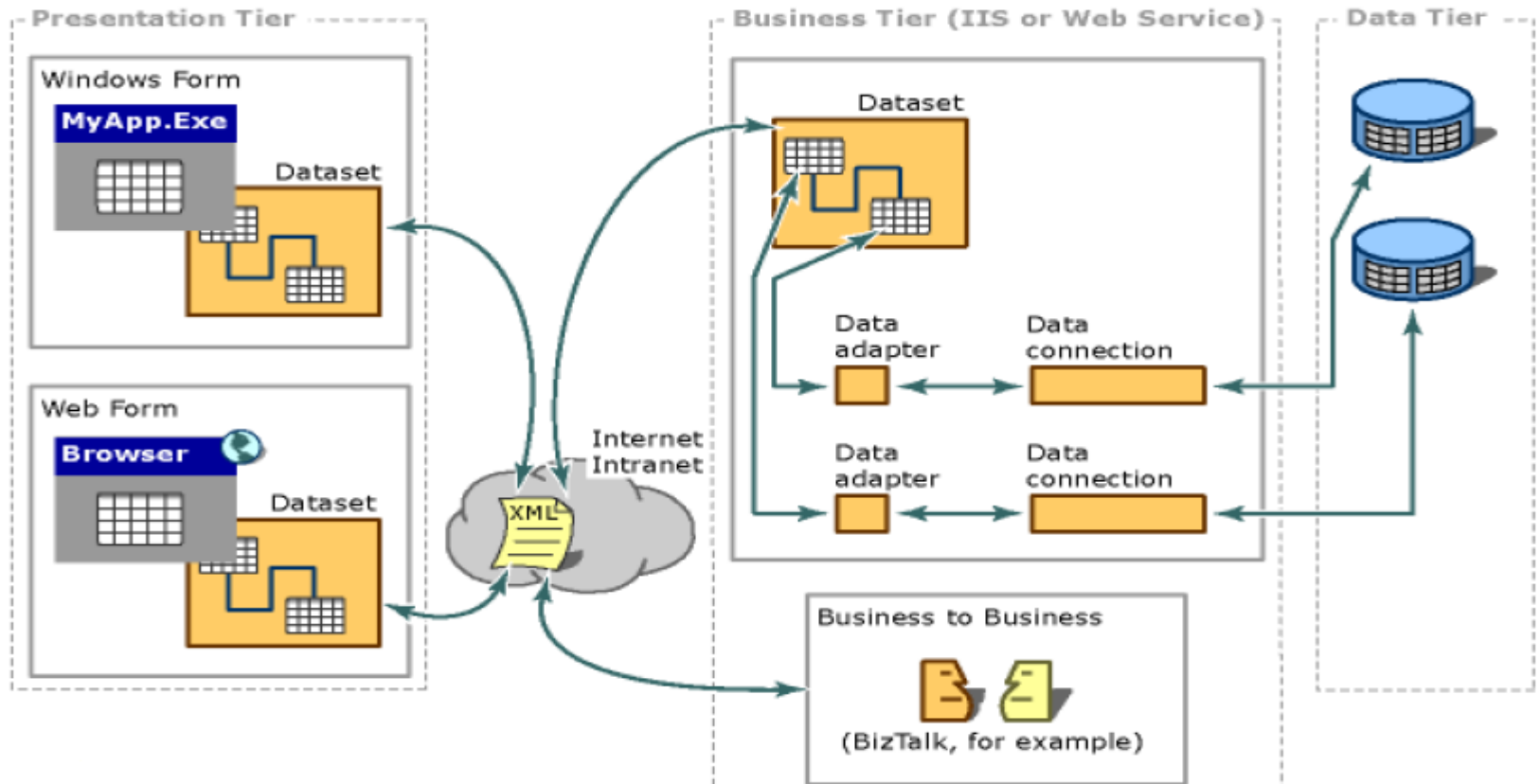

ADO.Net

Prof. Francesco Accarino

IIS Altiero Spinelli

Via Leopardi 132 Sesto San Giovanni

Applicazioni Three Tier



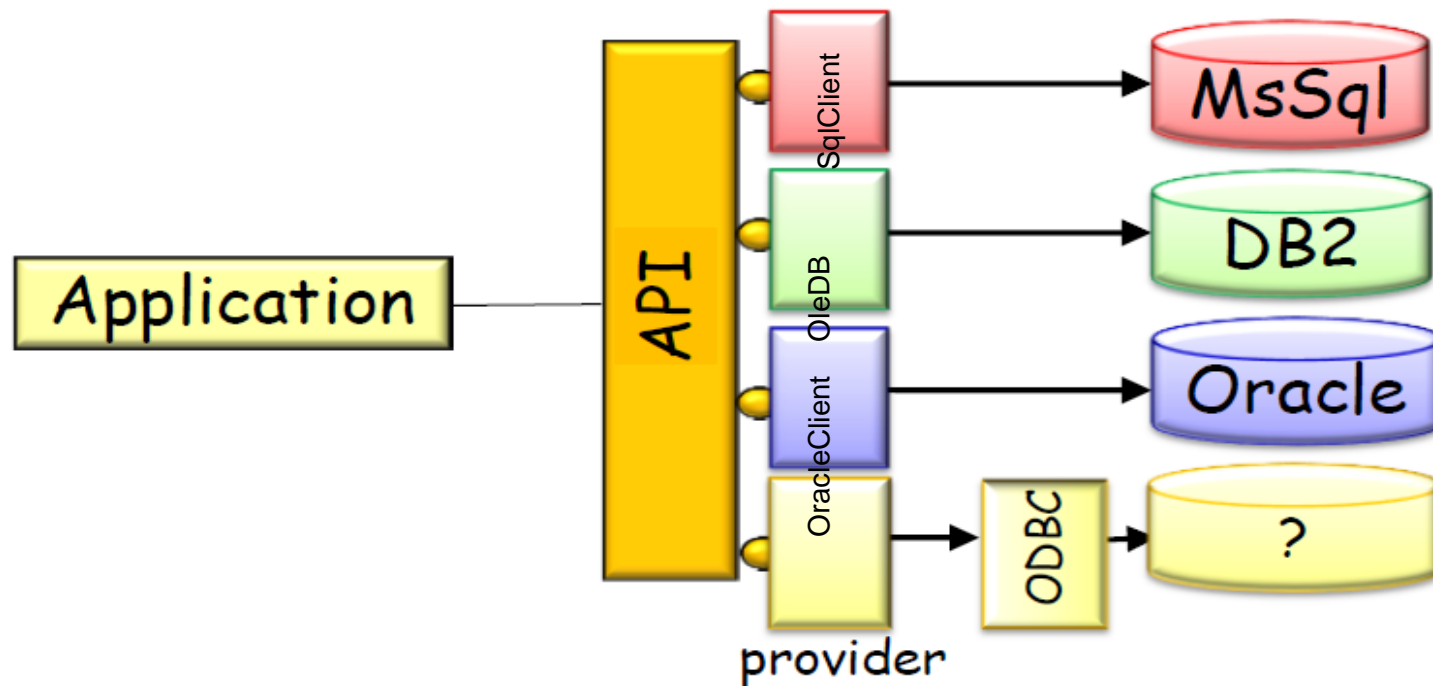
In ingegneria del software, l'espressione **architettura three-tier** ("a tre strati") indica una particolare architettura software che prevede la suddivisione del sistema in tre diversi moduli dedicati rispettivamente alla interfaccia utente, alla logica funzionale (*business logic*) e alla gestione dei dati persistenti.

Tali moduli sono intesi interagire fra loro secondo le linee generali del paradigma client-server (l'interfaccia è cliente della business logic, e questa è cliente del modulo di gestione dei dati persistenti) e utilizzando interfacce ben definite. In questo modo, ciascuno dei tre moduli può essere modificato o sostituito indipendentemente dagli altri. Nella maggior parte dei casi, si intende anche che i diversi moduli siano distribuiti su diversi nodi di una rete anche eterogenea.

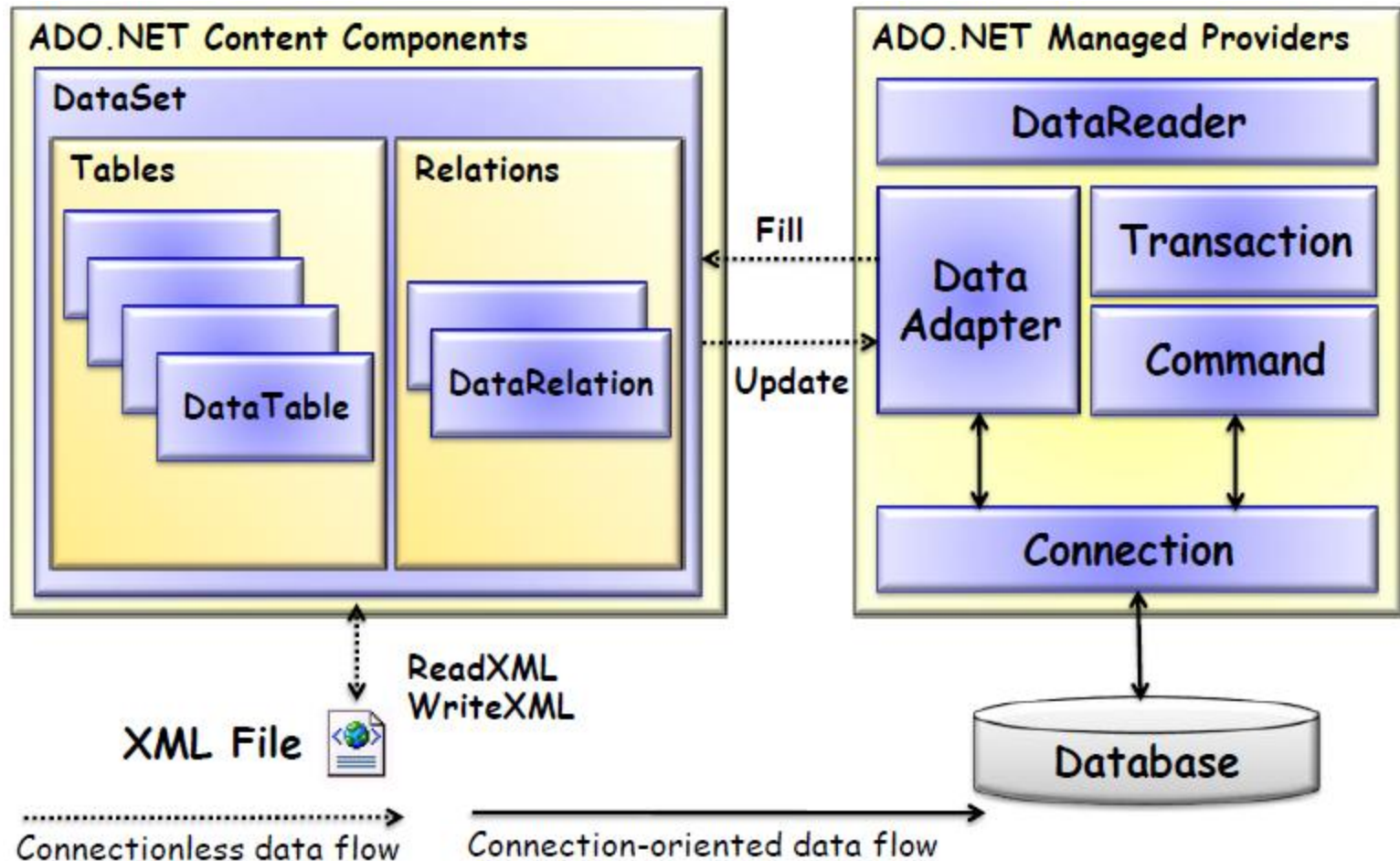
- ❑ ADO.NET è un insieme di librerie object-oriented che forniscono funzionalità di **accesso ai dati**.
 - ❑ Tipicamente la sorgente dati è un **database**, ma ADO.NET permette anche l'accesso a **file di testo**, **documenti XML**, **fogli Excel**, ecc. attraverso una stessa interfaccia.
- ❑ Prevede due diverse modalità di accesso ai dati:
 - ❑ Connection-oriented;
 - ❑ Connectionless.

Universal Data Access

- ❑ Modello per la connessione tra linguaggi a oggetti e database (relazionali e non relazionali).
 - ❑ Una stessa API per l'accesso a tutti i tipi di dato.
 - ❑ Sono necessarie librerie diverse (**Data Provider**) per l'accesso a differenti sorgenti dati. Ogni provider contiene un insieme di classi che implementano interfacce comuni.



Architettura di ADO.NET



Connection-oriented Connectionless

Due diverse modalità di accesso ai dati:

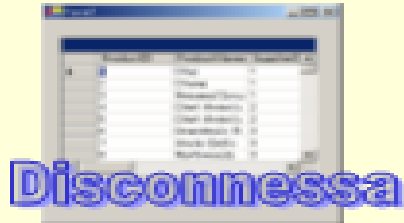
❑ Connection-oriented

- ❑ mantiene **attiva la connessione** al database;
- ❑ i **dati** sono sempre **aggiornati**;
- ❑ utile per applicazioni con le seguenti caratteristiche:
 - ❑ transazioni brevi;
 - ❑ pochi accessi paralleli;
 - ❑ necessità di dati sempre aggiornati.

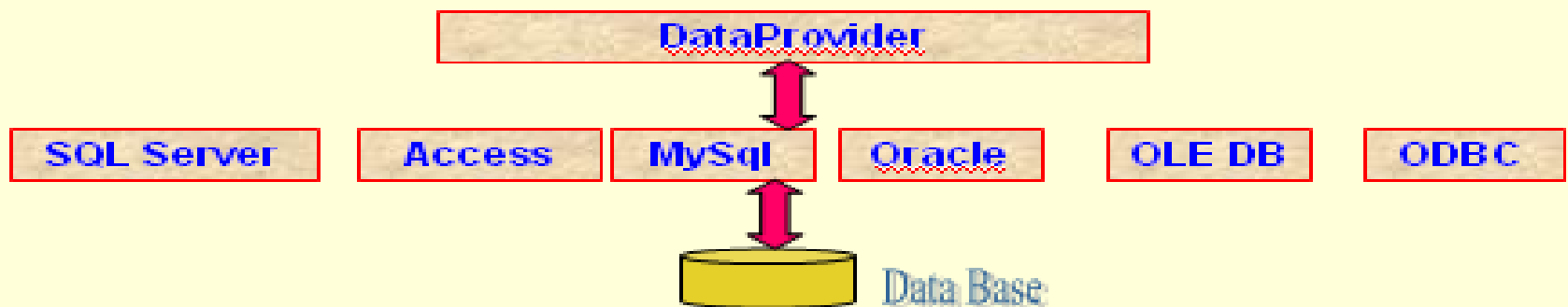
❑ Connectionless

- ❑ non utilizza una connessione permanente al db;
- ❑ i **dati** sono caricati in **memoria centrale**;
- ❑ le modifiche apportate ai dati in memoria centrale non sono immediatamente memorizzate nel db;
- ❑ utile per applicazioni che richiedono **numerosi accessi contemporanei** al database e che eseguono transazioni di lunga durata (es. applicazioni web).

Oggetti fondamentali in modalità connessa



DataReader
Command
Connection

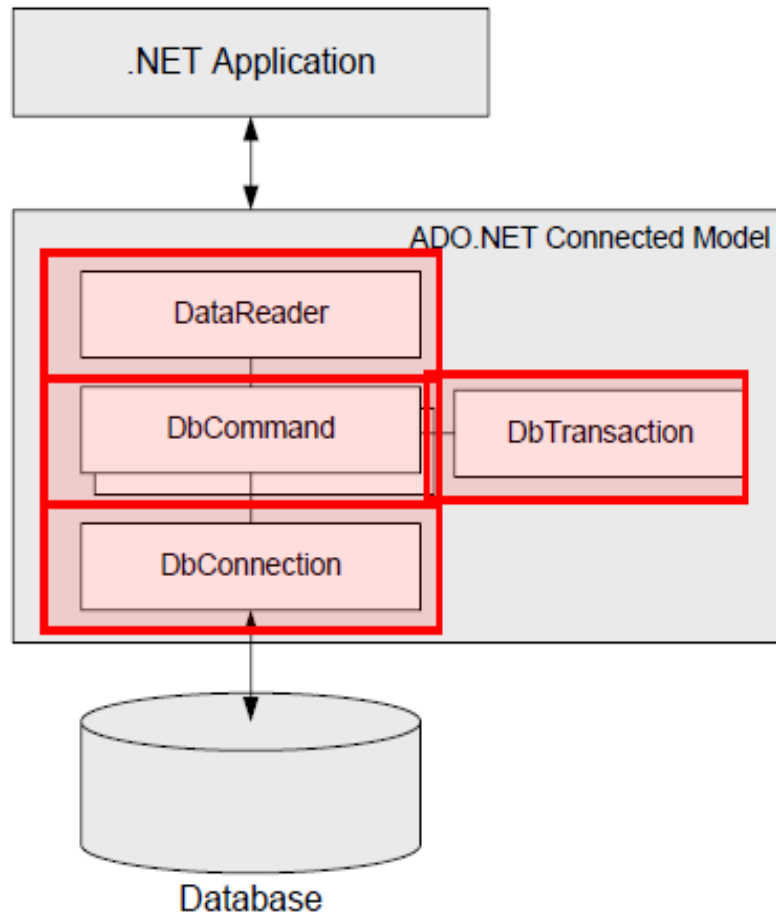


ADO.NET namespace

System.Data	Fornisce l'accesso alle classi che rappresentano l'architettura di ADO.NET
System.Data.Common	Classi condivise dai diversi DataProvider
System.Data.OleDb	Classi che permettono la connessione a sorgenti dati OLE DB compliant
System.Data.SqlClient	Classi ottimizzate per la connessione a Microsoft® SQL Server
System.Data.SqlTypes	Tipi di dato nativi in Microsoft® SQL Server

Architettura

- ❑ **DbConnection:**
 - rappresenta la connessione a un data source.
- ❑ **DbCommand:**
 - rappresenta un comando SQL.
- ❑ **DbTransaction:**
 - rappresenta una transazione;
 - i comandi possono essere eseguiti all'interno di una transazione.
- ❑ **DataReader:**
 - rappresenta il risultato di una query;
 - consente la lettura sequenziale delle righe.



Gerarchia delle Classi

Interfacce generali:

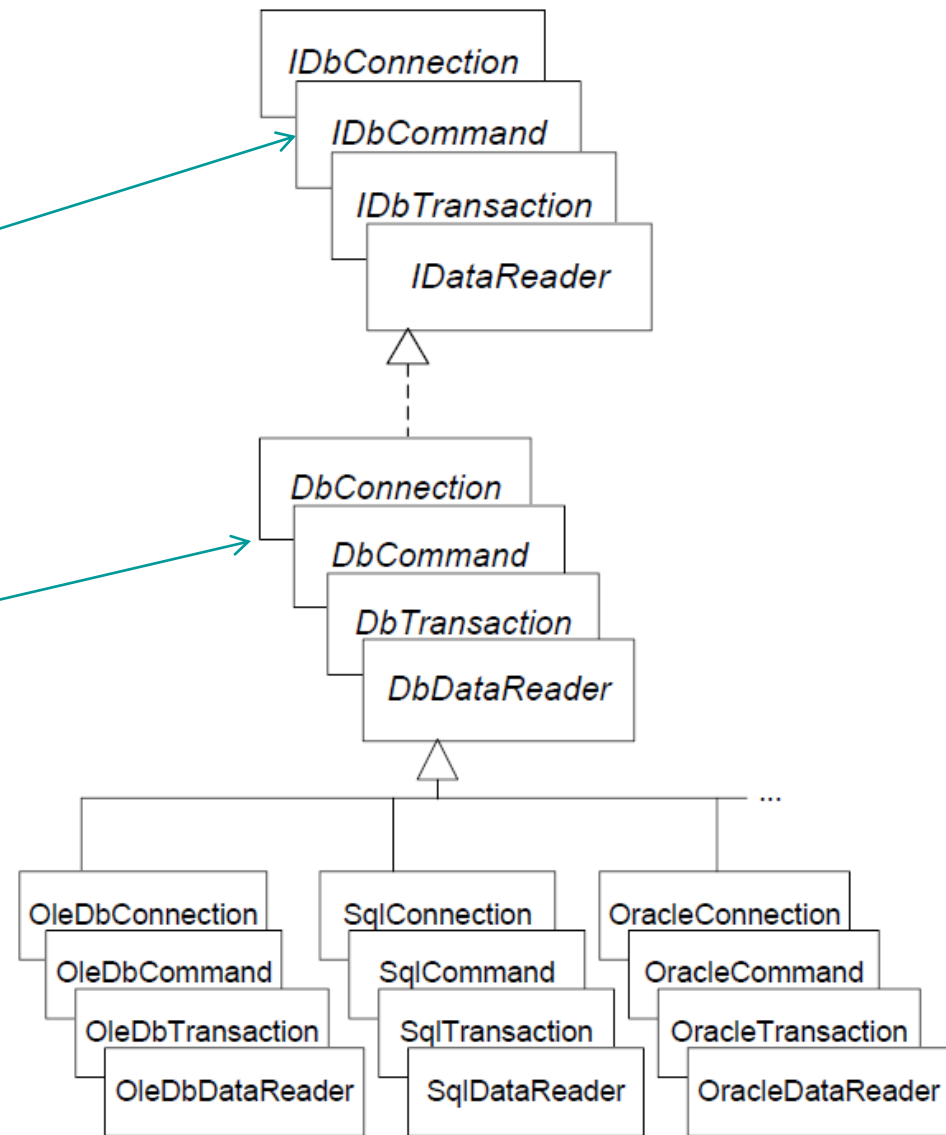
- IDbConnection
- IDbCommand
- IDbTransaction
- IDataReader

Classi base astratte:

- DbConnection
- DbCommand
- DbTransaction
- DbDataReader

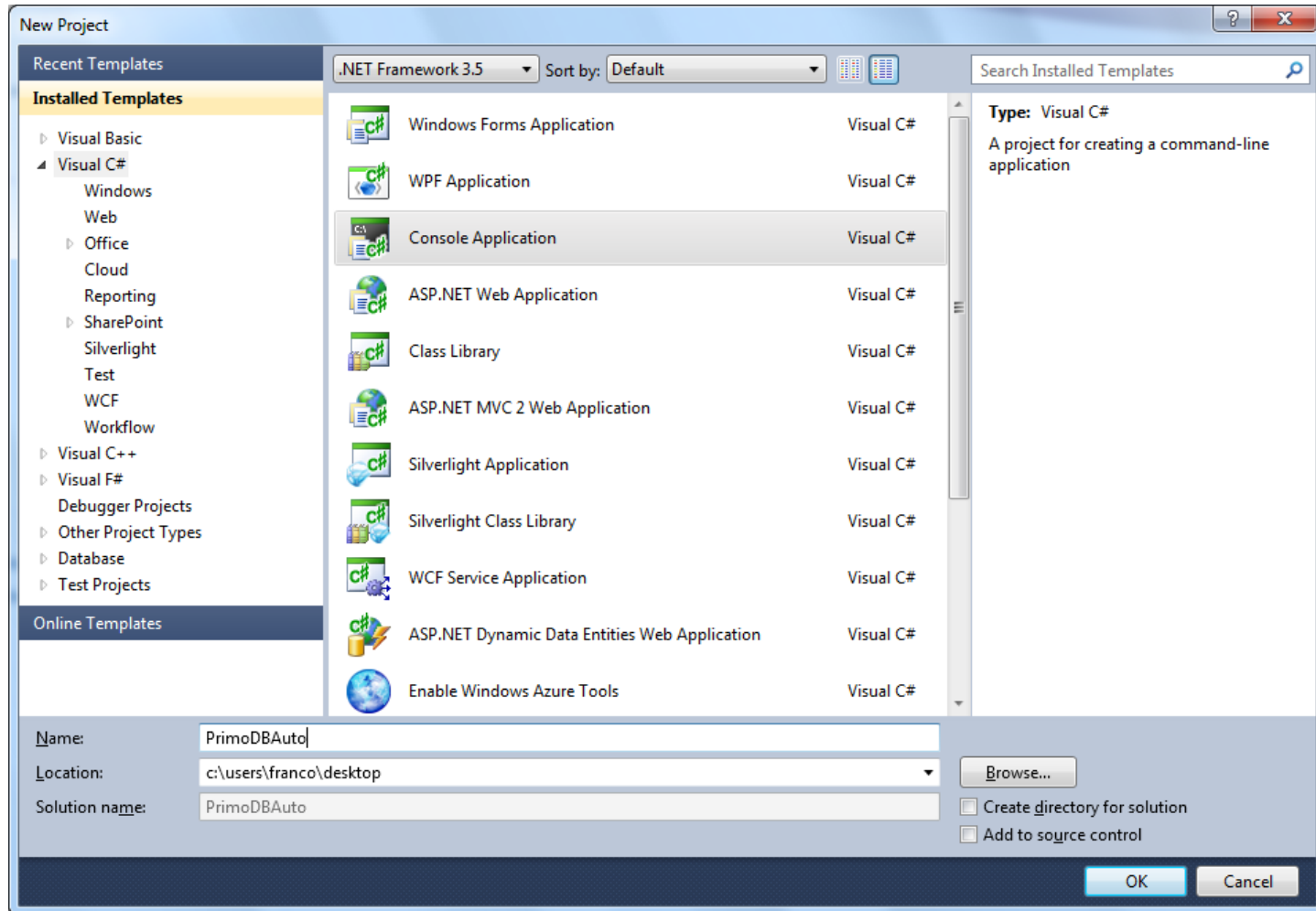
Implementazioni specifiche:

- OleDb, Sql, Oracle, Odbc, SqlCe



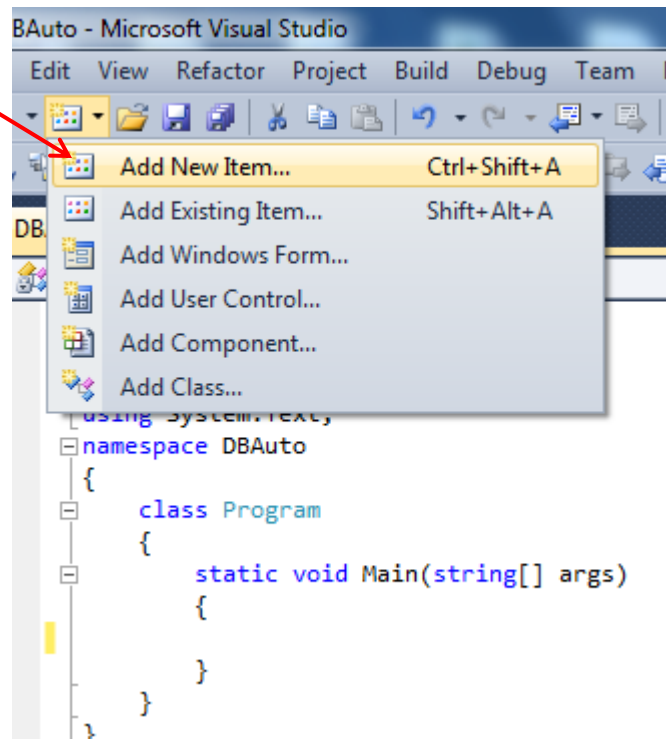
Primo Esempio Modalità Connessa

Apriamo Visual Studio Creiamo un nuovo progetto C# di tipo Console Application e lo chiamiamo PrimoDBAuto



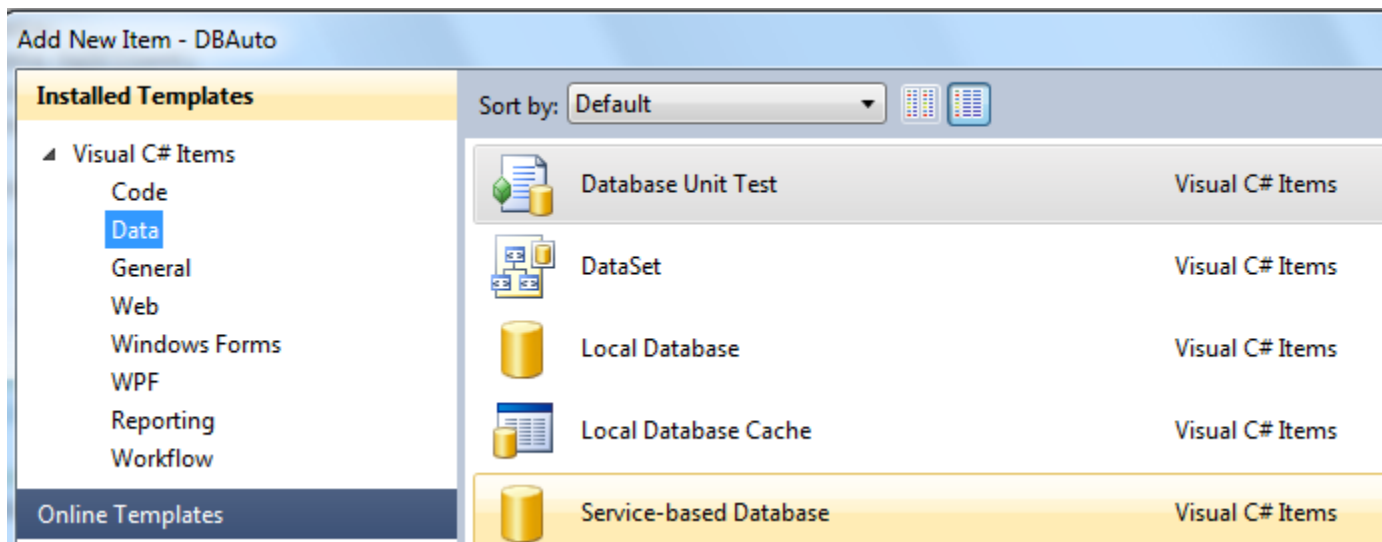
Primo Esempio Modalità Connessa

Visual Studio permette di creare un database SQL Server completamente dall'interno dell'ambiente di sviluppo e ovviamente permette anche un collegamento ad un database già esistente. Noi le proveremo entrambe partendo dalla prima: scegliere aggiungi nuovo elemento come mostrato in figura

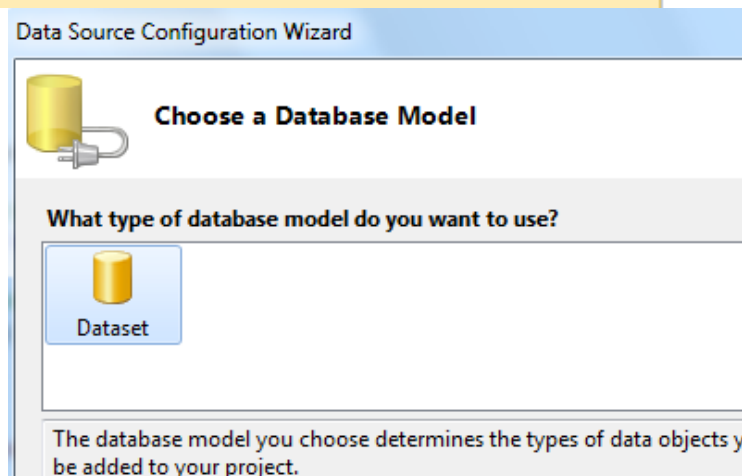


Primo Esempio Modalità Connessa

E nella finestra che si apre selezioneremo la classe data e poi service based database:

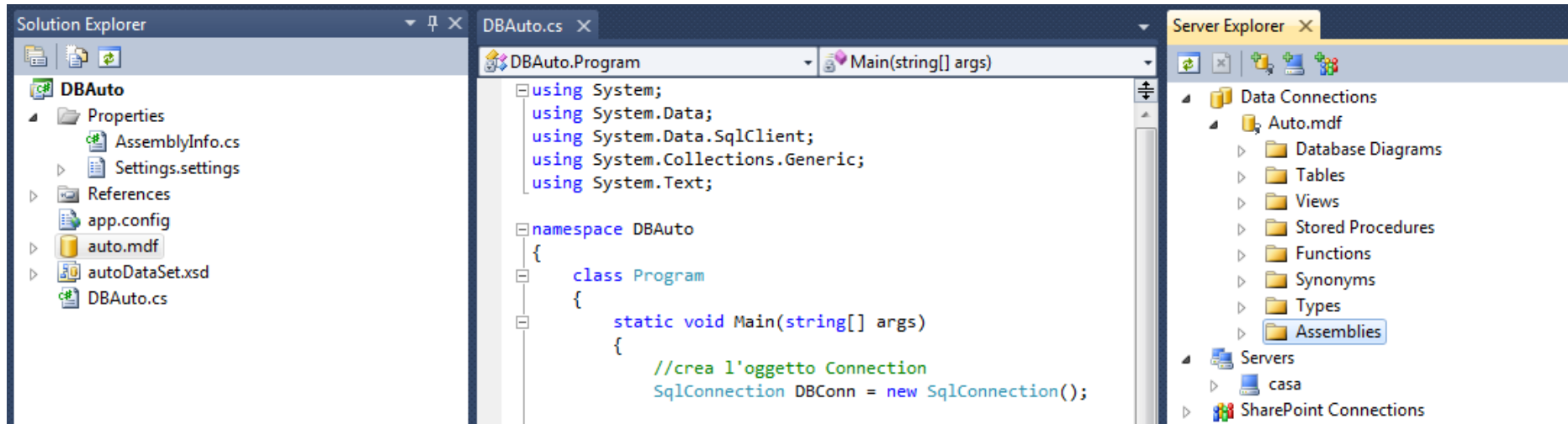


Diamo il nome Auto al nostro database e clicchiamo su next nella finestra successiva scegliamo Dataset clicchiamo su finish.

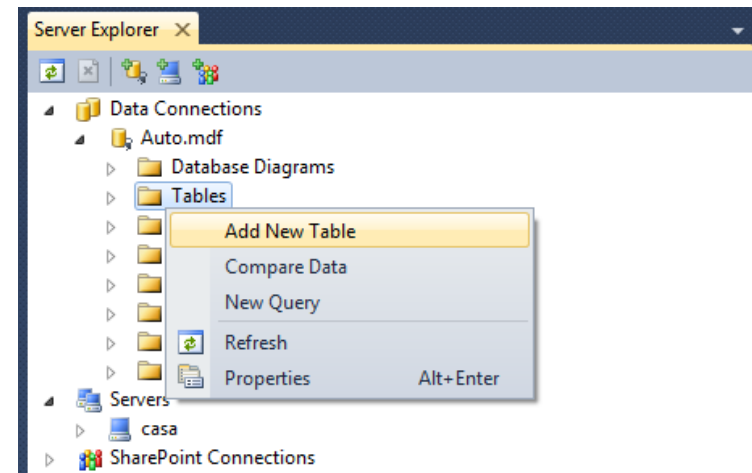


Primo Esempio Modalità Connessa

Facendo doppio click sul database appena creato nella finestra di solution explorer si apre la finestra Server explorer che ci permette di manipolare il database appena creato:

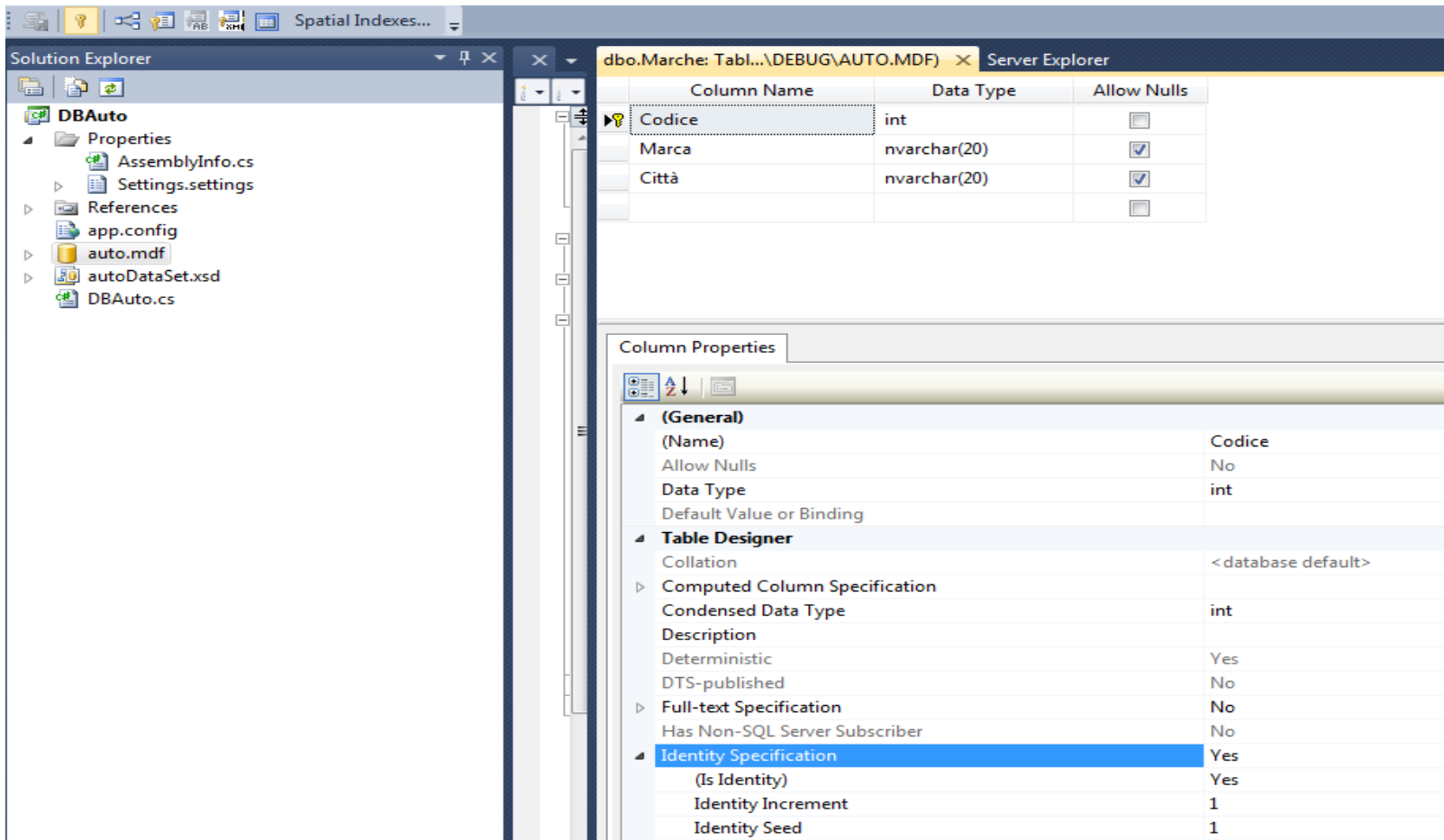


Facciamo click destro su tabelle e scegliamo aggiungi tabella:



Primo Esempio Modalità Connessa

La tabella avrà tre campi e per ogni campo dobbiamo poi scegliere il nome, il tipo ed altre caratteristiche ad esempio Codice è la chiave primaria, è di tipo intero e si auto incrementa. Per marca e città scegliamo il tipo nvarchar(20) che è il tipo carattere unicode a lunghezza limitata



The screenshot displays the SQL Server Enterprise Manager interface. On the left, the 'Solution Explorer' shows the project 'DBAuto' with files like 'AssemblyInfo.cs', 'Settings.settings', 'app.config', 'auto.mdf', 'autoDataSet.xsd', and 'DBAuto.cs'. The main pane shows the 'Server Explorer' with the 'dbo.Marche' database selected. A table named 'Tabl...' is open in the 'Table Designer' view. The table has three columns:

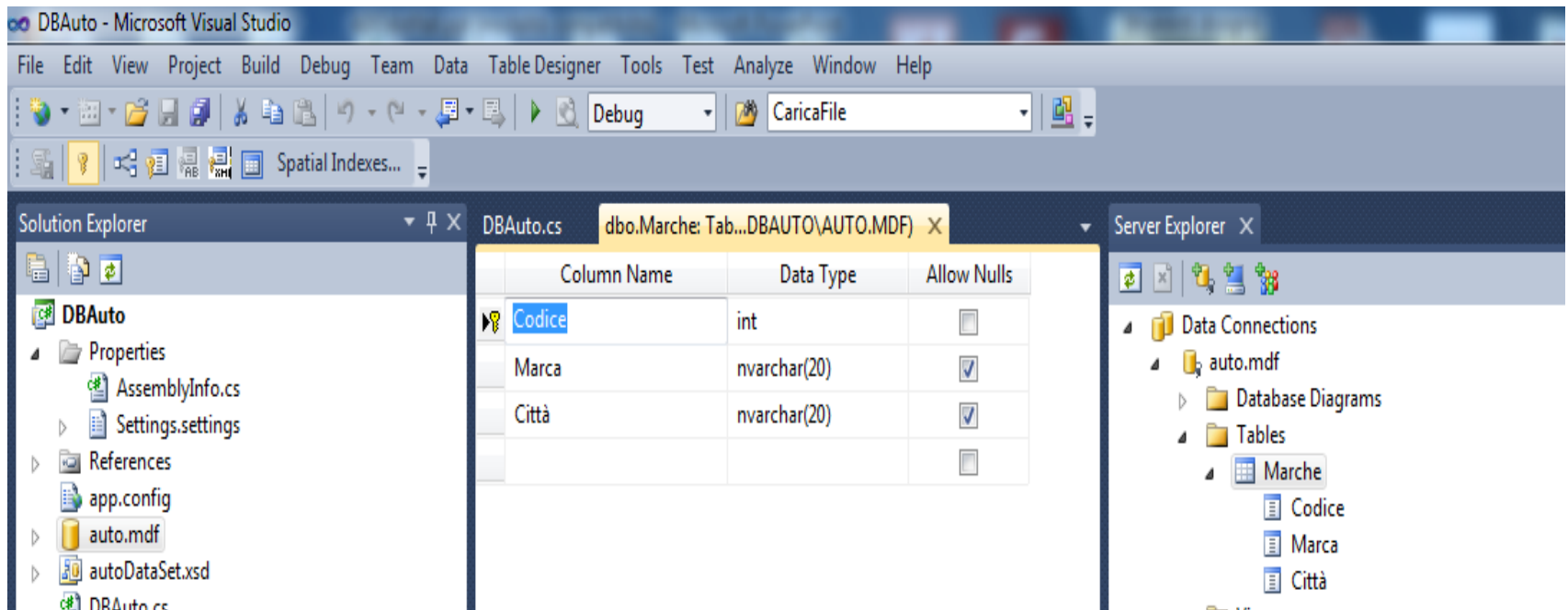
Column Name	Data Type	Allow Nulls
Codice	int	<input type="checkbox"/>
Marca	nvarchar(20)	<input checked="" type="checkbox"/>
Città	nvarchar(20)	<input checked="" type="checkbox"/>

The 'Column Properties' window is open for the 'Codice' column. It shows the following properties:

- (General)**
 - (Name): Codice
 - Allow Nulls: No
 - Data Type: int
 - Default Value or Binding:
- Table Designer**
 - Collation: <database default>
 - Computed Column Specification
 - Condensed Data Type: int
 - Description
 - Deterministic: Yes
 - DTS-published: No
 - Full-text Specification
 - Has Non-SQL Server Subscriber: No
- Identity Specification**
 - (Is Identity): Yes
 - Identity Increment: 1
 - Identity Seed: 1

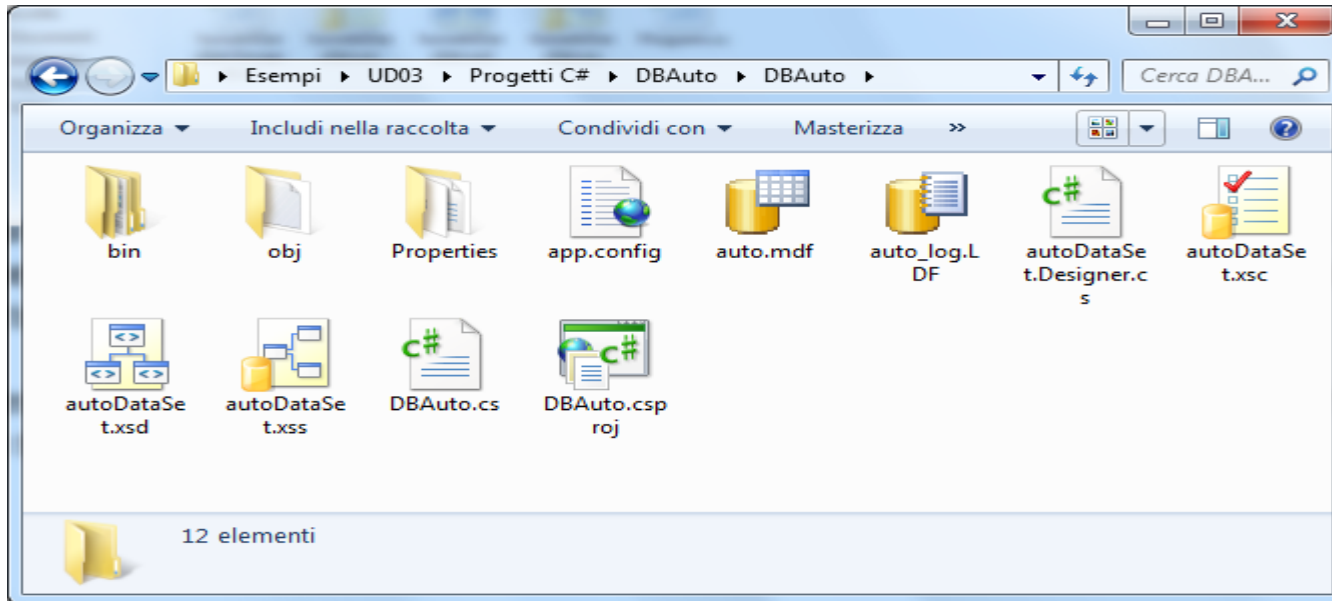
Primo Esempio Modalità Connessa

Chiudiamo la finestra struttura e diamo alla tabella il nome Marche possiamo vedere quanto abbiamo creato nella finestra Server explorer:



Primo Esempio Modalità Connessa

Salviamo tutto. Se ora andiamo a vedere nella cartella del nostro progetto troviamo sia il file database che il file di log.

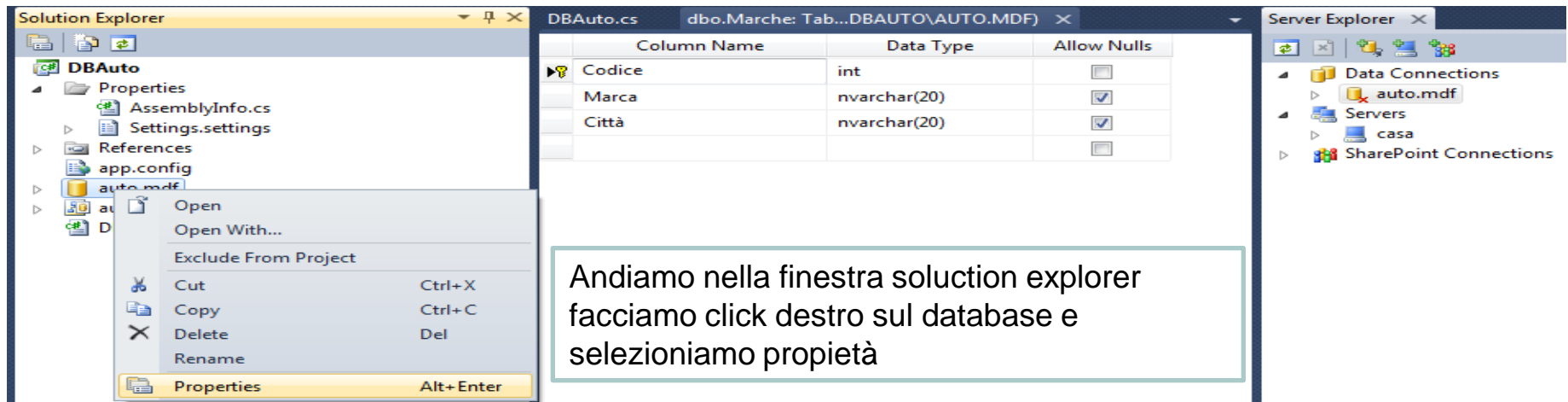
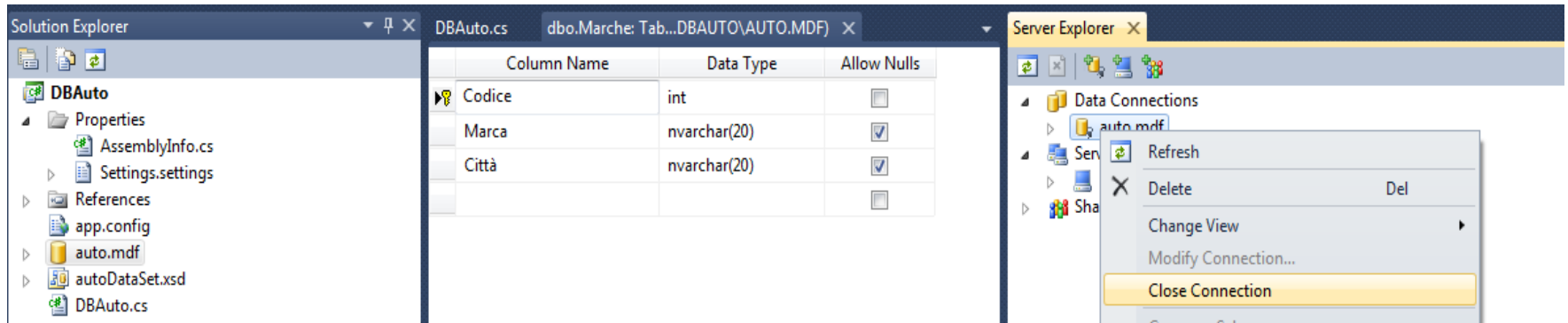


Quando viene fatta partire l'applicazione in modalità debug o release i due files `auto.mdf` e `auto.log` vengono automaticamente copiati nella cartella corrispondente.

Questi due files però sono vuoti e quindi le eventuali modifiche apportate ai dati non sarebbero riscontrate nelle sessioni successive.

Primo Esempio Modalità Connessa

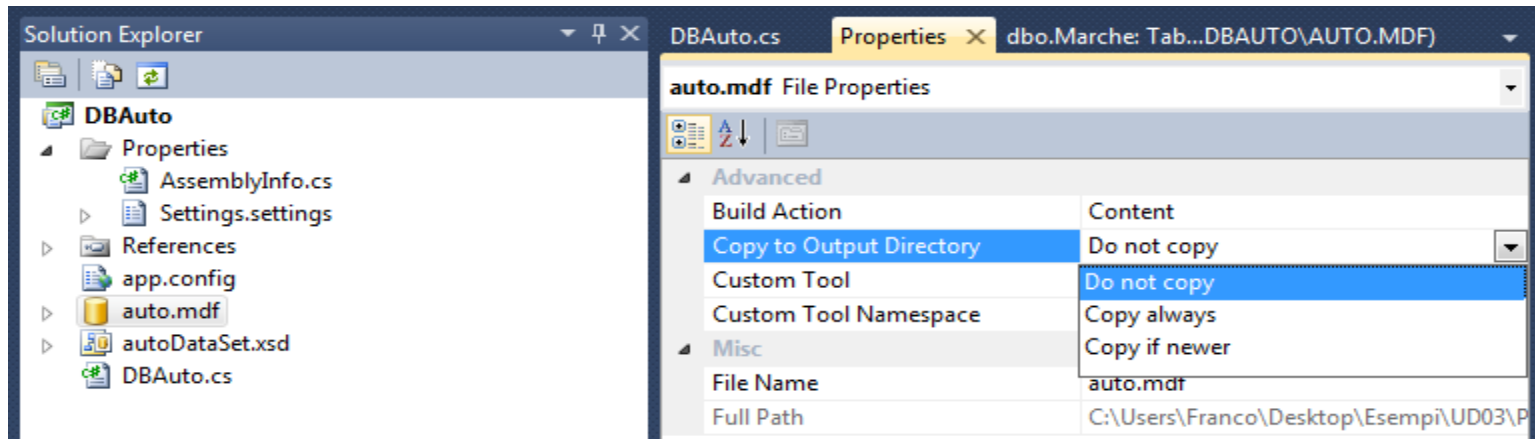
Per sopperire a questo inconveniente diciamo all'ambiente di non copiare in automatico questi due files nelle cartelle debug e release e li copiamo però noi manualmente. Prima però bisogna assicurarsi che il database non sia connesso altrimenti la copia non ci sarebbe consentita. In Server Explorer click destro sul database e scegliamo close connection



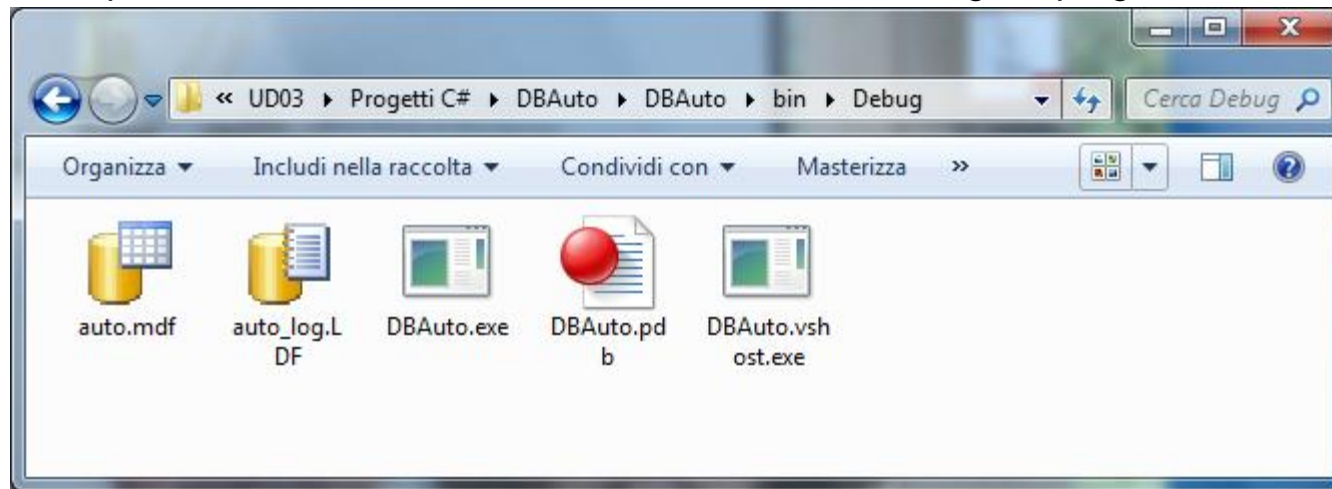
Andiamo nella finestra solution explorer facciamo click destro sul database e selezioniamo proprietà

Primo Esempio Modalità Connessa

E selezioniamo nella finestra proprietà di non copiare i file nelle cartelle debug e release.



E in Fine copiamo noi manualmente i due file nella cartella debug del progetto



Schema per l'accesso ai dati in modalità Connection-oriented

Dichiarazione della connessione

```
try {
```

Richiesta connessione al database

Esecuzione comandi SQL

Elaborazione risultati

Rilascio risorse

```
} catch ( Exception ) { Gestione eccezioni }
```

```
finally
```

```
{
```

```
    try {
```

Chiusura della connessione

```
    } catch (Exception) { Gestione eccezioni }
```

```
}
```

La stringa di connessione

Per creare una connessione ad un database dobbiamo creare una istanza di un oggetto `SqlConnection` e definire una cosiddetta stringa di connessione. Le modalità per ottenere la connessione sono due:

- ❑ `SqlConnection miaConn= new SqlConnection("stringa di connessione");
miaConn.Open();`
- ❑ `SqlConnection miaConn= new SqlConnection();
miaConn.ConnectionString="stringa di connessione";
miaConn.Open();`

Una stringa di connessione è formata da gruppi di coppie **chiave= valore** separati da un punto e virgola Esempi:

Standard Security

`"Server=myServerAddress; Database=myDataBase; User Id=myUsername; Password=myPassword;"`

Using an User Instance on a local SQL Server Express instance

`"Data Source=.\SQLEXPRESS; Integrated Security=true; User Instance=true;
AttachDbFilename=C:\MyFolder\MyDataFile.mdf; "`

Ci sono molte altre modalità per le quali si rimanda ai manuali SQL Server

```
using System;
using System.Data.SqlClient;
using System.Collections.Generic;
using System.Text;
namespace DBAuto
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            //crea l'oggetto Connection
            SqlConnection DBConn = new SqlConnection();
            try
            {
                //crea la stringa di connessione
                DBConn.ConnectionString = "Data Source=.\SQLExpress;Integrated Security=true;User Instance=true;
                    AttachDbFilename='C:\\Users\\Franco\\Desktop\\DBAuto\\DBAuto\\bin\\debug\\auto.mdf'";

                //apri la connessione
                DBConn.Open();
                Console.WriteLine("Connessione Aperta");
            }
            catch (Exception e)
            {
                Console.WriteLine("non rieso ad aprire la connessione");
            }
            //chiudi la connessione
            finally
            {
                try
                {
                    DBConn.Close();
                    Console.WriteLine("Connessione Chiusa");
                }
                catch (Exception e)
                {
                    Console.WriteLine("Problemi per chiudere la connessione");
                }
            }
            Console.ReadLine(); //attendi la pressione di un tasto
        }
    }
}
```

Esempio di applicazione console per verificare la connessione al nostro data base