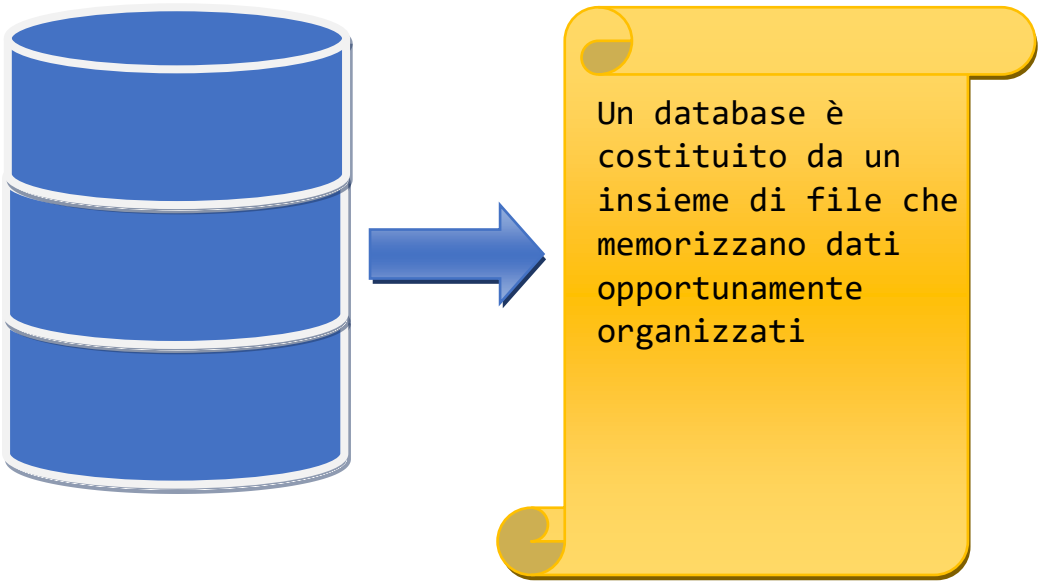
2019



INTRODUZIONE ALLA PROGRAMMAZIONE CON L'USO DI C# III° PARTE (gestione dati)

WWW.FILOWEB.IT

FILIPPO BRUNELLI

Introduzione alla programmazione con l'uso di C# III° parte

INDICE

CAPITOLO I

I DATABASE

I sistemi di database sono diventati irrinunciabili per la maggior parte dei programmi: tutti i software aziendali e quasi tutti i videogiochi si affidano ai database che non sono altro che un sistema organizzato per raccogliere i dati.
Uno dei principali tipi di database è quello relazionale, dove più tabelle sono messe in relazione tra di loro.
La tabella è l'elemento di partenza di ogni database ed è un insieme di righe e colonne dove ogni colonna contiene un dato relativo alla cosa che stiamo descrivendo e ogni riga corrisponde ad una istanza della cosa.
Se parlassimo di persone avremmo che ogni riga corrisponde ad una persona ed ogni colonna corrisponde ad una caratteristica della persona (nome, cognome, età, ecc.). In termini di database ogni colonna è un campo, ogni riga un record.

id	nome	cognome	eta	indirizzo	*Fare clic per aggiungere*
1	Filippo	Brunelli	46	Via le mani dal	
2	Pamela	Paolini	47	Via di torno	
3	Maurizio	Antonioni	52	Via Dos 2	
4	Alessandro	Alessandrovicl	25	Via Vai	
5	Giocanna	Colpodisole	30	Via il Calcare	
6	Noto	Ignoto	22	Via da qua	

Un database può essere composto di più tabelle. Ciò che rende un database relazionale è la presenza di legami fra le tabelle o di relazioni appunto.

Per i nostri esperimenti useremo un database creato con ACCESS (per chi volesse approfondire si consiglia vivamente di visitare il link https://www.filoweb.it/appunti.aspx la sezione ACCESS) contenente una sola tabella composta dai campi:

Id	Numeratore automatico
nome	Testo (String)
cognome	Testo (String)
eta	Numero (Int)
Indirizzo	Testo (String)

Creiamo quindi la nostra tabella e salviamola, per comodità, nella cartella dove stiamo creando il nostro progetto.

C# ED I DATABASE

Attraverso il Frameworks .NET C# permette di collegarsi a diversi database e per fare questo utilizza lo spazio nomi **System.Data** che fornisce accesso a classi che rappresentano l'architettura ADO.NET.
Il Name Sapce System.Data è composto da Classi, Interfacce, Enumerazioni e Delegati.

I principali Name Space per interfacciarsi con i vari database sono:

- **System.Data.SqlClient** offre il provider di dati .NET Framework per SQL Server
- **System.Data.Odbc** offre il provider di dati .NET Framework per ODBC
- **System.Data.OleDb** offre il provider di dati .NET Framework per OLE DB
- **System.Data.OracleClient** offre il provider di dati .NET Framework per Oracle

Per i nostri progetti che ci collegano ad Access useremo il il provider OLE DB che permette di connettersi ai database Sql Server, Access, documenti XML e molti altri.
Tra tutti i metodi che .Net mette a disposizione noi considerimo **DataReader** e **DataSet**.
Ecco un esempio di come creare la prima connessione ad un database:

```
using System;
using System.Data;
using System.Data.OleDb;

class database {

    public static void Main()
    {
    OleDbConnection myConn = new OleDbConnection("Provider=Microsoft.ACE.OLEDB.12.0; Data Source=
C:\\testC\\database.accdb");

    myConn.Open();
            Console.WriteLine("Connessione Aperta");
    myConn.Close();

    Console.WriteLine("Connessione Chiusa");
    }
}
```

NOTE IMPORTANTI: Negli esempio utilizziamo, per comodità OleDb; per i **database SQL** basta cambiare il nome delle classi mettendo al posto di tutti i OleDB SQL quindi: *SQLConnection*, *SQLCommand*, *SQLDataReader*, ecc., così come per utilizzare Oracle, o altri database supportati.
Per le connessioni ai vari database si consiglia di leggere le relative istruzioni per costruire le stringhe di connessione

DATAREADER

DataReader utilizza un flusso di dati di sola lettura e forward, il che vuol dire che recupera il record da databse e lo memorizza nel buffer di rete fornendolo ogni volta che lo si richieste; questo metodo necessita di una connessione aperta durante la sua elaborazione ed è particolarmente indicato per manipolare grandi quantità di dati e quando la velocità è un fattore essenziale.

LEGGERE I RECORD IN UNA TABELLA

La prima classe del nostro *Name Space* che analizziamo è **OleDbConnection** che rappresenta una connessione aperta ad un'origine dati; al momento dell'istanziazione è norma indicare una stringa che rappresenta l'origine dati.

```
OleDbConnection myConn = new OleDbConnection("STRINGA DI CONNESSIONE");
```

La seconda classe che ci interessa è **OleDbCommand** che serve per eseguire un comando SQL sulla nostra origine dati. Anche in questo caso al momento dell'instanziazione passo come parametri il mio comando SQL seguito dalla connessione:

```
OleDbCommand myCmd = new OleDbCommand("COMANDO SQL", "STRINGA DI CONNESSIONE");
```

Infine la terza classe che ci interessa è la classe **OleDbDataReader** che serve a leggere un flusso di dati di una riga e va associata al comando OleDB:

```
OleDbDataReader myReader = myCmd.ExecuteReader();
```

Con queste tre classi possiamo leggere i dati del nostro database!
Per estrapolare il contenuto della nostra tabella useremo I metodo **.GetInt32**() che server per recuperare un dato intero da un flusso di dati, mentre il metodo **.GetString**() lo estrae sotto forma di stringa.
Il metodo **.Read** della classe OleDbDataReader, in fine, serve a spostare il puntatore al record successivo.
Quando le operazioni sono concluse bisogna sempre ricordarsi di chiudere le connessioni ed i comandi aperti tramite **.Close()**.

Ecco il nostro programma per leggere il database:

```
using System;
using System.Data;
using System.Data.OleDb;

class database {

    public static void Main() {
    string strConn = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\testC\\database.accdb";
    string strSQL = "SELECT id, nome, cognome, eta FROM nomi";

    OleDbConnection myConn = new OleDbConnection(strConn);
    OleDbCommand myCmd = new OleDbCommand(strSQL, myConn);

    myConn.Open();
    OleDbDataReader myReader = myCmd.ExecuteReader();

    while (myReader.Read()) {
        Console.WriteLine("ID:" + myReader.GetInt32(0)); // Primo record Numero intero
        Console.WriteLine("Nome: \t" + myReader.GetString(1)); // Secondo record Stringa
        Console.WriteLine("Cognome:" + myReader.GetString(2)); // Terzo record Stringa
        Console.WriteLine("Età: \t" + myReader.GetInt32(3));  // Quarto record Numero intero
     }
    myReader.Close();
    myConn.Close();
    Console.ReadKey();
    }
}
```

Come si vede il flusso di dati da OleDbDataReader mi restituisce una serie di dati sequenziali che leggo da 0 a 3 come quando leggo i dati che arrivano da una matrice e di ogni uno devo sapere il tipo di dato per poterlo estrapolare (inter, sringa, boleano, ecc.).

SCRIVERE UN RECORD IN UNA TABELLA

Per poter scrivere dei dati in un database utilizzeremo le proprietà **.CommandText** e **.CommandType** del NameSpace **OleDbCommand**.
.CommandText serve per ottenere o imposta l'istruzione SQL da eseguire all'origine dati.
Un esempio di utilizzo è: .CommandText = "*SELECT [colonna],[colonna], ecc FROM [tabella] ORDER BY [colonna]*";
.CommandType serve, invece, ad impostare un valore che indica come viene interpretata la proprietà CommandText
Il meto **.ExecuteNonQuery** della classe OleDbCommand esegue un'istruzione SQL nella proprietà Connection

Ecco il programma che permette di inserire delle righe nel nostro database.

```
using System;
using System.Data;
using System.Data.OleDb;

class database {

    public static void Main(string[] args)
    {
     string nome = args[0];
     string cognome = args[1];
     int eta = int.Parse(args[2]);
     string strConn = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\testC\\database.accdb";

        OleDbConnection myConn = new OleDbConnection(strConn);
        OleDbCommand myCmd = new OleDbCommand();

        myCmd.CommandType = CommandType.Text;
        myCmd.CommandText = "INSERT INTO nomi (nome, cognome, eta) " + "VALUES ('"+nome+"', '"+co
gnome+"',"+eta+")";
        myCmd.Connection = myConn;

        myConn.Open();
            myCmd.ExecuteNonQuery();
        myConn.Close();
        Console.WriteLine("Valore inserito");
        Console.ReadLine();
    }
}
```

NOTE: Tramite *ExecuteNonQuery* è possibile eseguire operazioni di catalogo su di un database che non si limitano solo inserire dati nelle righe, ma anche creare tabelle, aggiungere righe, eliminare tabelle e colonne, ecc.

Supponendo di averlo compilato con il nome aggiungi ecco il nostro programma in esecuzione :

```
c:\testC>csc aggiungi.cs
Compilatore Microsoft (R) Visual C# versione 3.100.119.28106 (58a4b1e7)
Copyright (C) Microsoft Corporation. Tutti i diritti sono riservati.


c:\testC>aggiungi Mosca Alata 31
Valore inserito
```

MODIFICARE UN RECORD IN UNA TABELLA

Per modificare i dati, per quanto concerne le classi da utilizzare, si usano gli stessi metodi che sono utilizzati per inserirli all'interno di un database con la sola modifica della query utilizzata:

```
myCmd.CommandText = "UPDATE nomi SET nome='" + nome + "', cognome='" + cognome +"', eta=" + eta +
" WHERE id=" + id;
```

Nella query si usa il comando UPDATE e la clausola WHERE che identifica l'ID univoco che è associato al nome. Se non scrivo l'id corretto, nel nostro esempio che segue, mi viene generato un errore

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Text;

class database {

    public static void Main(string[] args)
    {
     int id = int.Parse(args[0]);
     string nome = args[1];
     string cognome = args[2];
     int eta = int.Parse(args[3]);

     string strConn = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\testC\\database.accdb";

        OleDbConnection myConn = new OleDbConnection(strConn);
        OleDbCommand myCmd = new OleDbCommand();

        myCmd.CommandType = CommandType.Text;
        myCmd.CommandText = "UPDATE nomi SET nome='" + nome + "', cognome='" + cognome +"', eta="
 + eta +" WHERE id=" + id;
        myCmd.Connection = myConn;
        myConn.Open();
            myCmd.ExecuteNonQuery();
        myConn.Close();
        Console.WriteLine("ID " + id +" modificato correttamente");
        Console.ReadLine();
    }
}
```

Una volta compilato il nostro programma (e chiamato modifica.exe) per eseguirlo sarà sufficiente lanciarlo con il riferimento all'ID da modificare

```
Nome:   Pinco
Cognome:Pallo
Età:    65
ID:9
Nome:   Mosca
Cognome:Alata
Età:    31

c:\testC>modifica 9 Marina Stella 22
```

CANCELLARE UN RECORD IN UNA TABELLA

Per eliminare un record si usa la clausola DELETE al posto di update

```
myCmd.CommandText = "DELETE FROM nomi WHERE id=" + id;
```

e al programma viene trasferito solo l'ID che si vuole eliminare; se volessi invece eliminare tutti i record che contendono un nome simile a "filippo" (quindi Filippo, filippo, FILIPPO,, ecc.) modifico la query così:

```
myCmd.CommandText = "DELETE FROM nomi WHERE nome LIKE'%filippo%'";
```

TROVARE DEI RECORD IN UNA TABELLA

Per cercare un record all'interno di un database utilizzo lo stesso metodo che si utilizza leggere i record da una tabella modificando la query in:

```
string strSQL = "SELECT id, nome, cognome, eta FROM nomi WHERE nome LIKE'%" + nome + "%'";
```

In questo modo passo al programma il parametro nome che cerco nella tabella. La clausola LIKE serve per cercare qualcosa che assomiglia e il carattere % che precede e segue equivale al carattere jolly *
(per i riferimenti alle query si consiglia di guardare la lezione su access su www.filoweb.it).
Dato che *nome* è di tipo stringa devo inserire la ricerca tra due apici ' ', se avessi cercato per ID o per età non sarebbe stato necessario:

```
string strSQL = "SELECT id, nome, cognome, eta FROM nomi WHERE eta=" + eta;
```

Nell'esempio di query sopra avrei ottenuto la lista di tutti i nominativi che hanno età corrispondente alla mia richiesta

Ecco il programma completo.

```
using System;
using System.Data;
using System.Data.OleDb;

class database {

    public static void Main(string[] args)
    {
   string nome = args[0];
   string strConn = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\testC\\database.accdb";
   string strSQL = "SELECT id, nome, cognome, eta FROM nomi WHERE nome LIKE'%" + nome + "%'";

    OleDbConnection myConn = new OleDbConnection(strConn);
    OleDbCommand myCmd = new OleDbCommand(strSQL, myConn);
    myConn.Open();
    OleDbDataReader myReader = myCmd.ExecuteReader();
    while (myReader.Read())
     {
        Console.WriteLine("ID:" + myReader.GetInt32(0));
        Console.WriteLine("Nome: \t" + myReader.GetString(1));
        Console.WriteLine("Cognome:" + myReader.GetString(2));
        Console.WriteLine("Età: \t" + myReader.GetInt32(3));
     }

    myReader.Close();
    myConn.Close();
    Console.ReadLine();
    }
}
```

CAPITOLO II

DATASET

L' oggetto **DataSet** è un oggetto che corrisponde ad una rappresentazione in memoria di un database. Al suo interno contiene l'oggetto **DataTable**, che corrisponde alle tabelle del database, che contiene a sua volta l'oggetto **DataColumn**, che definisce la composizione delle righe chiamate **DataRow**.
I vantaggi di usare un DataSet è che permette di definire relazioni tra le DataTable di un DataSet, allo stesso modo di quanto avviene in un database.
Una delle principali differenze tra un DataReader (visto precedentemente) e un DataSet è che il primo manterrà una connessione aperta al database fino a quando non la si chiude, mentre un DataSet sarà un oggetto in memoria. Questo porta come conseguenza che un DataSet risulta più *pesante* di un DataReader.
Un DataReader è di tipo forward alla lettura di dati mentre un DataSet permette di spostarti avanti e indietro e manipolare i dati.
Una delle caratteristiche aggiuntive dei DataSet è che possono essere serializzati e rappresentati in XML (l'uso dei file XML sarà oggetto del prossimo capitolo). mentre i DataReader non possono essere serializzati.
Se però si ha una grande quantità di righe da leggere dal database un DataReader è preferibile ad un DataSet che carica tutte le righe, occupa memoria ed influenzare scalabilità.
Uno dei principali oggetti utilizzati da DataSet è **DataAdapter** che funge da ponte tra un Dataset ed il Database.

LEGGERE I RECORD IN UNA TABELLA TRAMITE DATASET

Abbiamo introdotto i dataset e abbiamo visto che caricano in memoria i risultati di una query; una volta fatto questo sarà sufficiente scorrere i dati come si fa con un normale array…

```
using System;
using System.Data;
using System.Data.OleDb;
class database {
    public static void Main()
    {
     string strConn = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\testC\\database.accdb";
     string strQuery =" Select * from nomi";

        OleDbConnection myConn = new OleDbConnection(strConn);
        OleDbDataAdapter myAdapter = new OleDbDataAdapter(strQuery, myConn);
        DataSet myDataSet = new DataSet();
        myConn.Open();

        myAdapter.Fill(myDataSet, "nomi");
        DataTable tblNomi = myDataSet.Tables["nomi"];

        foreach (DataRow drnomi in tblNomi.Rows)
            {
                Console.WriteLine("ID:" + drnomi ["ID"]);
                Console.WriteLine("Nome: \t" + drnomi ["nome"].ToString() );
                Console.WriteLine("Cognome:" + drnomi ["cognome"].ToString() );
                Console.WriteLine("eta:" + drnomi ["eta"]);
                Console.WriteLine("----------------------");
            }
           myAdapter.Dispose();
    }
}
```

Come si vede abbiamo caricato i contenuti della tabella all'interno di tblNomi tramite DataAdapter e poi fatto scorrere.

Un'altra cosa che si nota è che in questo caso non dobbiamo preoccuparci di aprire e chiudere una connessione in quanto viene tutto gestito dal DataSet.

SCRIVERE UN RECORD IN UNA TABELLA TRAMITE DATASET

Il metodo per aggiungere una nuova riga di dati ad un database è simile a quello visto sopra per leggerle.
Anche qua creiamo un DataSet contenente la tabella che ci interessa, poi usiamo il metodo **.Add** per aggiungere una nuova riga passando una matrice di valori, tipizzata come Object.
Importante è ricordarsi di utilizzare il metodo **.Update** per aggiornare la tabella del nostro database una volta che sono stati inseriti i dati del DataSet in memoria.
Un altro metodo importante è **.MissingSchemaAction** che indica quale azione eseguire quando si aggiungono dati all'oggetto DataSet e risultano mancanti gli oggetti.
Nel nostro caso MissingSchemaAction**.AddWithKey** aggiunge l'id (che è un valore tipo *int* autoincrementale univoco) automaticamente.

Altre opzioni per MissingSchemaAction possono essere:

- **Add:** Aggiunge le colonne necessarie per completare lo schema;
- **Ignore:** Vengono ignorate le colonne supplementari;
- **Error:** Se il mapping della colonna specificata risulta mancante, verrà generato l'oggetto InvalidOperationException.

```
using System;
using System.Data;
using System.Data.OleDb;

class database {
    public static void Main(string[] args)
    {
     string nome = args[0];
     string cognome = args[1];
     int eta = int.Parse(args[2]);
     string strConn = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\testC\\database.accdb";
     string strQuery =" Select * from nomi";

    OleDbConnection myConn = new OleDbConnection(strConn);
    OleDbDataAdapter myAdapter = new OleDbDataAdapter(strQuery, myConn);
    DataSet myDataSet = new DataSet();
    DataRow myDataRow;

    OleDbCommandBuilder myBuilder = n4111.ew OleDbCommandBuilder(myAdapter);

    myAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey
    myAdapter.Fill(myDataSet, "nomi");
        myDataRow = myDataSet.Tables["nomi"].NewRow();
        myDataRow["nome"] = nome;
        myDataRow["cognome"] = cognome;
        myDataRow["eta"] = eta;

         myDataSet.Tables["nomi"].Rows.Add(myDataRow);

    myAdapter.UpdateCommand = myBuilder.GetUpdateCommand();
    myAdapter.Update(myDataSet,"nomi");
   myAdapter.Dispose();
    Console.WriteLine("Valore inserito");
    }
}
```

Come si vede, inserendo i dati in questo modo è più facile evitare errori anche se si ha un utilizzo maggiore di memoria.

MODIFICARE UN RECORD IN UNA TABELLA TRAMITE DATASET

Per modificare un record tramite DataSet è sufficiente caricare in memoria i record e tramite:

```
myDataRow = myDataSet.Tables["nomi"].Rows.Find(id);
```

recuperare l'ID (che è un numero univoco autoincrementale) della riga che voglio modificare e sovrascriverne i contenuti.

```
using System;
using System.Data;
using System.Data.OleDb;

class database {

    public static void Main(string[] args)
    {

        int id = int.Parse(args[0]);
        string nome = args[1];
        string cognome = args[2];
        int eta = int.Parse(args[3]);
        string strConn = "Provider=Microsoft.ACE.OLEDB.12.0; Data Source=C:\\testC\\database.accd
b";
        string strQuery =" SELECT * FROM nomi WHERE id=" + id;

    OleDbConnection myConn = new OleDbConnection(strConn);
    OleDbDataAdapter myAdapter = new OleDbDataAdapter(strQuery, myConn);
    DataSet myDataSet = new DataSet();
    DataRow myDataRow;

    OleDbCommandBuilder myBuilder = new OleDbCommandBuilder(myAdapter);

    myAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
    myAdapter.Fill(myDataSet, "nomi");


        myDataRow = myDataSet.Tables["nomi"].Rows.Find(id);
        myDataRow["nome"] = nome;
        myDataRow["cognome"] = cognome;
        myDataRow["eta"] = eta;

        myAdapter.UpdateCommand = myBuilder.GetUpdateCommand();
        myAdapter.Update(myDataSet,"nomi");
        myAdapter.Dispose();
        Console.WriteLine("Valore Modificato");
    }
}
```

TROVARE UN RECORD TRAMITE DATASET

Per recuperare uno o più record in un DataSet è sufficiente modificare la stringa della query che si usa per legere i dati:

```
string strQuery =" Select * from nomi WHERE nome LIKE '%" + nome + "%'";
```

NOTE: per una spiegazione rapida dell'uso delle istruzioni SQL si consiglia di guardare le appendici a fine lezione o i tutorial online.

DATASET O DATAREADER?

Prima di considerare quale sia migliore o meno tra le due soluzioni dobbiamo considerare cosa richiede la nostra applicazione, la velocità di esecuzione, la scalabilità, le dimensioni di dati, ecc.
Sia l'uso di DataSet che di DataReader sono molto frequenti nelle applicazioni .Net per recuperare i dati da un database.
Possiamo fare un piccolo raffronto veloce, per vedere quando usare uno o l'altro:

DataSet	DataReader
• Applicazione Windows • Dati non troppo grandi • Restituzione di più tabelle • Il risultato è da serializzare • Architettura disconnessa • Per inviare attraverso i livelli • Memorizzazione nella cache dei dati • Non è necessario aprire o chiudere la connessione	• applicazione web • Dati di grandi dimensioni • Restituzione di più tabelle • Per un accesso rapido ai dati • Deve essere esplicitamente chiuso • Il valore del parametro di output sarà disponibile solo dopo la chiusura • restituisce solo una riga dopo la lettura

DataReader è uno stream di sola lettura e forward. Recupera il record da databse e memorizza nel buffer di rete e fornisce ogni volta che lo si richiede. DataReader rilascia i record mentre la query viene eseguita e non attende l'esecuzione dell'intera query. Pertanto è molto veloce rispetto al set di dati. Se è necessario un accesso di sola lettura ai dati, è utile eseguire spesso operazioni all'interno di un'applicazione utilizzando un DataReader.
Con DataSet i valori vengono caricati in memoria significa lavorare con un metodo disconnesso ovvero che non necessita di una connessione aperta durante il lavoro sul set di dati.
Quando viene eseguita la query, tramite DataReader, la prima riga viene restituita tramite lo stream e memorizzata sul client. Il metodo Read() di DataReader passa al database e legge la riga successiva, la memorizza nuovamente sul client. Lo stream rimane quindi collegato al database, pronto a recuperare il record successivo.
Quindi, affinché il metodo Read() recuperi la riga successiva, deve esistere la connessione al database. Pertanto, utilizzando DataReader è possibile gestire solo una riga alla volta dato che non memorizza tutti i record sul client durante l'esecuzione della query come nel caso di DataSet.
Quando si popola un elenco o si recuperano 10.000 record è meglio utilizzare Data Reader; se è necessario recuperare un'enorme quantità di dati in un processo aziendale, il caricamento di un DataSet, il trasferimento dei dati dal database e la memorizzazione in memoria può richiedere del tempo.
In un'applicazione Web in cui centinaia di utenti potrebbero essere connessi, la scalabilità diventa un problema. Se si prevede che questi dati vengano recuperati e per l'elaborazione DataReader potrebbe accelerare il processo in quanto recupera una riga alla volta e non richiede le risorse di memoria richieste dal DataSet.
Proprio come un database DataSet, d'altronde, è costituito da un insieme di tabelle e relazioni tra di loro, insieme a vari vincoli di integrità dei dati sui campi delle tabelle.
Tramite DataSet è possibile selezionare le tabelle dei moduli di dati, creare viste basate sulla tabella e chiedere righe figlio sulle relazioni. Inoltre, DataSet offre funzionalità avanzate come il salvataggio di dati come XML, eccetera.
Concludendo diciamo che se parliamo della performance, c'è un enorme divario tra i due metodi: DataReader è circa trenta volte più performante del DataSet. Nei DataSet, poiché tutti i dati verranno caricati in memoria, c'è un sovraccarico in memoria, ma i DataSete permettono una flessibilità maggiore nella manipolazione dei dati.

CAPITOLO III

XML

XML è un linguaggio di markup creato dal World Wide Web Consortium (W3C) per definire una sintassi per la codifica dei documenti.
A differenza di altri linguaggi di markup, come ad esempio come HTML, l' XML non ha un linguaggio di markup predefinito ma consente agli utenti di creare i propri simboli di marcatura per descrivere il contenuto del documento, creando un insieme di simboli illimitato e auto-definente.
L'XML è oggi molto utilizzato anche come mezzo per l'esportazione di dati tra diversi DBMS, è usato nei file di configurazione di applicazioni e sistemi operativi ed in molti altri settori.
Per fare alcuni esempi il formato XML viene usato per gli RSS Feed, per l'interscambio delle fatture elettroniche,lo scambio dati tra dati tra diversi portali e software gestionali, per la formattazione nei documenti nel formato Open Document, per le confeme delle PEC, ecc..
Un file XML non è altro che un file di testo che deve rispettare alcune regole:

- l'intestazione (<?xml version="1.0" encoding="UTF-8"?>) deve comprendere l'idendificazione del file XML appunto, la versione e l'encoding dei caratteri, ogni
- Deve esistere uno e soltanto uno elemento Root
- Ogni Tag apertodeve essere chiuso

Ecco come può apparire un file XML contenente i nostri dati

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nomi>
    <persona>
        <id>1</id>
        <nome>Filippo</nome>
        <cognome>Brunelli</cognome>
        <eta>47</eta>
    </persona>
    <persona>
        <id>2</id>
        <nome>Pamela</nome>
        <cognome>Paolini</cognome>
        <eta>48</eta>
    </persona>
    <persona>
        <id>3</id>
        <nome>Francesco</nome>
        <cognome>Assisi</cognome>
        <eta>838</eta>
    </persona>
    <persona>
        <id>4</id>
        <nome>Mario</nome>
        <cognome>Mari</cognome>
        <eta>33</eta>
    </persona>
</nomi>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nomi>
    <persona id=”1”>
        <nome>Filippo</nome>
        <cognome>Brunelli</cognome>
        <eta>47</eta>
    </persona >
    <persona id=”2”>
        <nome>Pamela</nome>
        <cognome>Paolini</cognome>
        <eta>48</eta>
    </persona >
    <persona id=”3”>
        <nome>Francesco</nome>
        <cognome>Assisi</cognome>
        <eta>838</eta>
    </persona >
    <persona id=”4”>
        <nome>Mario</nome>
        <cognome>Mari</cognome>
        <eta>33</eta>
    </persona >
</nomi>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<nomi>
    <persona >
        <id>1</id>
        <nome>Filippo</nome>
        <cognome>Brunelli</cognome>
        <eta>47</eta>
        <indirizzi>
            <via>Via Vai </via>
            <n>12</n>
        </indirizzi>
    </persona >
    <persona >
        <id>2</id>
        <nome>Pamela</nome>
        <cognome>Paolini</cognome>
        <eta>48</eta>
        <indirizzi>
            <via>Via dai guai </via>
            <n>2</n>
        </indirizzi>
    </persona >
    <persona >
        <id>3</id>
        <nome>Francesco</nome>
        <cognome>Assisi</cognome>
        <eta>838</eta>
        <indirizzi>
            <via>Via cielo</via>
            <n>7</n>
        </indirizzi>
    </persona >

</utenti>
```

La root è il tag nomi, mentre i nodi sono persona; i sotto nodi sono quelli contenuti all'interno dei nodi come mostrato nell' ultimo esempio.

XML E .NET

C#, tramite il framework .Net, ha a disposizione diversi metodi per leggere e manipolare i dati XML, proprio come per i database relazionali e le principali classi sono XmlDocument e XmlReader.
XmlDocument, si comporta come un DataSet: legge l'intero contenuto XML in memoria e quindi consente di spostarti avanti e indietro come si preferisce, o di interrogare il documento utilizzando la tecnologia XPath.
XmlReader è invece simile alla classe DataReader: risulta più veloce e meno dispendioso in termini di memoria e consente di scorrere il contenuto XML un elemento alla volta, permettendo di esaminare il valore e quindi passare all'elemento successivo.
I principali Name Space messi a disposizione da .Net per utilizzare il formato XML sono: System.Xml, System.Xml.Schema, System.Xml.Serialization, System.Xml.XPath e System.Xml.Xsl per supportare le classi XML.

LEGGERE I DATI TRAMITE XMLDOCUMENT

Ecco un esempio tramite l'uso di XmlDocument:

```
using System;
using System.Xml;
using System.Xml.XPath;
    class xmluno {
        static void Main() {

            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load("c:\\testC\\esempiXML.xml");
            XmlNodeList lista = xmlDoc.SelectNodes("/nomi/persona");
            string nome;
            string cognome;
            string eta;

            foreach (XmlNode ls in lista)
                {
                        nome = ls["nome"].InnerText;
                        cognome = ls["cognome"].InnerText;
                        eta = ls["eta"].InnerText;

                        Console.WriteLine(nome + " " + cognome + " " + eta);
                }
            }
        }
```

NOTE: Solitamente un file XML (essendo un file di testo) difficilmente è di grandissime dimensioni e caricarlo in memoria non costituisce un grosso problema, per cui utilizzare XmlDocument non è necessariamente sinonimo di calo di prestazioni.

Abbiamo visto che si utilizza la classe XmlNodeList per selezionare un nodo del nostro documento XML.
XmlNodeList ha due proprietà fondamentali:
XmlNode.ChildNodes: Restituisce un XmlNodeList oggetto contenente tutti gli elementi figlio del nodo.
XmlNode.SelectNodes: Restituisce un XmlNodeList oggetto contenente una raccolta di nodi che corrispondono alla query XPath.
Nel nostro caso abbiamo utilizzato la seconda proprietà per ottenere la lista dei dati contenuti nel nodo persona, caricarli in memoria in memoria e poi, tramite un ciclo *foreach* mostrarli.
Se avessi modificato la riga nel seguente modo

```
XmlNodeList lista = xmlDoc.SelectNodes("/nomi/persona/indirizzi");
```

ed il ciclo in

```
                nome = ls["via"].InnerText;
                cognome = ls["n"].InnerText;
                Console.WriteLine(nome +" "+ cognome );
```

avrei avuto la lista degli indirizzi.
Per una lista completa delle proprietà visitare il sito microsoft: https://docs.microsoft.com/it-it/dotnet/api/system.xml.xmlnodelist?view=netframework-4.8

ESTRARRE UN DATO DA UN NODO

Il metodo più semplice per estrarre uno o più dati tramite XmlDocument è quello di inserire tra parentesi quadre nella stringa del nodo:

```
XmlNodeList lista = xmlDoc.SelectNodes("/nomi/persona[nome='Pamela']");
```

Un altro metodo è quello, di scorrere la lista fino a quando non si trova il dato cercato e, a quel punto, mostrarlo

```
foreach (XmlNode ls in lista)
            {
                if (ls["nome"].InnerText=="Pamela")
                {
                nome = ls["nome"].InnerText;
                cognome = ls["cognome"].InnerText;
                eta = ls["eta"].InnerText;
                Console.WriteLine(nome +" "+ cognome + " " +eta);
                }
            }
```

RIMUOVERE UN NODO

Per rimuovere un nodo da un file XML si usa la classe XmlNode; questa classe astratta che permette l'accesso ad un nodo del file Xml.
La classe XmlNode contiene una serie di proprietà tra le quali le principali sono:

.AppendChild(Node)	Aggiunge il nodo specificato alla fine dell'elenco dei nodi figlio del nodo corrente.
.Clone()	Crea un duplicato del nodo.
.InsertAfter(1° Nodo, 2° Nodo)	Inserisce il 2°nodo immediatamente dopo il 1° nodo
.InsertBefore(1° Nodo, 2° Nodo)	Inserisce 2° nodo immediatamente prima il 1° nodo
.RemoveAll()	Rimuove tutti gli elementi figlio e/o gli attributi del nodo corrente.
.RemoveChild(Nodo)	Rimuove il nodo figlio specificato.

```
using System;
using System.Xml;
using System.Xml.XPath;

    class xml2
    {
        static void Main()
        {
            string fileXml = @"c:\\testC\\esempiXML.xml";
            XmlDocument xmlDoc = new XmlDocument();
            XPathNavigator navigator = xmlDoc .CreateNavigator();
            xmlDoc.Load(fileXml);

            XmlNode Nd = xmlDoc.SelectSingleNode("/nomi/persona[nome='Nuovo']");

            xmlDoc.DocumentElement.RemoveChild(Nd);
            xmlDoc.Save(fileXml);
        }
    }
```

Come si vede è sufficiente caricare il nodo desiderato e utilizzare la proprietà **.RemoveChild**

AGGIUNGERE UN NODO

Per aggiungere un nodo utilizzerò sempre la classe XmlNode ma il metodo **.AppendChild()** dopo aver selezionato il nodo padre che mi interessa.

Un esempio può aiutare a capire meglio:

```
using System;
using System.Xml;
using System.Xml.XPath;

    class xml2
    {
        static void Main()
        {
            string fileXml = @"c:\\testC\\esempiXML.xml";
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load(fileXml);

            XmlNode node = xmlDoc.CreateNode(XmlNodeType.Element, "persona", null);

            XmlNode nodeId = xmlDoc.CreateElement("id");
            nodeId.InnerText = "5";
            XmlNode nodeNome = xmlDoc.CreateElement("nome");
            nodeNome.InnerText = "Nuovo";
            XmlNode nodeCognome = xmlDoc.CreateElement("cognome");
            nodeCognome.InnerText = "Nome";
            XmlNode nodeEta = xmlDoc.CreateElement("eta");
            nodeEta.InnerText = "100";

            //aggiungo i nodi
            node.AppendChild(nodeId);
            node.AppendChild(nodeNome);
            node.AppendChild(nodeCognome);
            node.AppendChild(nodeEta );

            //aggiung i nodi all'XML
            xmlDoc.DocumentElement.AppendChild(node);
            xmlDoc.Save(fileXml);

        }
```

Come si vede è sufficiente creare i nodi tramite

XmlNode *oggetto* = xmlDoc.**CreateElement(**"*nome_nuovo_nodo*"**)**;

All'interno del nodo padre ed assegnare un valore tramite .**InnerText**

E per finire aggiungo i nodi con il metodo .**AppendChild(***oggetto***)**

NOTE: Negli ultimi esempi visti abbiamo visto la proprietà **.Save(***nome_File***)** e la proprietà **.Load(***nome_File***)** della classe XmlDocument; queste proprietà servono, come dice il nome a salvare su disco e caricare in memoria un documento XML; **è Importante**, quando si usa la proprietà **.Save** che il file non sia in uso o sia aperto da un altro programma o utente in accesso esclusivo!

XPATH

XPath (XML Path Language) è un linguaggio standard che utilizza una sintassi non XML che è stato sviluppato per fornire un modo flessibile per scorrere un documento XML e trovarne i nodi ed i contenuti.
C# ed il frameworks .Net permette di utilizzare XPath all'interno dei propri progetti tramite lo spazio nomi:

```
using System.Xml.XPath;
```

Abbiamo visto l'utilizzo di questo spazio nomi negli esempi precedenti ma, poiché questo è solo un veloce tutorial generico, esamineremo solo le espressioni XPath di base e il loro significato.
Dato che XPath vede un documento XML come un albero di nodi e il suo scopo è proprio quello di individuare nodi e insiemi di nodi all'interno di questo albero una cosa importante da ricordare quando lo si utilizza è il contesto in cui ci si trova quando si tenta di utilizzare l'espressione.
Cosideriamo quindi che:
nodo radice: Il nodo radice dell'albero non è lo stesso elemento radice del documento infatti il nodo radice dell'albero contiene l'intero documento (compreso l'elemento radice; i commenti; le istruzioni di elaborazione presenti prima del tag iniziale o dopo il tag finale), quindi ha come figlio l'intero documento e non ha padre.
nodi elemento: hanno come padre il nodo radice o un altro nodo elemento. Come figli ha eventuali sotto nodi, testo e istruzioni contenute al suo interno. Gli attributi (<PERSONA ID="1">)non sono figli di un nodo elemento.
nodi attributo: hanno come padre un altro elemento, ma non si considera figlio di quell'elemento. Per accedervi ci vuole una richiesta di attributo.

Ad esempio /nomi/persona/nome applicato sul nostro file XML otterrà come risultato:

```
Element='<nome>Filippo</nome>'
Element='<nome>Pamela</nome>'
Element='<nome>Francesco</nome>'
Element='<nome>Mario</nome>'
```

Gli attributi degli elementi possono essere selezionati tramite XPath utilizzando il carattere @ seguito dal nome dell'attributo.
Posso quindi applicare, ad esempio, /nomi/persona[@id='3'] per selezionare il nodo specifico in:

```
  <persona id="1">
    <nome>Filippo</nome>
    <cognome>Brunelli</cognome>
    <eta>47</eta>
  </persona>

 <persona id="3">
    <nome>Mario</nome>
    <cognome>Mari</cognome>
    <eta>33</eta>
  </persona>

  <persona id="2">
    <nome>Francesco</nome>
    <cognome>Franchi</cognome>
    <eta>60</eta>
  </persona>
```

Una delle classi più importanti nel lo spazio nomi *System.Xml.XPath* è sicuramente **XPathNavigator** che fornisce un set di metodi usati per modificare nodi e valori in un documento XML ; per poter fare ciò è però necessario che l'oggetto XPathNavigator sia modificabile, quindi la proprietà CanEdit deve essere true.
La classe **XPathNodeIterator** fornisce i metodi per eseguire un'iterazione in un set di nodi creato come risultato di una query XPath (Usando il linguaggio XML Path) tramite l'utilizzo di un cursore di tipo forward-only di sola lettura e quindi utilizza il metodo **.MoveNext**; il set di nodi viene creato in base all'ordine con

cui è riportato nel documento, quindi la chiamata a questo metodo consente di spostarsi al nodo successivo nell'ordine del documento. XPathNodeIterator consente di ottenere informazioni dal nodo corrente tramite **.Current .Name** e **.Current.Value**.

Ecco un esempio di come estrarre dei dati tramite XPath:

```
using System;
using System.Xml;
using System.Xml.XPath;
    class xmluno {
        static void Main() {

            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load("c:\\testC\\esempiXML.xml");
            XPathNavigator navigator=xmlDoc.CreateNavigator();
            XPathNodeIterator nodo=navigator.Select("/nomi/persona/nome");

            while (nodo.MoveNext())
                {
                    Console.WriteLine(nodo.Current.Name +": " + nodo.Current.Value);
                }
            }
        }
```

Modificando la riga

```
XPathNodeIterator nodo=navigator.Select("/nomi/persona[nome='Filippo'] ");
```

otterremo tutti i dati del nodo persona il cui nome corrisponde a Filippo.

```
c:\testC>xml6
persona: 1FilippoBrunelli47Via Vai 12
```

Un'altra proprietà interessante della classe XPathNavigator è **.OuterXml** che ottiene il markup che rappresenta i tag di apertura e di chiusura del nodo corrente e dei relativi nodi figlio.

```
using System;
using System.Xml;
using System.Xml.XPath;
    class xmluno {
        static void Main() {
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load("c:\\testC\\esempiXML.xml");
            XPathNavigator navigator=xmlDoc.CreateNavigator();
            Console.WriteLine(navigator.OuterXml);
            }
        }
```

LEGGERE I DATI TRAMITE XMLREADER

Vediamo adesso come visualizzare un documento XML con l'utilizzo della classe XmlReader:

```
using System;
using System.Xml;
    class xml2
    {
        static void Main()
        {
            using (XmlReader reader = XmlReader.Create(@"c:\testC\esempiXML.xml"))
            {
                while (reader.Read())
                {
                    if (reader.IsStartElement())
                    {
                        switch (reader.Name.ToString())
                        {
                            case "nome":
                                Console.Write(reader.ReadString());
                                break;
                            case "cognome":
                                Console.Write(" " + reader.ReadString());
                                break;
                            case "eta":
                                Console.WriteLine(" " + reader.ReadString());
                                break;
                            case "via":
                                Console.Write("ABITA in " + reader.ReadString());
                                break;
                            case "n":
                                Console.WriteLine(" n.° " + reader.ReadString());
                                break;
                        }
                    }
                }
            }
        }
    }
```

Come si vede viene letto il flusso di dati riga per riga e a seconda del nodo viene intrapresa una relativa azione.
Notiamo la presenza dell'azione **.IsStartElement()** che serve a verificare se il nodo di contenuto corrente è un tag di inizio; se questo è vero iniziamo una serie si *switch* (vedere la prima parte del corso) per leggere i contenuti dei nodi e dei sotto nodi.
Anche se questo metodo sembra più macchinoso del precedente, in determinati casi, può offrire un controllo maggiore del risultato.

APPENDICI

APPENDICE SQL

SELECT
Serve ad estrarre una o più righe da una tabella di un database
Sintassi:
SELECT "*nome_di_colonna*" FROM "*nome_della_tabella*"
Esempi:

SELECT * FROM nomi WHERE nome=’Filippo’
SELECT nome, cognome FROM nomi
SELECT Nome, Cognome, eta FROM nomi GROUP BY eta ORDER BY eta
SELECT id, rif, prezzo, idtipo FROM catalogo WHERE idtipo IN (SELECT idsottotipo FROM sottotipo WHERE idtipo=6) ORDER BY id DESC

INSERT INTO
Serve per inserire una nuova riga di dati in una tabella
Sintassi:
INSERT INTO "*nome_della_tabella*" ("col.1", "col.2", ...) VALUES ("val.1", "val.2", ...)
Esempi:

INSERT INTO nomi (nome, cognome, eta) VALUES ('Filippo', 'Brunelli',47)

UPDATE
Serve per aggiornare i dati di una riga in una tabella
Sintassi:
UPDATE "*nome_della_tabella*" SET Col.1=Val1, Col.2=Val.2 WHERE *condizione*
Esempi:

UPDATE nomi SET nome='Mario', cognome='Bianchi', eta=20 WHERE id=5

DELETE
Serve per eliminare una o più righe da una tabella
Sintassi:
DELETE * FROM "*nome_della_tabella*" WHERE *condizione*
Esempi:

DELETE * FROM nomi
DELETE * FROM nomi WHERE id=6
DELETE * FROM nomi WHERE nome LIKE ‘%ippo%’

APPENDICE XML

Quando si lavora con i file XML bisogna ricordare che:

- Tutti i tag devono essere chiusi:

```
<tag1>
        …
</tag1>
```

- I tag devono essere correttamente innestati:

```
<tag1>
        <tag2>
                <tag3>
                   …
                </tag3>
        </tag2>
</tag1>
```

- Ogni documento XML deve avere uno ed un solo elemento radice:

```
<root>
        <tag1>
           …
        </tag1>
        <tag1>
          ….
        </tag1>
</root>
```

- Gli attributi devono sempre essere inclusi tra apici doppi:

```
<tag1 id=”25”>
  ….
</tag1>
```

- Tramite il costrutto CDATA è possibile introdurre del testo che non viene elaborato dal parser XML, ma venga semplicemente restituito come scritto:

```
<description>
        <![CDATA[<br/> Ma in fondo è questo il motivo per il quale scriviamo i post, la ricerca di visibilità, la caccia ai Like, e rimaniamo profondamente turbati se un nostro post non viene accolto con entusiasmo dato che la maggior parte dei nostri contatti sono affini a noi per idee. Proprio il fatto che i social network siano uno spazio pubblico ci dovrebbe portare a considerare quello che postiamo prima di pubblicarlo; così come un giornalista o un proprietario di un blog o di un sito è responsabile di ciò che pubblica anche l’accesso ai social network dovrebbe essere subordinato all’uso di una qualche forma deontologica. Questo non succede!<br/> <img src="https://www.filoweb.it/giornale/foto/S58UDO53R9F8L8C.jpg" style="width:100%; height:auto; max-width:300px;" /><br/>
        ]]>
</description>
```

- La prima riga deve indicare la versione di XML in uso e specifica la codifica UTF-8 per la corretta interpretazione dei dati

```
<?xml version="1.0" encoding="UTF-8"?>
```