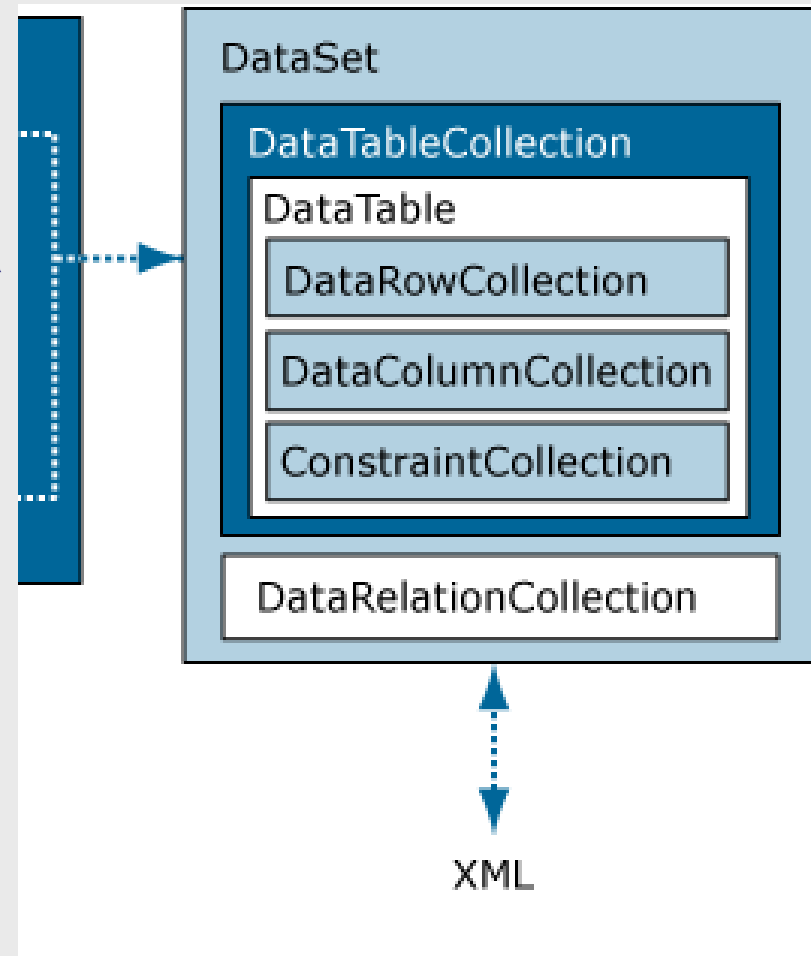
- definira stupce u `DataTable`

## ❑ DataRelationCollection

- kolekcija `DataRelation` objekata
- `DataRelation` - veza između dvije tablice (`DataTable`)

## ❑ DataViewManager

- definira poglede nad skupovima podataka



# Vrste *DataSet* skupova

## ❑ Primjeri:

- Dva načina pristupa do vrijednosti atributa `SifArtikla` prvog zapisa tablice `Artikl` u skupu `dsArtikl`

## ❑ **Typed** - instanca `System.Data.DataSet`

- sadrži shemu podataka, koju prevoditelj koristi za provjeru sintakse i kompatibilnosti tipova podataka u izrazima (*strong typing*)
- `dsArtikl.Artikl[0].SifArtikla`
- pogreška pri prevođenju ukoliko ne postoji `Artikl` ili `SifArtikla`

## ❑ **UnTyped** - razred naslijeđen iz `System.Data.DataSet`

- struktura podataka ne mora biti unaprijed poznata - program tijekom izvođenja može primiti podatke od vanjske komponente ili servisa (npr. Web servis)
- `dsArtikl.Tables["Artikl"].Rows[0].Item["SifArtikla"]`
- `dsArtikl.Tables["Artikl"].Rows[0]["SifArtikla"]`
- pogreška tek pri izvođenju kada ne postoji `Artikl` ili `SifArtikla`

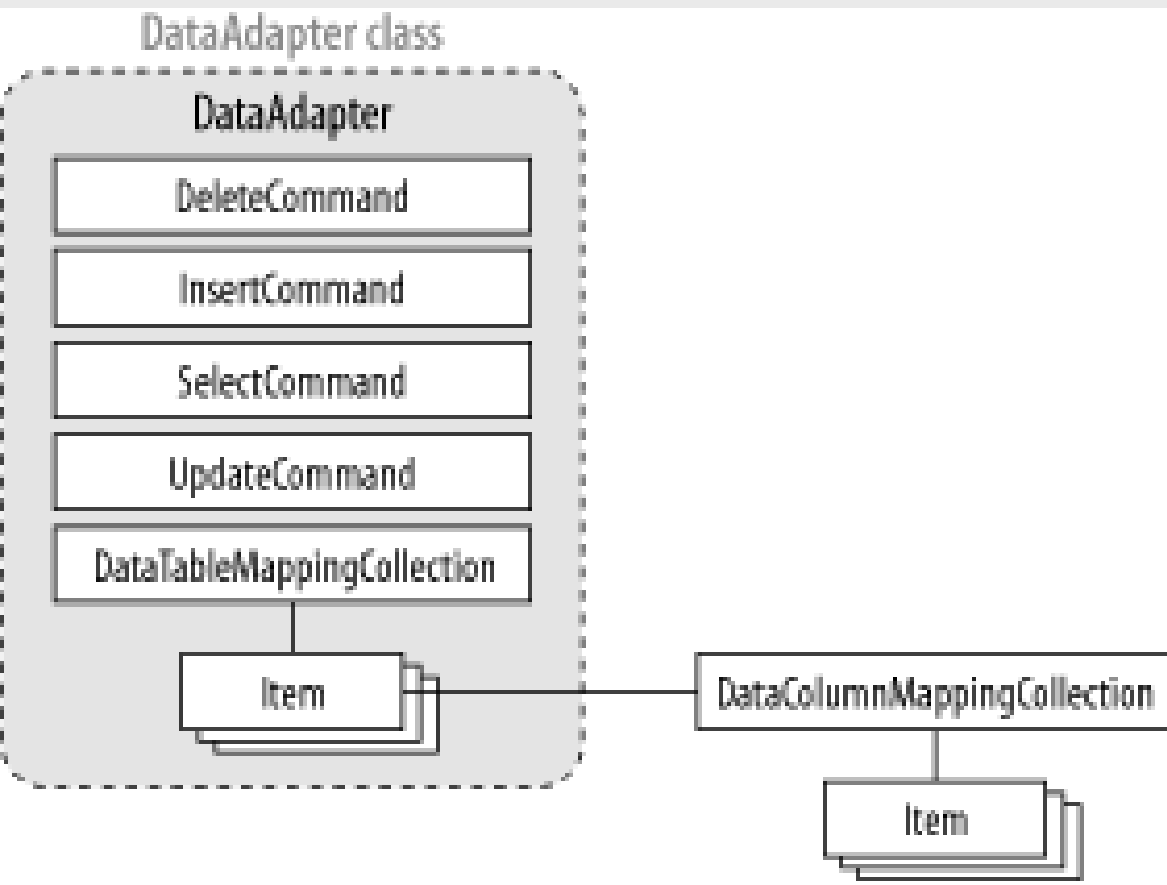
# IDataAdapter

## ❑ Prilagodnik - most između skupa podataka i izvora podataka

- SqlDataAdapter i OleDbDataAdapter nasljeđuju  
System.Data.Common.DataAdapter

## ❑ Toolbox \ Data (drag-drop na formu)

- SqlDataAdapter, SqlConnection, SqlCommand (analogno OleDb)
  - ukoliko se komponente ne vide treba ih omogućiti desnim klikom na Toolbox \ Data \ Choose Items ... a zatim označiti na popisu



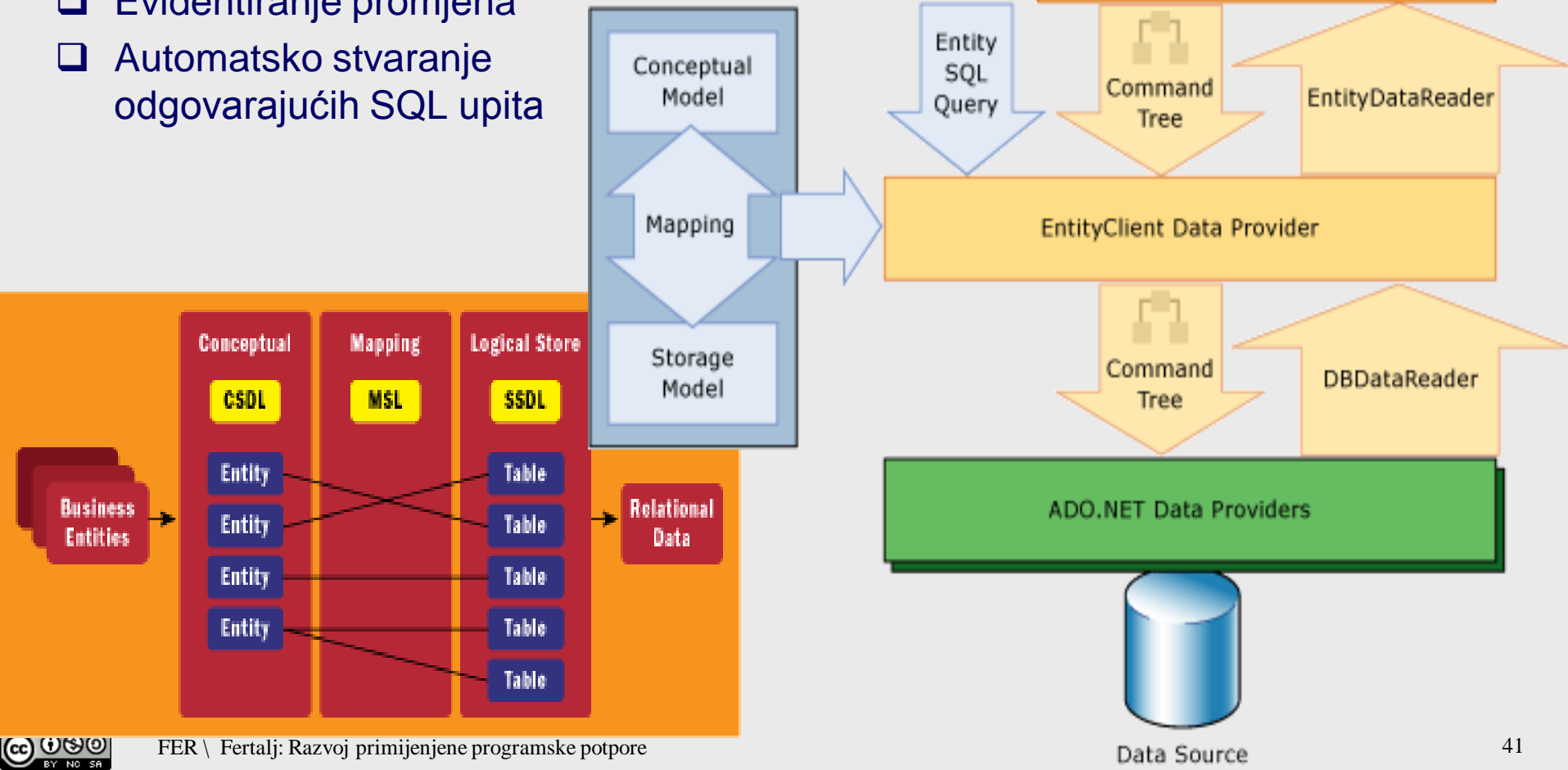
# Entity Framework

---



# Entity Framework

- ❑ Nadgradnja nad ADO.NET-om
- ❑ Rad s BP na višoj razini putem objektnog modela (preslikavanje između modela i podataka u BP)
- ❑ Evidentiranje promjena
- ❑ Automatsko stvaranje odgovarajućih SQL upita



# Načini kreiranja EF modela

## ❑ **Database First**

- Baza podataka već postoji i model nastaje reverznim inženjerstvom BP

## ❑ **Model First**

- Model se dizajnira kroz grafičko sučelje, a BP nastaje na osnovu modela.

## ❑ **Code First**

- Model opisan kroz ručno napisane razrede te nema vizualnog modela
- BP se stvara na osnovu napisanih razreda. Izgled BP određen nazivima razreda, nazivima i vrstama asocijacija između razreda te dodatnim atributima.

## ❑ **Razredi generirani opcijama *DatabaseFirst* i *ModelFirst* ovise o verziji Entity Frameworka i pridruženom generatoru koda**

- Visual Studio 2010 + EF 4: razredi izvedeni iz razreda *EntityObject*
- Visual Studio 2012,2013 + EF 5,6: T4 predlošci za „čiste” razrede
- Način generiranja razreda moguće promijeniti
  - Svojstvo modela *Code Generation Strategy* + kontekstni izbornik *Add Code Generation Item* dok je model prikazan
  - Detaljnije kroz sljedeće slajdove

# Stvaranje modela na osnovu postojeće BP (1)

❑ Add New Item

⇒ ADO.NET Entity Data Model

⇒ Generate from database

The screenshot shows the 'Add New Item' dialog in Visual Studio. The 'Installed' tab is selected, and the 'Visual C# Items' category is expanded. The 'ADO.NET Entity Data Model' item is highlighted. The 'Entity Data Model Wizard' is open, showing the 'Choose Model Contents' step. The 'Generate from database' option is selected. The 'Name' field is set to 'Firma.edmx'.

Sort by: Default

Icon	Item Name	Category
	ADO.NET Entity Data Model	Visual C# Items
	DataSet	Visual C# Items
	EF 5.x DbContext Generator	Visual C# Items
	LINQ to SQL Classes	Visual C# Items
	Local Database	Visual C# Items
	Service-based Database	Visual C# Items
	XML File	Visual C# Items
	XML Schema	Visual C# Items
	XSLT File	Visual C# Items

**Entity Data Model Wizard**

**Choose Model Contents**

What should the model contain?

Generate from database

Empty model

Name: Firma.edmx

# Stvaranje modela na osnovu postojeće BP (2)

Connection Properties

Enter information to connect to the database. You can choose a different data source and/or server.

Data source:  
Microsoft SQL Server (SqlClient)

Server name:  
rppp.fer.hr\SQL2012

Log on to the server  
☐ Use Windows Authentication  
☒ Use SQL Server Authentication  
User name: rppp  
Password:   
☐ Save my password

Connect to a database  
☒ Select or enter a database name:  
Firma  
☐ Attach a database file:  
Logical name:

Which data connection should your application use to connect to the database?

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.  
☒ Yes, include the sensitive data in the connection string.


Entity connection string:  
metadata=res://\*/Firma.csdl|res://\*/Firma.ssdl|res://\*/Firma.msl;provider=System.Data.SqlClient;provider connection string="data source=rppp.fer.hr\SQL2012;initial catalog=Firma;user id=rppp;password=\*\*\*\*\*;MultipleActiveResultSets=True;App=EntityFramework"

☒ Save entity connection settings in App.

FirmaEntities

Test Connection OK Cancel

Entity Data Model Wizard

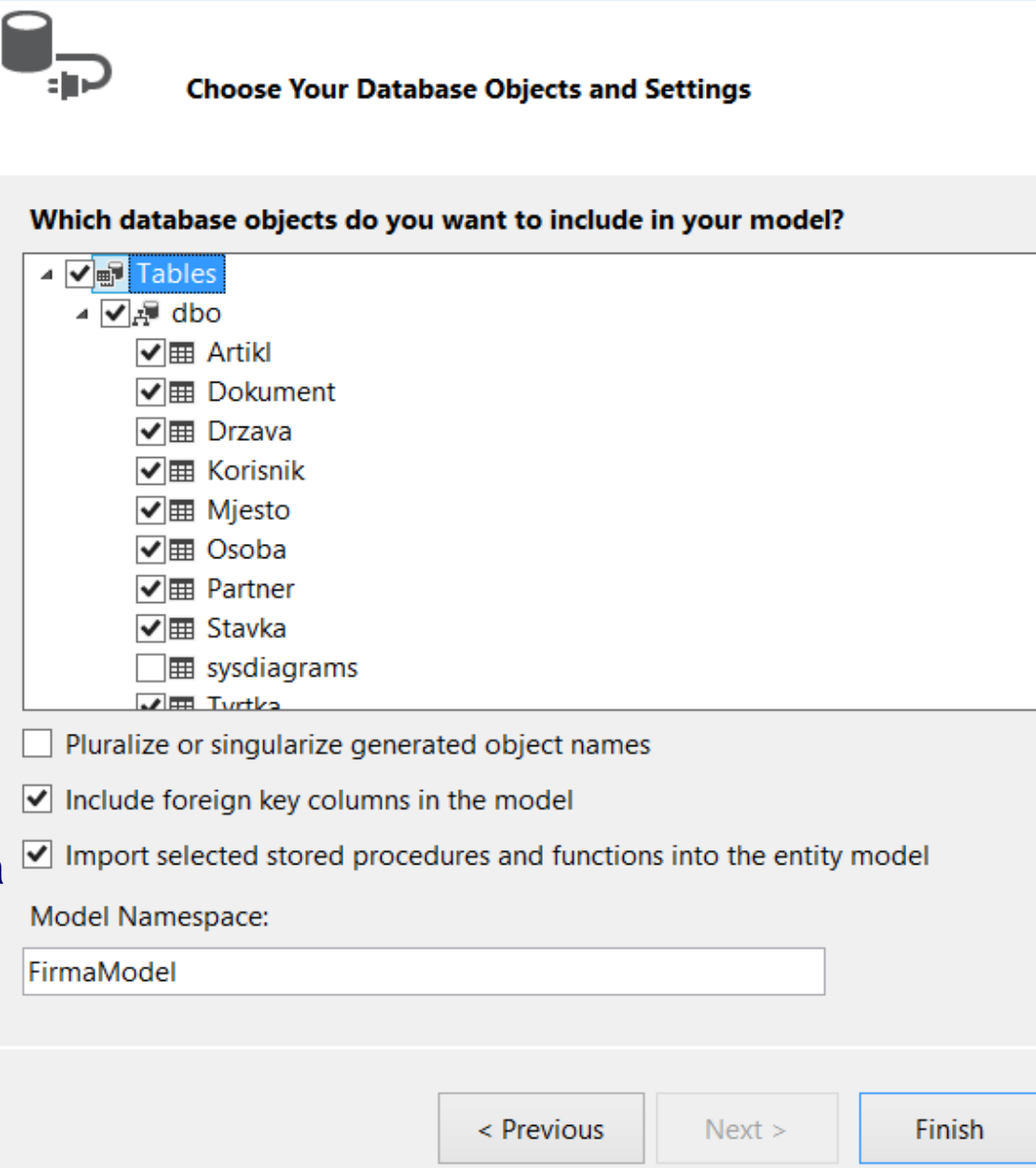
 Choose Your Version

Which version of Entity Framework do you want to use?

☒ Entity Framework 6.0  
☐ Entity Framework 5.0

# Stvaranje modela na osnovu postojeće BP (3)

- ☐ Uključivanjem stranih ključeva EF automatski stvara asocijacije između stvorenih razreda
- ☐ Moguće dodati tablice, poglede, pohranjene procedure i funkcije
- ☐ Za naš primjer stvaraju se:
  - Firma.edmx
  - Firma.Context.tt + Firma.Context.cs
  - Firma.Designer.cs
  - Firma.tt + jedna cs datoteka za svaku tablicu
  - Promjena modela ponovo stvara .cs datoteke na osnovu T4 (tt) predloška!!



**Choose Your Database Objects and Settings**

Which database objects do you want to include in your model?

- ☒ Tables
  - ☒ dbo
    - ☒ Artiki
    - ☒ Dokument
    - ☒ Drzava
    - ☒ Korisnik
    - ☒ Mjesto
    - ☒ Osoba
    - ☒ Partner
    - ☒ Stavka
    - ☐ sysdiagrams
    - ☒ Tvrka

☐ Pluralize or singularize generated object names

☒ Include foreign key columns in the model

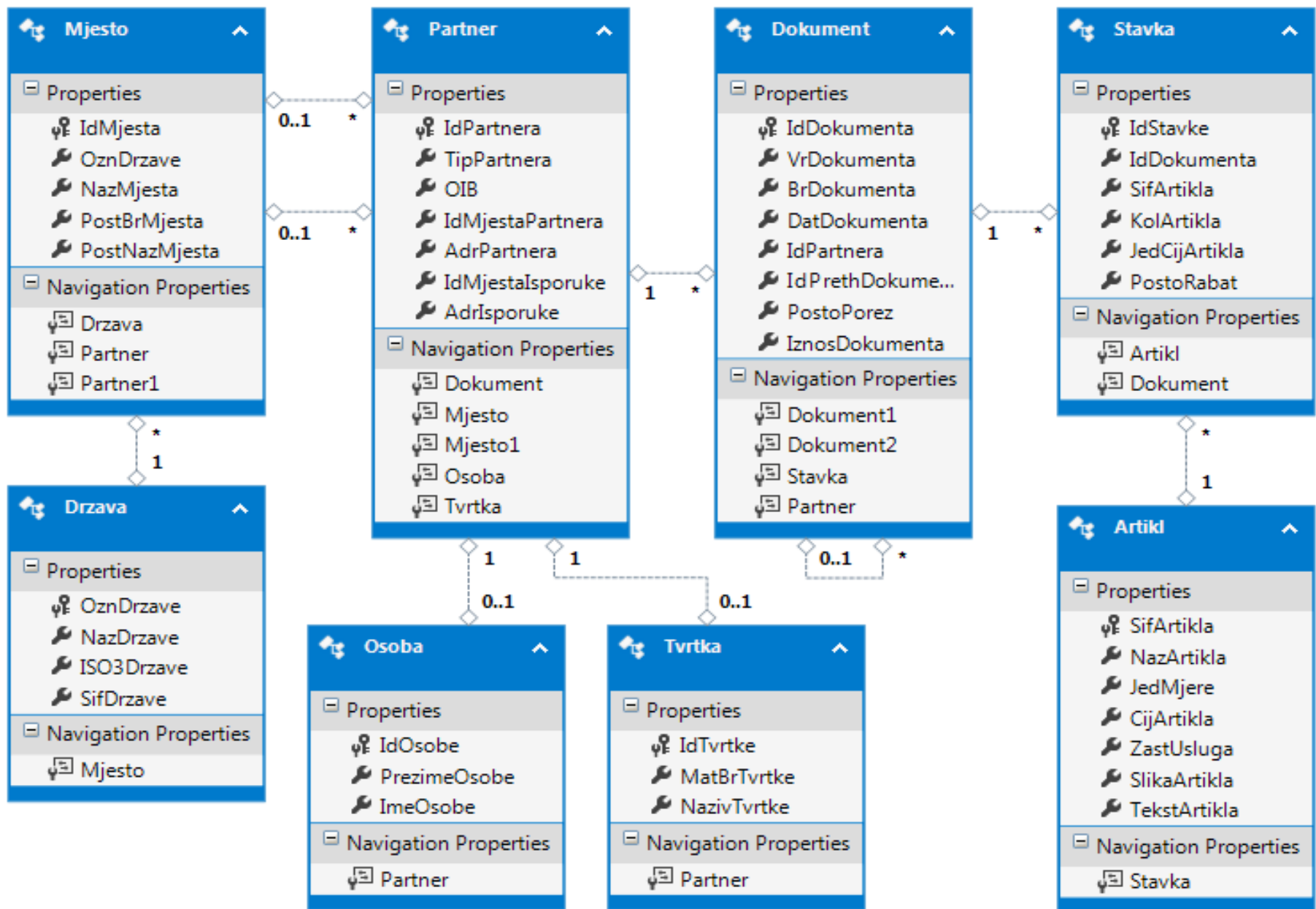
☒ Import selected stored procedures and functions into the entity model

Model Namespace:

FirmaModel

< Previous    Next >    Finish

# Početni model



# Elementi EF modela

## ❑ Primjer: ADO\EF\_Firma\Firma.edmx i pripadni Firma.designer.cs

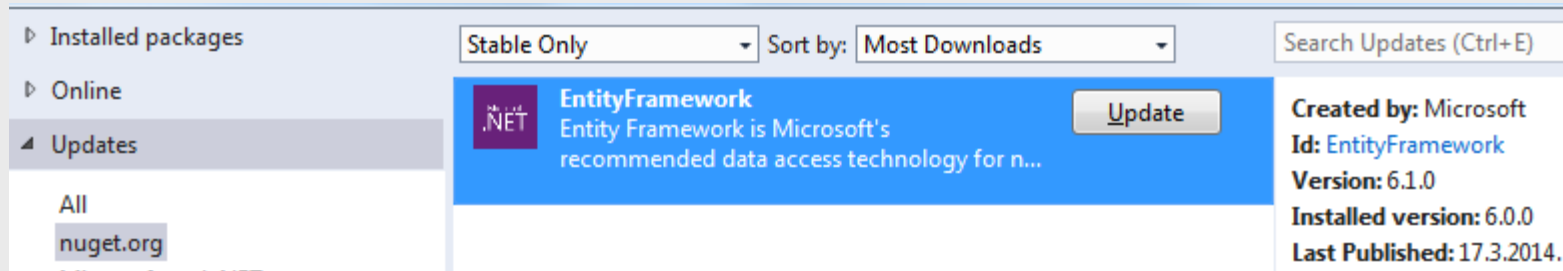
- edmx datoteka – xml datoteka s nekoliko sekcija
  - opis modela (csdl), opis fizičkog modela (ssdl), preslikavanja (c-s mapping) i postavke grafičkog prikaza
- Navodi se i u postavkama priključka

## ❑ Primjer: ADO\EF\_Firma\app.config

```
connectionString="metadata=res://*/Firma.csdl|res://*/Firma.ssdl|  
res://*/Firma.msl;provider=System.Data.SqlClient;  
provider connection string=&quot;Data  
Source=rppp.fer.hr\SQL2012,3000;Initial Catalog=Firma;  
User ID=rppp;Password=šifra; MultipleActiveResultSets=True&quot;;"
```


## ❑ Opcionalno napraviti ažuriranje verzije EF-a.

- Desni klik na projekt/rješenje *Manage NuGet Packages -> Update*




# Elementi EF modela (2)


## ❑ ***FirmaEntities*** – naslijeđen iz razreda ***DbContext***

- predstavlja kontekst za pristup bazi podataka
- podaci pohranjeni unutar konteksta u skupu entiteta tipa `DbSet<T>`, gdje je `T` tip entiteta
- Definiran u  **ADO \EF\_Firma \ Firma.edmx \Firma.Context.tt \ Firma.Context.cs**

## ❑ **Svaki entitet predstavljen parcijalnim razredom**

- Asocijacije kao virtualna svojstva (`ICollection<T>` za agregacije)
- Interno se stvara *proxy* koji pruža vlastitu implementaciju virtualnih svojstava
- U strukturi projekta unutar  **ADO \EF\_Firma \ Firma.edmx \ Firma.tt**

## ❑ **“Korisnički” definiran dio parcijalnih klasa smješta se unutar projekta po volji**

- U našem primjeru unutar fizičke mape  **ADO\EF\_Firma\Partial**
- Parcijalne klase moraju biti definirane unutar istog prostora imena (npr. namespace `EF_Firma`)



# Podešavanje neodređenih veza

## ❑ Mjesto i Mjesto1 su paralelne veze

- Postoji neodređenost u nazivu je se može sa sigurnošću reći koje je koje
- Promjene naziva asocijacija:
  - Mjesto => MjestoIspruke i
  - Mjesto1 => MjestoSjedista















## ❑ Promjena ostalih naziva asocijacija (zbog jednostavnosti u primjerima koji slijede)

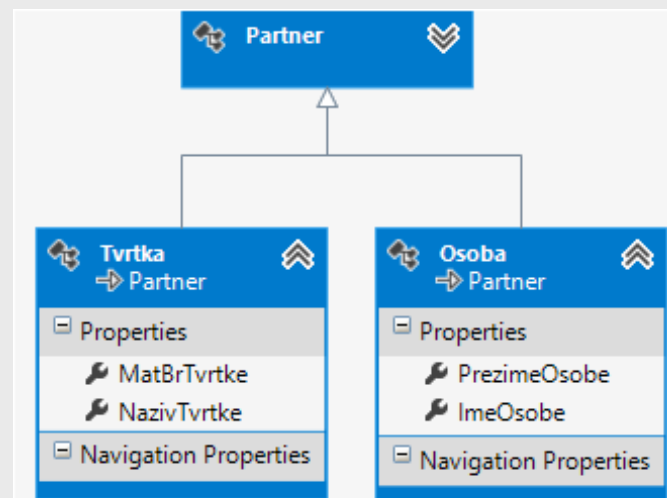
- Dokument: Dokument2 => PrethodniDokument i Stavka => Stavke
- Drzava: Mjesto => Mjesta

# Podešavanje specijalizacije

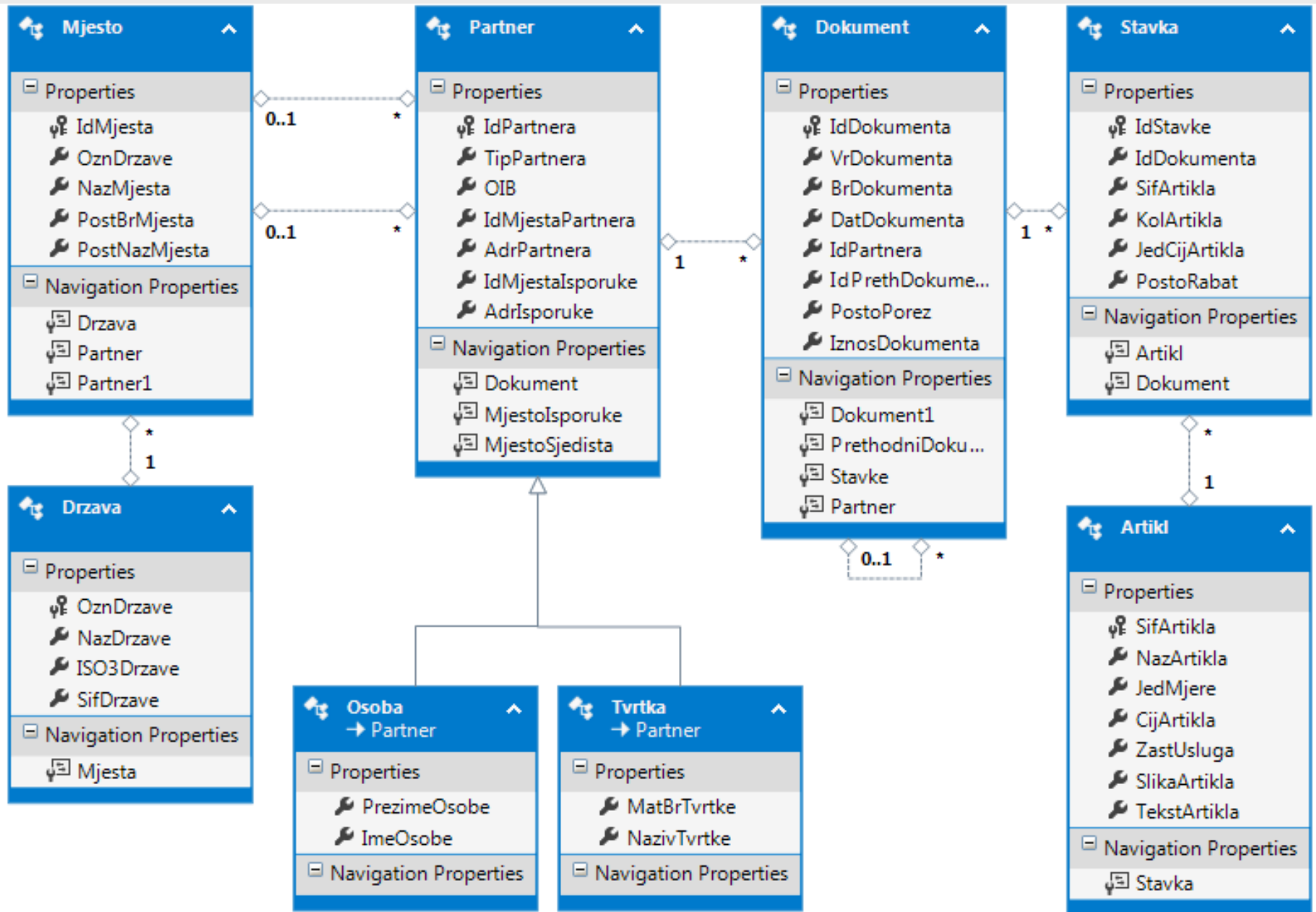
- ❑ **Partnera treba označiti kao apstraktni razred**
  - da ga se ne može nezavisno instancirati
- ❑ **Osoba i Tvrtka nasljeđuju Partnera**
  - Izbrisati veze Partnera-Osoba i Partner-Tvrtka
  - Svojstvo BaseType postavljeno na Partner
  - Obrisati IdOsobe iz Osobe i IdTvrke iz Tvrtke
  - Podesiti preslikavanje (desni klik na entitet \ Table Mappings) između IdTvrke i IdPartnera (odnosno između IdOsobe i IdPartnera)

Properties	
<b>FirmaModel.Partner</b> EntityType	
Abstract	True
<b>FirmaModel.Tvrtka</b> EntityType	
Abstract	False
Access	Public
Base Type	<b>Partner</b>
Documentation	
Entity Set Name	<b>Partner</b>
Fill Color	0; 12
Name	<b>Tvrtka</b>

Mapping Details - Tvrtka			
	Column	Oper...	Value / Property
	└ <b>Tables</b>		
	└  Maps to Tvrtka		
	└  <Add a Condition>		
	└  Column Mappings		
	 IdTvrtke : int		
 MatBrTvrtke : nvarchar			MatBrTvrtke : String
 NazivTvrtke : nvarchar			NazivTvrtke : String



# Ciljani ogledni EF model



# Važnija svojstva i postupci razreda *DbContext* i *DbSet*

## □ *DbContext*

- *SaveChanges[Async]* – spremanje promjena u bazi podataka
- *Database* – svojstvo koje omogućava direktni rad s BP (npr. kreiranje i brisanje BP, izvršavanje vlastitih SQL upita i procedura)
- *ChangeTracker* – pristup do razreda koji prati promjene na objektima u kontekstu
- *Set* i *Set<T>* - vraćaju *DbSet* za konkretni tip entiteta (Koristi se ako se želi napisati općeniti postupak, inače je svaki entitet već sadržan u kontekstu kao svojstvo)
- *Entry* i *Entry<T>* - služi za dohvat informacije o nekom entitetu u kontekstu i promjenu njegovog stanja (npr. otkazivanje promjena)

## □ *DbSet<T>*

- *Add* – dodavanje objekta u skup
- *Remove* – označavanje objekta za brisanje
- *Local* – kolekcija svih trenutno učitanih podataka (koristi se za povezivanje na forme)
- *Find [Async]* - Dohvat objekta unutar konteksta na osnovu primarnog ključa
- *AsNoTracking* – Dohvat podataka za koje se ne evidentiraju promjene

# Dodavanje novog zapisa

❑ **Primjer:**  **ADO\EF\_Firma\MainForm.cs - dodajPromijeniObrisiToolStripMenuItem\_Click**

- Stvoriti novi objekt konstruktorom (1) te ga dodati u kolekciju
- ili
- Stvoriti objekt statičkim postupkom na DbSetu (2) te ga dodati u kolekciju
- Konačno, pohraniti promjene u kontekstu (jednom za sve promjene)

```
using (FirmaEntities context = new FirmaEntities()) {  
    Artikl artikl = new Artikl() { // (1)  
        SifArtikla = 12345678, CijArtikla = 10m,  
        JedMjere = "kom", NazArtikla = "Burek sa sirom"  
    };  
    context.Artikl.Add(artikl);  
  
    artikl = context.Artikl.Create(); // (2)  
    ... //punjenje objekta  
    context.Set<Artikl>().Add(artikl);  
  
    await context.SaveChangesAsync; // pohrani sve promjene
```

# Ažuriranje postojećeg zapisa

❑ **Primjer:**  **ADO\EF\_Firma\MainForm.cs**  
- **dodajPromijeniObrisiToolStripMenuItem\_Click**

■ Dohvatiti entitet

- korištenjem postupka Find ili FindAsync na DbSetu – traži zapis na osnovu vrijednosti primarnog ključa
  - Pretražuje unutar već učitano konteksta, a ako ga ne pronađe obavlja se upit na bazu. Vraća *null* ako traženi zapis ne postoji
- Ili postavljanjem Linq upita

■ Promijeniti željena svojstva

■ Pohraniti promjene u kontekstu

```
using (FirmaEntities context = new FirmaEntities())  
{  
    Artikl artikl = await context.Artikl.FindAsync(12345678) ;  
    artikl.CijArtikla = 11m;  
    await context.SaveChangesAsync() ;  
}
```

# Brisanje zapisa

- ❑ **Primjer:**  **ADO\EF\_Firma>MainForm.cs**
  - **dodajPromijeniObrisiToolStripMenuItem\_Click**
    - Dohvatiti entitet
    - Izbaciti ga iz konkretnog *DbSeta* ili označiti ga za brisanje pomoću *context.Entry*
    - Pohraniti promjene u kontekstu

```
using (FirmaEntities context = new FirmaEntities())
{
    Artikl artikl = await context.FindAsync(12345678);
    context.Artikl.Remove(artikl);
    //ili context.Entry(artikl).State = EntityState.Deleted;
    await context.SaveChangesAsync();
    ...
}
```

# Upiti nad EF modelom

## ❑ LINQ To Entities

- Where, OrderBy, OrderByDescending, ThenBy, First, Skip, Take, Select, ...
- Davatelj usluge pretvara Linq upit u SQL upit
  - Nije uvijek moguće sve pretvoriti u SQL upit
- Upit se izvršava u trenutku dohvata prvog podataka ili eksplicitnim pozivom postupka *Load* (*LoadAsync*)
- Moguće ulančavanje upita (rezultat upita najčešće IQueryable<T>)
- Podaci iz vezane tablice se učitavaju pri svakom dohvatu ili eksplicitno korištenjem postupka *Include* (kreira *join* upit u sql-u)

## ❑ ADO\EF\_Firma\MainForm.cs - dohvatiNajskupljeToolStripMenuItem\_Click

- Primjer upita za dohvat prvih n najskupljih artikala

```
var upit = context.Artikl.  
    Include(a => a.Stavka).AsNoTracking().  
    OrderByDescending(a => a.CijArtikla).  
    Take(broj);  
foreach (Artikl artikl in upit) { ... }
```



# Pohranjene procedure i EF model

## ❑ Desni klik na modelu \ *Update Model From Database* i odabir procedure

- Generira se postupak istoimenog imena u kontekstu i novi tip podatka za povratnu vrijednost

## ❑ Rezultat je kolekcija tipa *ObjectResult<T>* gdje je T oblika *naziv procedure\_Result*

- Implementira *IEnumerable<T>*
- može se koristiti unutar *foreach* petlje
- Bez vraćanja unatrag i ponavljanja

## ❑ Definiranje rezultata kao postojećeg tipa

- Postavke procedure (desni klik na modelu \ Model Browser \ Function Imports)
- npr. tip procedure *ap\_ArtikliSkupljiOd* postavimo na *Artikl*, umjesto generiranog *ap\_ArtikliSkupljiOd\_Result*

Edit Function Import

Function Import Name:

☐ Function Import is composable

Stored Procedure / Function Name:

Returns a Collection Of

☐ None

☐ Scalars:

☐ Complex:

☒ Entities:

# Poziv pohranjene procedure

## ❏ Primjer: ADO\EF\_Firma\MainForm.cs - btnProc\_Click

- Povratni parametri se dohvaćaju korištenjem razreda *ObjectParameter*
- Parametri koji su isključivo ulazni mogu se predati navođenjem vrijednosti

```
using (FirmaEntities context = new FirmaEntities())
{
    ObjectParameter brojSkupljih = new
        ObjectParameter("BrojSkupljih", typeof(int));
    ObjectParameter brojJefitnijih = new
        ObjectParameter("BrojJeftinijih", typeof(int));
    foreach (Artikl artikl in context.ap_ArtikliSkupljiod(20000m,
        brojSkupljih, brojJefitnijih))
    {
        sb.AppendLine(artikl.NazArtikla);
    }
    sb.AppendLine("Broj skupljih: " + brojSkupljih.Value);
    ...
}
```

# Reference

## ❑ Overview of ADO.NET

- [http://msdn.microsoft.com/en-us/library/h43ks021\(v=VS.100\).aspx](http://msdn.microsoft.com/en-us/library/h43ks021(v=VS.100).aspx)


## ❑ Entity Framework Overview

- <http://msdn.microsoft.com/en-us/library/bb399567.aspx>

## ❑ Entity Framework Getting Started

- <http://msdn.microsoft.com/en-us/data/ee712907>

## ❑ Dodatni materijali

- RPPP05 - dodatak.pdf : Primjeri s transakcijama i DataSetovima
-  ADODataset\DataSet.sln