Parole Chiave:
XML
RDBMS
Trasferimento Dati
QUERY
XQuery

Ringraziamenti

Desidero ringraziare i miei amici per avermi sostenuto durante gli anni di università.

Un ringraziamento particolare alla mia famiglia per non avere mai smesso di incoraggiarmi e per il sostegno economico fornitomi.

Inoltre desidero ringraziare per l'aiuto fornitomi durante lo svolgimento della tesi e per la costante disponibilità l'Ing. Daniele Bergonzini, Giovanni Zuffolini, l'Ing. Lorenzo Canali e la Professoressa Sonia Bergamaschi.

INDICE DEGLI ARGOMENTI TRATTATI

INTRODUZIONE

Con questo elaborato si vogliono analizzare delle metodologie per il trasferimento di dati tra database in formato XML in modo selettivo, e si vuole inoltre fornire uno strumento utile per effettuare delle interrogazioni su tali dati esportati in formato XML. Per operare sui database acquisendo ed esportando dati e per creare ed analizzare documenti XML si è utilizzata la tecnologia messa a disposizione dal Framework .NET. Questi strumenti sono stati utilizzati in applicazioni Webdatabase realizzate tramite la tecnologia ASP .NET, tali applicazioni forniscono servizi per reti civiche. In questo elaborato si è analizzato anche la possibilità di utilizzare documenti XML come piccoli database per applicazioni Webdatabase che richiedono una gestione dinamica dei dati, in tali documenti non sono implementate una gestione sicura degli accessi o il recupero dei dati danneggiati, ma per piccoli database con una mole di informazioni memorizzate ridotta e di cui viene fatto un uso limitato tale soluzione è accettabile. Tramite XQuery viene messo a disposizione dello sviluppatore un potente linguaggio di interrogazione paragonabile per potenza all'SQL 92. Si è poi analizzato come ottenere una interpretazione relazionale dei dati esportai da database in documenti XML e come eseguire delle query su tali documenti.

1 – XML E HTML A CONFRONTO

HTML (Hyper Text Markup Language) nacque come lingua franca per pubblicare informazioni ipertestuali nel World Wide Web. L'HTML è un formato non proprietario basato sullo standard SGML (Standard Generalized Markup Language) che permette di condividere in rete documenti di tipo molto specifico, dei semplici documenti di testo con qualche immagine e dei link ipertestuali. Negli ultimi anni le informazioni disponibili nel Web sono sempre più complesse e varie comprendendo anche video suoni e sempre più spesso consentono l'accesso e l'elaborazione di veri e propri database o parti di essi e le relazioni tra i dati rappresentati sono sempre più complicate. L'HTML utilizza i Tag d'apertura e chiusura come ad esempio <H1> e </H1> utilizzato per descrivere l'aspetto di quanto contenuto all'interno dei tag organizzando il documento in paragrafi intestazioni link tabelle liste e quant'altro, ma questo linguaggio era utilizzato più per una rappresentazione grafica dei dati che per una rappresentazione strutturata e semantica. La semplicità di questo linguaggio che lo ha reso così popolare è divenuta in seguito una debolezza che per gli usi che si fanno oggi del Web che richiedono sempre più informazioni sul significato che i dati testuali che sono forniti assumono, sui valori che essi possono avere e sulle operazioni che possono essere fatte su di essi hanno reso il solo HTML insufficiente. Nel 1998 da una specifica del World Wide Web Consortium noto come W3C è nato l'XML (Extensible Markup Language) un linguaggio che come l'HTML è un linguaggio di markup (ossia utilizza particolari etichette "Tag" per definire il contenuto di un documento), ma a differenza di esso l'XML è stato progettato per la descrizione dei dati; inoltre i Tag non sono prestabiliti dalle specifiche del linguaggio (come per l'HTML in cui il significato del tag è fisso ed esprime lo stesso concetto per tutti) ma ciascuno può crearsi i suoi, creandosi così il proprio linguaggio di Markup da cui la denominazione di "Extensible". Da questo risulta un codice molto più leggibile che si può meglio adattare alle esigenze del progettista e può meglio rappresentare la struttura dei dati rendendola più comprensibile oltre che alle applicazioni che dovranno trattare i dati stessi anche all'occhio umano. La struttura ad albero e l'assenza di regole di minimizzazione rendono più semplice la visualizzazione e l'analisi della struttura del documento rendendo possibile la visualizzazione del documento indipendentemente dal foglio di stile che vi si applica. Nell'XML inoltre viene utilizzato un Document Type Definition (DTD) o uno schema (XML Schema) che possono essere contenuti

all'interno del documento stesso o in un altro documento per definire e descrivere i dati del documento XML ed esplicitare le regole di composizione ed i rapporti possibili tra le varie parti del documento. In questo modo l'XML riesce ad essere un linguaggio autodescrittivo, diventando uno strumento molto utile per la comunicazione di dati tra sistemi diversi resa possibile dalla portabilità di questo linguaggio. L'interdipendenza tra l'XML e l'HTML inoltre facilita la conversione tra i formati dei dati interni ai database e i formati per il Web. Lo standard XML rispetto agli altri formati di dati per l'interscambio (che utilizzano strutture lineari) permette di sfruttare una struttura ad albero che rende più chiare le interdipendenze tra vari elementi. Un altro vantaggio infine è che l'XML consente di definire elementi ripetibili, permettendo strutture più flessibili e complesse d'altri formati di dati.

1.1 - ULTERIORI VANTAGGI DELL'XML

Tutte le volte che delle applicazioni si debbono scambiare dati, sorgono problemi di compatibilità, in quanto ogni applicazione ha le sue assunzioni in termini di caratteri speciali, di separatori, di ripetizione degli elementi, di rappresentazione di elementi vuoti e di elementi assenti (NULL). L'XML si propone così come un linguaggio intermedio per rappresentare dati anche complessi in forma indipendente dall'applicazione che li ha creati; inoltre L'XML può esprimere documenti strutturati in modo indipendente dalla loro destinazione finale, sia che essa sia una pagina Web o un database. Per questi e altri motivi l'XML è un linguaggio molto importante che viene sempre più utilizzato nei giorni nostri e su cui si basano sempre più linguaggi come XPath, XQuery e altri utili per la navigazione all'interno dei dati del documento. Un utilizzo molto comune e utile dell'XML è ad esempio lo scambio di dati tra due database diversi rendendo possibile la comprensione della sintassi e della struttura dei dati del database da cui si esporta al database in cui sono importati i dati, cosa resa possibile dalla generalità dall'indipendenza e dalla capacità di autodescrivere i suoi elementi tipica dell'XML.

2 - BREVE PANORAMICA SULL'XML

I documenti XML sono costituiti da unità di memoria dette entità, che contengono sia dati analizzati che non analizzati. I dati analizzati sono costituiti da caratteri, alcuni dei quali formano i character data, e alcuni i markup. I markup codificano una descrizione dell'organizzazione di memorizzazione e della struttura logica del documento. XML fornisce un meccanismo per imporre dei vincoli sull'organizzazione di memorizzazione e sulla struttura logica. Ciascun documento XML ha una struttura logica e fisica. Fisicamente, il documento è composto di unità dette entità. Un'entità può indirizzare altre entità per includerle nel documento. Un documento inizia con una "radice root" o entità documento. Logicamente, il documento è composto di dichiarazioni, elementi, commenti, riferimenti a caratteri, e istruzioni di elaborazione, ciascuno dei quali è indicato nel documento da espliciti markup. La struttura logica e quella fisica devono annidarsi propriamente rispettando le regole di well formed. Un documento può dirsi ben formato se contiene uno o più elementi, se c'è un solo elemento root che non deve essere contenuto in altri elementi, inoltre gli elementi delimitati dal tag di inizio e dal tag di fine si annidano uno dentro l'altro e ad ogni tag di apertura deve necessariamente corrispondere un tag di chiusura a meno che l'elemento non sia vuoto, in questo caso può utilizzare un simbolo speciale <TagVuoto/> oppure i tag di apertura e chiusura con niente dentro <TagVuoto></TagVuoto>. Gli elementi si dividono in quattro tipi, questi sono:

- Element content: un elemento che contiene soltanto altri elementi.
- Mixed content: un elemento che contiene sia testo che altri elementi.
- Simple content: un elemento che contiene soltanto testo.
- Empty content: un elemento vuoto.

Gli attributi sono invece specificati nel tag d'apertura di un elemento e i loro valori sono racchiusi da virgolette, essi sono principalmente utilizzati per rendere più specifico il comportamento di default di un elemento anche se a volte sono utilizzati per rappresentare dati. La dichiarazione XML si trova nella prima riga del documento ed è espressa mediante <? Contenuto dichiarazione ?> il contenuto della dichiarazione indica la versione XML in cui è stato scritto il documento ed il set di caratteri utilizzato come ad esempio la codifica UTF-8. I Commenti possono apparire ovunque in un documento fuorché all'interno degli altri markup e sono espressi in questo modo <!-- Commento --!> per una corretta interpretazione dell'XML da parte di un parser XML la stringa "--" non deve essere presente all'interno del testo del commento. Inoltre tutte le entità sono definite e hanno un loro nome e valore.

Ecco un esempio di documento XML Ben Formato:

In questo esempio è presente una dichiarazione XML seguita da un commento e dall'elemento Root del documento, in esso sono innestati altri elementi di vario tipo, tra cui l'elemento Book contenente un attributo. Come si può notare l'XML riesce a descrivere strutture dati in modo molto esplicativo e chiaro anche all'occhio umano.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Rappresentazione di un catalogo di libri --!>
 <Catalogo>
        <Book Isbn="IJK7-NB" >
                <Titolo>Dune</Titolo>
                <Autore>
                        <Nome>Frank</Nome>
                        <Cognome>Herbert</Cognome>
                </Autore>
                <Editore>Sperling Kupfer</Editore>
                <Prezzo/>
                <Genere>Fantascienza</Genere>
        </Book>
</Catalogo>
```

Un documento XML per essere valido oltre ad essere Well Formed deve essere associato ad un DTD (DOCUMENT TYPE DEFINITION) o ad uno Schema (XML SCHEMA). Questi sono gli strumenti che descrivono la grammatica del documento dichiarando ciascun elemento usato nel documento XML e definendo una serie di regole che specificano tutti gli aspetti architetturali e semantici da verificare nel documento così che le applicazioni siano in grado di validarne la struttura; questi documenti si possono trovare nel documento XML stesso o in un altro file separato. Il DTD è uno strumento antiquato con limiti inerenti la rappresentazione della struttura dei dati, inoltre esso è poco flessibili nella descrizione dei mixed content e non permette di definire vincoli sugli elementi di testo, perciò in seguito si utilizzerà l' XML Schema (che è anche supportato dal Framework di .NET) per descrivere e validare i documenti XML.
(Per approfondimenti riguardo al linguaggio XML vedi (4) dispense del corso di "Tecnologia database" riguardanti l'XML "Teoria_XMLNEW.pdf").

2.1 - NAMESPACE IN XML

Nell'XML i documenti si mescolano e si fondono tra loro in maniera complessa, il documento stesso potrebbe avere alcuni elementi definiti in uno schema ed altri in un altro, si deve riuscire a conciliare la presenza in differenti documenti di elementi che pur essendo diversi hanno lo stesso nome o di elementi diversi ma con lo stesso nome nello stesso documento; per risolvere questo problema sono stati introdotti i namespaces ovvero spazi dei nomi. Gli elementi, attributi del documento XML sono preceduti da un prefisso che specifica l'origine del nome, questo è separato attraverso il carattere ':' dal nome dell'elemento o attributo. L'attributo "xmlns" è utilizzato per identificare i prefissi utilizzati dai namespace nel documento. Il valore dell'attributo "xmlns" è un URI (Universal Resource Identifier) che è una stringa che definisce in modo univoco il nome di una risorsa sul Web, questo ha un valore puramente informativo e serve per identificare l'elemento/attributo in un contesto generale. Gli elementi e attributi identificati da un prefisso di namespace e da un nome locale sono detti qualificati e sono univocamente identificabili dato che ogni namespace ha un prefisso diverso. Attraverso gli attributi elementFormDefault e attributeFormDefault contenuti all'interno dell'elemento schema di un XMLSchema si può stabilire che tutti gli elementi o gli attributi debbano necessariamente essere qualificati impostando il valore dei suddetti attributi come "qualified" mentre se non si desidera che sia necessario qualificare tutti gli elementi/attributi del documento si può impostare il valore dei suddetti attributi come "unqualified". Se nella dichiarazione xmlns non si pone nessun prefisso si assume quel namespace come namespace di default e non lo si può applicare agli attributi.

Esempio di utilizzo dei namespace:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!-- Rappresentazione di un catalogo di libri --!>
 <myCatalog:Catalogo xmlns: myCatalog =”http://www.bol.it/books”>
        < myCatalog:Book Isbn=”IJK7-NB” >
                < myCatalog:Titolo>Dune</ myCatalog:Titolo>
                < myCatalog:Autore>
                        < myCatalog:Nome>Frank</ myCatalog:Nome>
                        < myCatalog:Cognome>Herbert</ myCatalog:Cognome>
                </ myCatalog:Autore>
                < myCatalog:Editore>Sperling Kupfer</ myCatalog:Editore>
                < myCatalog:Prezzo valuta=”Euro”>10</myCatalog:Prezzo>
                < myCatalog:Genere>Fantascienza</ myCatalog:Genere>
        </ myCatalog:Book>
</ myCatalog:Catalogo>
```

3 - XSD L'XML SCHEMA DEFINITION

XML Schema è descritto da una specifica del W3C del 2001 ed è composto da tre parti, la parte 0 è un'introduzione allo schema, la parte 1 descrive la struttura dello schema mentre la parte 2 descrive il modello dei dati e i meccanismi di estensione dei tipi. Un documento XML Schema è contenuto dentro l'elemento <schema> questo a sua volta contiene vari elementi tra cui element, attribute, complexType, simpleType, tramite i quali vengono descritte la struttura e i tipi di elementi e attributi dei documenti XML.
Per ulteriori chiarimenti vedere il sito del W3C sull'XmlSchema
(vedi (5) e (6) "http://www.w3.org/TR/xmlschema-*x*" con x = {0,1,2}).

Ogni elemento dello schema è associato al prefisso xs: che è a sua volta associato tramite namespace all'URI che identifica gli elementi dello schema XML con questa dichiarazione xmlns:xs=http://www.w3.org/2001/XMLSchema. Nello schema si può trovare anche il prefisso msdata: che si riferisce alla dichiarazione xmlns:msdata="urn:schemas-microsoft-com:xml-msdata".

3.1 - XML SCHEMA DEFINIZIONI, TIPI COMPLESSI SEMPLICI ED ATTRIBUTI

I'XMLSchema divide gli elementi in tipi complessi **complexType** e tipi semplici **simpleType**, i tipi semplici possono solo contenere testo e non possono avere ne attributi né elementi figli, inoltre sono limitati nel tipo di testo che possono contenere dai tipi definiti dall'XMLSchema; mentre i tipi complessi possono avere sia attributi che elementi figli. Gli elementi sono definiti mediante l'elemento **element** mentre gli attributi sono identificati dall'elemento **attribute**.

```
<xs:complexType name="autor">
    <xs:sequence>
        <xs:element name="Nome" type="xs:string"/>
        <xs:element name="Cognome" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="book">
    <xs:sequence>
        <xs:element name="Titolo" type="xs:string"/>
        <xs:element name="Autore" type="autor"/>
        <xs:element name="Catalogazione" type="xs:string"/>
        <xs:element name="Personaggio" type="xs:string"/>
        <xs:element name="Prezzo" type="xs:int"/>
        <xs:element name="DataPubblicazione" type="xs:dateTime"/>
        <xs:element name="Editore" type="xs:string"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID"/>
</xs:complexType>

<xs:element name="Book" type="book"/>
```

L'elemento **sequence** contenuto nell'elemento complexType racchiude al suo interno una sequenza di elementi che dovranno necessariamente essere tutti presenti (a meno che non sia specificato diversamente attraverso l'attributo minOccurs in seguito spiegato) e nello specifico ordine in cui sono indicati nella definizione del tipo complesso, in tutti gli elementi indicati come di tipo "book". Il nome di un elemento è identificato dal valore dell'attributo **name** dell'elemento stesso, mentre il tipo dell'elemento è identificato dal valore dell'attributo **type**, questo può essere uno dei tipi semplici definiti da XMLSchema oppure un tipo definito all'interno dello schema come ad esempio "autor".

Gli attributi possono comparire in un elemento una volta o nessuna, non un certo numero di volte inoltre tramite l'attributo **use** dell'elemento attribute possono essere dichiarati come richiesti, opzionali o proibiti (required, optional, prohibited).

```
<xs:attribute name="id" type="xs:ID" use="required"/>
```

Attraverso gli attributi **minOccurs** e **maxOccurs** di un xs:element è possibile specificare il numero minimo e massimo di istanze dell'elemento che possono essere presenti in quel punto del documento se a maxOccurs viene dato il valore "unbounded" significa che l'elemento in questione può apparire in quel particolare contesto un numero indefinito di volte; se min/maxOccurs non sono dichiarati all'interno dell'elemento di default valgono uno, attribuendo a minOccurs il valore 0 si rende l'elemento opzionale.

```
<xs:element name="Personaggio" type="xs:string" minOccurs="0" maxOccurs="unbounded"
default="protagonista"/>
```

Questo è molto più flessibile dei **?**,***** e **+** che si usavano nel DTD.

Tramite **default** è possibile attribuire un valore di default agli elementi e agli attributi, nel caso in cui non fosse attribuito nessun valore ad un attributo con un valore di default indicato questo assumerebbe come valore il valore di default. Per gli elementi, se non sono presenti nel documento XML non vi saranno inseriti, anche se l'attributo default è impostato ed ha un valore, mentre se sono presenti ma sono vuoti assumeranno il valore di default specificato.

L'attributo **fixed** è utilizzato nelle dichiarazioni di elementi ed attributi per assicurarsi che se gli elementi o attributi appariranno nel documento avranno il valore definito con fixed, in questo modo si riesce a creare delle costanti.

```
<xs:attribute name="Valuta" type="xs:string" use="required" fixed="Euro"/>
```

Settando l'attributo **mixed** col valore "true" si riesce a rappresentare un elemento Mixed content che conterrà al suo interno sia altri elementi e attributi che testo.

Schema:

```
<xs:complexType name="PersonaSposata">
  <xs:sequence>
    <xs:element name="FAMIGLIA">
     <xs:complexType mixed="true">
       <xs:sequence>
          <xs:element name="CONIUGE"  type="xsd:string"/>
          <xs:element name="FIGLIO" type="xsd:string" minOccurs="0"/>
```

```
            </xs:sequence>
        </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:element name="Padre" type="PersonaSposata"/>
```

XML:

```
<Padre>
  <FAMIGLIA>
       Giovanni Rossi <CONIUGE>Maria Bianchi</CONIUGE>
       <FIGLIO>Carlo Rossi</FIGLIO>Padre
   </FAMIGLIA>
 </Padre>
```

Negli schemi possono essere dichiarati diversi elementi e attributi definiti come tipi semplici **simpleType**, XMLSchema ha implementato diversi tipi semplici che possono essere utilizzati così come definiti dalle specifiche W3C oppure gli si possono applicare delle restrizioni; qui è mostrata una tabella che rappresenta tutti i tipi semplici e i relativi valori che possono assumere.

Tabella 1: rappresenta i tipi semplici definiti da XMLSchema

Simple Type	Esempi
string	Il solito Hello World
normalizedString	Esempio di stringa, non contiene tabs, carriage returns o linefeeds
token	p1 p2, ss123 45 6789, _92, red, green, NT Decl, seventeenp1, p2, 123 45 6789, ^*&^*&_92, red green blue, NT-Decl
byte	-1, 126
unsignedByte	0, 126
base64Binary	GpM7
hexBinary	0FB7
integer	-126789, -1, 0, 1, 126789
positiveInteger	1, 126789
negativeInteger	-126789, -1
nonNegativeInteger	0, 1, 126789
nonPositiveInteger	-126789, -1, 0
int	-1, 126789675
unsignedInt	0, 1267896754
long	-1, 12678967543233
unsignedLong	0, 12678967543233
short	-1, 12678
unsignedShort	0, 12678

decimal	-1.23, 0, 123.4, 1000.00
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN
boolean	true, false 1, 0
time	13:20:00.000, 13:20:00.000-05:00
dateTime	1999-05-31T13:20:00.000-05:00
duration	P1Y2M3DT10H30M12.3S
date	1999-05-31
gMonth	--05--
gYear	1999
gYearMonth	1999-02
gDay	---31
gMonthDay	--05-31
Name	shipTo
QName	po:USAddress
NCName	USAddress
anyURI	http://www.example.com/, http://www.example.com/doc.html#ID5
language	en-GB, en-US, fr ,it
ID	p1, p2, ss124-45-6789, _92, red, green, NT-Decl, seventeen
IDREF	p1, p2, ss124-45-6789
IDREFS	Una lista di nomi XML separati da spazi bianchi già usati come valori di attributi di tipo ID
ENTITY	Qualsiasi nome XML dichiarato come unparsed entity in un DTD
ENTITIES	Una lista di entity
NOTATION	GIF, jpeg, TIF, pdf, TeX
NMTOKEN	US, Brasile
NMTOKENS	US UK, Brasile Canada Messico (una lista di name token separati da uno spazio bianco)

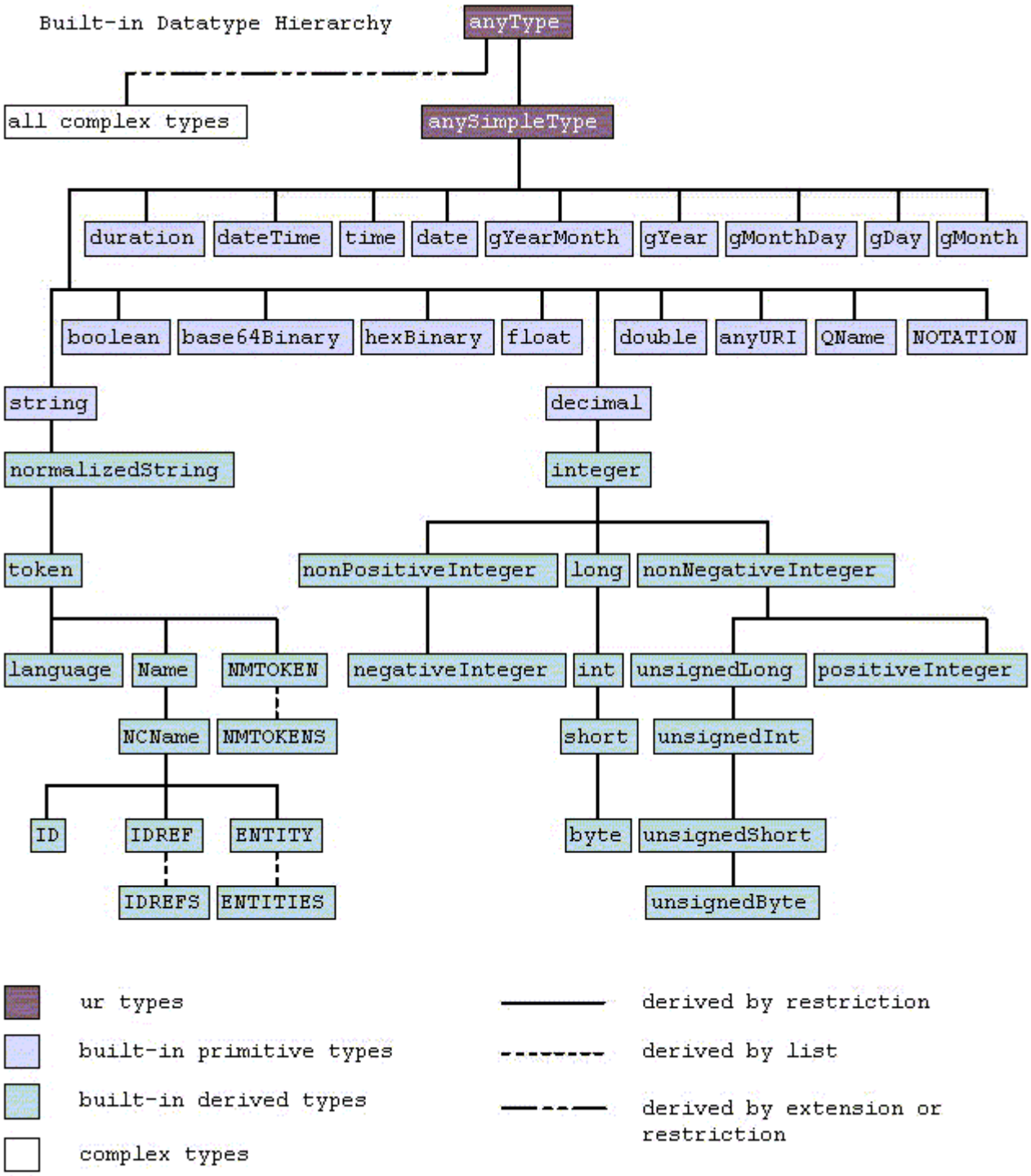

figura 1 Gerarchia dei tipi semplici definiti da XML Schema

Si possono definire dei nuovi tipi semplici derivandoli da tipi semplici già esistenti per esempio facendo delle restrizioni sui valori che possono assumere i tipi definiti nella Tabella 1, i nuovi tipi avranno un dominio di valori che è un sottoinsieme di quello del tipo già definito. Per fare questo si deve definire un nuovo tipo tramite simpleType, dopodiché tramite l'elemento **restriction** si identifica il tipo base da cui derivare e tramite delle "facets" si applicano i vincoli sul range di valori che può assumere il nuovo tipo. Le facets non possono essere applicate a tutti i tipi, ecco di seguito una lista di alcune facets:

length: definisce il numero di unità di lunghezza, per esempio il numero dei caratteri di una stringa, ponendo l'attributo fixed='true' si imposta la lunghezza del nuovo tipo in modo che non possa avere un valore diverso da quello specificato dall'attributo value.

```
<xs:simpleType name="codiceProdotto">
  <xs:restriction base="xs:string">
    <xs:length value="5" fixed="true"/>
  </xs:restriction>
</xs:simpleType>
```

minLength e **maxLength** rappresentano rispettivamente il numero minimo e massimo di caratteri che i nuovi tipi potranno avere.

pattern: definisce una espressione regolare che il valore dell'elemento del nuovo tipo deve soddisfare.

Un esempio è il tipo "Moneta" che tramite l'utilizzo dei pattern rappresenta una stringa che deve contenere un simbolo di moneta uno o più caratteri numerici e una parte decimale opzionale che se presente consiste in un punto e due caratteri decimali (compresi tra 0 e 9).

```
<xs:simpleType name="Moneta">
 <xs:restriction base="xs:string">
   <xs:pattern value="\p{Sc}\p{Nd}+(\.\p{Nd}\p{Nd})?"/>
 <xs:restriction>
<xs:simpleType>
```

minExclusive, **maxExclusive** rappresentano il valore minimo e massimo esclusivo per l'elemento del nuovo tipo, servono per implementare vincoli di > e <.

minInclusive e **maxInclusive** rappresentano il valore minimo e massimo inclusivo per il nuovo tipo, servono per rappresentare vincoli di >= o <=.

```
<xs:simpleType name="numIntPos">
 <xs:restriction base="integer"/>
   <xs:minInclusive value="1"/>
   <xs:maxExclusive value="101"/>
 </xs:restriction>
</xs:simpleType>
```

Il tipo "numIntPos" rappresenta un numero intero compreso tra 1 e 100 inclusi.

totalDigits rappresenta il numero massimo di cifre in un valore del tipo a cui è stato applicato il vincolo suddetto, se l'attributo fixed ="true" allora il numero di cifre dovrà necessariamente essere uguale al numero indicato dall'attributo value.

fractionDigits è come totalDigits solo che si riferisce al numero di cifre della parte decimale.

whiteSpace indica il comportamento che sarà adottato nei confronti degli spazi bianchi, quest'elemento può assumere tre diversi valori preserve, replace, collapse che rispettivamente identificano diversi comportamenti. Con preserve non è attuata nessuna normalizzazione e i valori non sono cambiati, tramite replace tutte le occorrenze dei caratteri tab, line feed e carriage return

vengono sostituiti con uno spazio, mentre collapse oltre a fare quello che fa replace sostituisce sequenze contigue di caratteri spazio con un singolo carattere spazio.

enumeration definisce una lista che specifica i valori che il nuovo tipo a cui è applicata la restrizione può assumere, in pratica si definisce lo spazio dei valori per il tipo. Nell'esempio seguente il tipo "CasaDiscografica " potrà assumere soltanto uno dei cinque valori specificati tramite enumeration.

```
<xs:simpleType name="CasaDiscografica">
   <xs:restriction base="xs:string">
     <xs:enumeration value="Warner-Elektra-Atlantic"/>
     <xs:enumeration value="Universal Music Group"/>
     <xs:enumeration value="Sony Music Entertainment, Inc."/>
     <xs:enumeration value="Capitol Records, Inc."/>
     <xs:enumeration value="BMG Music"/>
   </xs:restriction>
 </xs:simpleType>
```

Si rimanda alle Appendici per la lista completa dei tipi semplici di XML Schema e delle facets ad essi applicablili.

In addizione ai tipi atomici in XML Schema sono definiti anche i tipi **list**, questi sono formati da sequenze di tipi atomici e di conseguenza gli elementi di questa sequenza sono essi stessi dei tipi e possono essere considerati singolarmente. Il tipo degli elementi della lista viene definito dal valore dato all'attributo itemType, gli elementi della lista nel documento XML sono separati da uno spazio bianco.

```
<xs:simpleType name="EruoStateList">
    <xs:list itemType="EuroState"/>
</xs:simpleType>
```

I tipi **union** fanno sì che l'insieme dei valori leciti per il nuovo tipo sia dato dall'unione dei valori leciti dei tipi raggruppati dalla union.

```
<xs:element name="Costo" type="CostoElement">
<xs:simpleType name="CostoElement">
    <xs:union>
        <xs:simpleType>
            <xs:restriction base="xsd:decimal">
              <xs:minExclusive value="0.0"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType>
            <xs:restriction base="xsd:string">
                <xs:enumeration value="gratuito"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>
```

L’elemento Costo potrà avere come valore un decimale maggiore di 0.0 oppure la stringa gratuito.

Un tipo semplice come detto precedentemente non può contenere attributi, perciò per definire un tipo semplice con un attributo si deve definire un tipo complesso che contenga la dichiarazione dell’attributo e il tipo semplice. Per fare questo si utilizza un’estensione **extension** del tipo base semplice dentro un elemento di contenuto semplice **simpleContent**.

```
<xs:element name="PrezzoInternazionale">
 <xs:complexType>
   <xs:simpleContent>
      <xs:extension base="xsd:decimal">
         <xs:attribute name="valuta" type="xsd:string"/>
      </xs:extension>
   </xs:simpleContent>
 </xs:complexType>
</xs:element>
```

In XML Schema esiste un posto specifico in cui inserire note ed istruzioni, questo è l’elemento **annotation**, questo elemento può contenere due diversi elementi che sono l’elemento **documentation** realizzato per inserire nel documento commenti che possano essere letti da esseri umani e l’elemento **appInfo** pensato per inserire istruzioni che possano essere interpretate da applicazioni. Tramite l’attributo “lang” si può definire la lingua in cui è scritto il commento.

```
<xs:element name="PrezzoInternazionale">
 <xs:annotation>
   <xs:documentation xml:lang="it">
         dichiarazione di elemento con tipo anonimo
    </xs:documentation>
 </xs:annotation>
 <xs:complexType>
   <xs:annotation>
   <xs:documentation xml:lang="it">
         tipo anonimo vuoto con due attributi
   </xs:documentation>
    </xs:annotation>
    <xs:complexContent>
       <xs:restriction base="xs:anyType">
         <xs:attribute name="valuta" type="xs:string"/>
         <xs:attribute name="valore"    type="xs:decimal"/>
       </xs:restriction>
   </xs:complexContent>
 </xs:complexType>
</xs:element>
```

3.2 - GROUPING

XML Schema ha fornito dei vincoli applicabili a gruppi di elementi che appaiono nei modelli di contenuto “content model”. Questi possono essere vincoli sull’ordine in cui appariranno gli elementi, sul fatto che vi possa essere un elemento soltanto di una lista di elementi che può apparire o sul fatto che debbano essere presenti tutti gli elementi di una lista indipendentemente dall’ordine in cui appaiono, per fare ciò si utilizzano gli elementi **sequence**, **choice** e **all**.

sequence: il raggruppamento prevede che tutti gli elementi siano presenti esattamente una volta e nella sequenza specificata da sequence.

choice: il raggruppamento prevede che soltanto uno degli elementi del raggruppamento possa apparire, quale elemento sia non è specificato, corrisponde al | del DTD.

all: il raggruppamento richiede che tutti gli elementi del gruppo devono apparire una volta, ma l'ordine in cui appariranno non è importante, corrisponde al & del DTD.

Per tutti gli elementi dei raggruppamenti si può specificare ulteriormente la ripetibilità e la facoltatività attraverso l'utilizzo di minOccurs e di maxOccurs che possono essere applicati oltre che agli elementi stessi dei raggruppamenti anche alle strutture all, sequence e choice.

```
<xs:element name="Prodotto" maxOccurs="unbounded">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Titolo"  type="xs:string"/>
         <xs:element name="Quantita" type="xs:string"/>
         <xs:choice>
            <xs:element name="Autore" type="xs:string"/>
            <xs:element name="Regista" type="xs:string"/>
            <xs:element name="GruppoMusicale" type="xs:string"/>
         </xs:choice>
         <xs:element name="Prezzo">
            <xs:all>
               <xs:element name="Costo">
                  <xs:simpleType>
                     <xs:restriction base="xsd:decimal">
                        <xs:minExclusive value="0.0"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:element>
               <xs:element name="Valuta" type="xs:string"/>
            </xs:all>
         </xs:element>
      </xs:sequence>
      <xs:attribute name="Codice" type="xs:string" use="required"/>
   </xs:complexType>
</xs:element>
```

Gli elementi "Prodotto" sono costituiti da una sequenza d'elementi costituita da Titolo, Quantita uno tra gli elementi Autore, Regista, GruppoMusicale, e un elemento Prezzo costituito a sua volta dagli elementi Costo e Valuta (per cui non è importante l'ordine di apparizione). I tipi definiti possono avere come si è visto strutture complesse che meglio si adattano alla descrizione di certi elementi, in quest'esempio si è riusciti ad identificare un prodotto di un negozio che vende libri, film e cd musicali.

3.3 - VALORI NULLI

In generale l'assenza di un elemento in XML non ha un particolare significato, può indicare che l'informazione è sconosciuta o non applicabile, oppure l'elemento può essere assente per qualunque altra ragione. Può essere utile rappresentare un elemento NULL risultante da un'operazione su un database relazionale con un elemento presente invece che tramite l'assenza dell'elemento nullo. Questi casi si possono rappresentare attraverso l'utilizzo del **nil** in XML Schema che permette la presenza o meno di elementi con valori nulli, questo è fatto utilizzando il segnale di out of band, in pratica non vi è nessun elemento con valore null, ma la segnalazione che l'elemento è nullo si ottiene tramite l'attributo nil="true".
Per rappresentare il fatto che un elemento possa avere un valore nil nelle istanze del documento si deve settare l'attributo **nillable**="true" nella rappresentazione dell'elemento nello schema.

Si definisce l'elemento che può avere valori nulli nello schema in questo modo:

```
<xs:element name="Data" type="xs:date" nillable="true"/>
```

dopodiché nell'XML gli elementi nulli saranno rappresentati in questo modo:

```
<Data xsi:nil="true"></Data>
```

Con il prefisso xsi che viene associato al namespace
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" nel file XML contenente l'elemento "Data".

3.4 - VINCOLI DI UNICITA'

XML Schema dà la possibilità di indicare il valore di un attributo od un elemento come unico. Per indicare che il valore di un certo elemento o attributo è unico si utilizza l'elemento **unique**. Prima si utilizza l'elemento **selector** per selezionare un set di elementi del file XML da rendere unici e poi tramite l'elemento **field** si specificano gli attributi o elementi relativi a tutti gli elementi precedentemente selezionati che debbono avere un valore unico. I percorsi assegnati agli elementi selector e field tramite i quali sono selezionati attributi ed elementi sono dei percorsi XPath; questa è una raccomandazione W3C del 16/11/1999 e specifica i meccanismi per indicare percorsi all'interno di file XML in modo da individuare parti del documento stesso.

```
<xs:unique name="Constraint1">
    <xs:selector xpath=".//Prodotto"/>
    <xs:field xpath="Titolo"/>
    <xs:field xpath="@Codice"/>
</xs:unique>
```

Grazie a quest'espressione si dichiara che le combinazioni di elemento Titolo e attributo Codice entrambi figli degli elementi Prodotto debbano avere valori unici, così da rendere unici ed univocamente identificabili tutti gli elementi Prodotto all'interno del documento XML. La dichiarazione dei campi unici deve essere fatta all'interno dell'elemento root o all'interno dell'elemento genitore degli elementi figli che devono essere unici.

3.5 - CHIAVI E LORO RIFERIMENTI

I costrutti **key** e **keyref** sono applicati utilizzando pressappoco la stessa sintassi di unique, il loro uso fa sì che non sia possibile definire gli elementi o attributi indicati mediante questi costrutti come nil cioè con valori nulli, ed inoltre li rende unici; inoltre il nome associato alla chiave la rende referenziabile altrove.

```
<xs:keyref name="ProdottoVenduto_Constraint1" refer="Constraint1"
msdata:ConstraintName="Constraint1" msdata:ConstraintOnly="true">
    <xs:selector xpath=".//ProdottoVenduto"/>
    <xs:field xpath="@CodProd"/>
</xs:keyref>
```

Tramite keyref si indica che il valore dell'elemento o attributo selezionato tramite field non deve per forza essere unico, ma deve necessariamente esistere un elemento o attributo identificato dal vincolo indicato da **refer** con lo stesso valore.

4 - INTRODUZIONE AL FRAMEWORK .NET E AL LINGUAGGIO C#

4.1 - IL FRAMEWORK .NET

Il Framework .NET è nato per semplificare lo sviluppo di applicazioni nell'ambiente altamente distribuito di Internet, esso fornisce un ambiente di programmazione orientato agli oggetti coerente, sia che il codice sia memorizzato ed eseguito localmente, eseguito localmente ma distribuito su Internet o eseguito in modalità remota. Le componenti principali nel Framework .NET sono CLR (Common Language Runtime) e la libreria di classi .NET Framework. Il compito di CLR è quello di gestire il codice in fase di esecuzione assicurando le funzionalità di base come la gestione della memoria, dei thread, l'esecuzione e la verifica della protezione del codice, la compilazione, la gestione dei servizi remoti assicurando al contempo protezione ed efficienza. Il codice destinato al runtime viene chiamato codice gestito .NET Framework può essere contenuto da componenti non gestiti che caricano Common Language Runtime nei processi e avviano l'esecuzione del codice gestito, creando così un ambiente software in grado di sfruttare funzionalità gestite e non gestite, inoltre supporta lo sviluppo di host di runtime di altri produttori.
L'ambiente gestito di runtime gestisce automaticamente il layout di oggetti e i riferimenti ad oggetti che vengono rilasciati quando non sono più utilizzati; tramite questa gestione della memoria vengono risolti i problemi delle perdite di memoria e dei riferimenti a memoria non validi.
Per quel che riguarda la protezione, ai componenti gestiti sono assegnati vari gradi di attendibilità così da consentire o no l'esecuzione di certe operazioni. L'efficienza del codice è attivata dal runtime tramite l'implementazione di una struttura di verifica di tipi e codice denominata CTS (Common Type System) che assicura che tutto il codice gestito sia autodescrittivo.
Il formato del codice compilato nella tecnologia .NET si chiama IL (Intermediate Language) ed è un linguaggio intermedio autodescrittivo formato da una serie di istruzioni indipendenti dalla CPU facilmente convertibile in codice nativo; assieme a questo al momento della compilazione sono generati anche metadati. I metadati descrivono i tipi contenuti nel codice, inclusa la definizione di ciascun tipo, i membri a cui fa riferimento il codice e altri dati utilizzati in fase di esecuzione; il codice IL e i metadati sono contenuti in un file eseguibile trasportabile chiamato PE (Portable Executable). Grazie a questo formato di file, che supporta sia IL che codice nativo che metadati, il sistema operativo è in grado di riconoscere immagini in Common Language Runtime, inoltre la presenza di metadati nel file, assieme al IL, consente al codice di autodescriversi.
Prima di poter eseguire il codice, è necessario convertire IL in codice specifico della CPU, difatti il codice gestito non è mai interpretato dal runtime, ma attraverso il compilatore JIT (Just In Tyme) viene eseguito nel linguaggio macchina nativo del sistema su cui è in esecuzione. Contemporaneamente il gestore di memoria impedisce la frammentazione della memoria e aumenta la rintracciabilità dei riferimenti della memoria per migliorare ulteriormente le prestazioni.
Il runtime può essere infine contenuto da applicazioni lato server a elevate prestazioni consentendo di utilizzare il codice gestito per scrivere programmi aziendali personalizzati continuando a sfruttare le maggiori prestazioni dei server aziendali che supportano l'hosting del runtime.

La libreria di classi è un insieme completo orientato agli oggetti di tipi riutilizzabili che possono essere impiegati nello sviluppo delle applicazioni. .NET Framework fornisce inoltre un insieme di classi e di strumenti che facilitano lo sviluppo e l'utilizzo di applicazioni di servizi Web XML. I servizi Web XML sono generati su standard quali SOAP (protocollo di chiamata a procedura remota), XML (formato dati estensibile) e WSDL (Web Services Description Language, linguaggio di descrizione dei servizi Web) garantendo l'interoperabilità tra diverse soluzioni. Per lo sviluppo e

la pubblicazione di un servizio Web XML .NET Framework fornisce un insieme di classi conformi a tutti gli standard di comunicazione sottostanti come SOAP, WSDL e XML. Mediante queste classi, è possibile individuare la logica del servizio, senza preoccuparsi delle infrastrutture di comunicazione richieste dallo sviluppo di software distribuiti.
In conclusione i tipi .NET Framework consentono di effettuare le più comuni attività di programmazione come la gestione di stringhe, la raccolta di dati, la connettività al database e l'accesso a file, oltre che allo sviluppo di applicazioni ASP.NET e Servizi Web XML, il lettore sappia che i programmi analizzati successivamente sono stati realizzati tramite il Framework .NET.

4.2 - IL LINGUAGGIO C#

C# (si pronuncia "C sharp") è un nuovo linguaggio di programmazione progettato per la creazione di un'ampia gamma di applicazioni aziendali eseguite su .NET Framework. C# rappresenta l'evoluzione dei linguaggi C e C++ difatti pur mantenendo la loro potenza ed il loro controllo risulta più semplice, è type-safe (gli oggetti sono opportunamente isolati l'uno dall'altro e quindi protetti da danneggiamenti accidentali o intenzionali) e orientato agli oggetti. Il codice C# viene compilato come codice gestito, il che lo rende in grado di accedere ai servizi di Common Language Runtime. I servizi comprendono interoperabilità dei linguaggi, Garbage Collection e protezione avanzata.

5 - ESPORTAZIONE ED IMPORTAZIONE DATI TRA DBMS TRAMITE FILE XML

5.1 - TECNOLOGIE UTILIZZATE

Di seguito sono trattati alcuni modi per esportare dati da un DBMS in un file XML da cui verranno riletti in vari modi, si eseguiranno operazioni e selezioni su di essi e si inseriranno i dati ricavati dal file XML in un altro database, saranno poi elencate varie tecniche per accedere ai dati XML tramite applicazioni .NET. In particolare come piattaforma DBMS si è utilizzato SQL Server 2000, questo prodotto della Microsoft si è dimostrato un buon DBMS Web e di commercio elettronico, le sue caratteristiche migliori sono l'utilizzo di un'interfaccia user friendly che permette di eseguire in modo intuitivo tutte le più comuni operazioni per la consultazione e il mantenimento del database; a questa si unisce il supporto della tecnologia XML e di tecnologie ad essa associate, difatti SQL Server 2000 supporta le Query URL, gli XML Updategram e Query XPath. Oltre alle tecnologie XML in precedenza menzionate SQL Server 2000 consente di ritornare il risultato di una query oltre che nel solito formato (come set di record) in formato XML, questo si ottiene semplicemente inserendo la clausola FOR XML nella query, tramite FOR XML AUTO per esempio i valori delle colonne vengono forniti come valori di attributi col nome della colonna dentro un elemento col nome della tabella. E' inoltre possibile per gli utenti crearsi le proprie funzioni SQL così da personalizzare maggiormente le funzionalità del DBMS.
Per finire SQL Server 2000 offre un elevato grado di sicurezza che permette di accedere in modo sicuro ai dati, autenticando l'utente e tramite una politica di concessione o meno di diritti di esecuzione di particolari istruzioni sul database si può evitare che vengano compromessi i dati o che l'utente abbia accesso ad informazioni riservate.

Per i motivi sopra elencati SQL Server 2000 è uno strumento utile per gli sviluppatori di applicazioni Web database.

Per accedere al database in modalità client dall'applicazione .NET sono stati utilizzati i componenti di ADO .NET che andremo ora ad esaminare.

5.2 - PANORAMICA SU ADO.NET

I componenti Data .NET sono forniti come un assortimento separato di componenti ognuno associato ad una particolare origine dati. ADO.NET fornisce uniformità di accesso sia per origini dati quali Microsoft SQL Server, sia per origini dati esposte tramite OLE DB e XML. Le applicazioni consumer che supportano la condivisione dei dati, sono in grado di utilizzare ADO.NET per connettersi a tali origini dati e recuperare, modificare e aggiornare i dati. Ogni serie di oggetti associata ad una particolare origine dati è chiamata provider gestito, esistono svariati provider gestiti integrati o integrabili in .NET, in ADO.NET sono inclusi i provider di dati .NET Framework per la connessione a un database, l'esecuzione di comandi e il recupero di risultati. Tali risultati sono elaborati direttamente o inseriti in un oggetto **DataSet** di ADO.NET, in modo da consentirne l'esposizione all'utente in una modalità adeguata, unitamente ai dati provenienti da più origini. È inoltre possibile utilizzare l'oggetto **DataSet** indipendentemente da un provider di dati .NET Framework per gestire i dati locali dell'applicazione o derivati da XML. Le classi ADO.NET sono contenute nella libreria **System.Data.dll** e sono integrate con le classi XML contenute in **System.Xml.dll**, tramite l'XML viene così fornito un accesso disconnesso ai dati che potranno

essere elaborati senza la necessità di essere connessi al database. Come si evince ADO.NET fornisce una stretta integrazione con XML, una rappresentazione comune dei dati con la possibilità di combinare dati provenienti da diverse origini e funzionalità ottimizzate per l'interazione con un database.

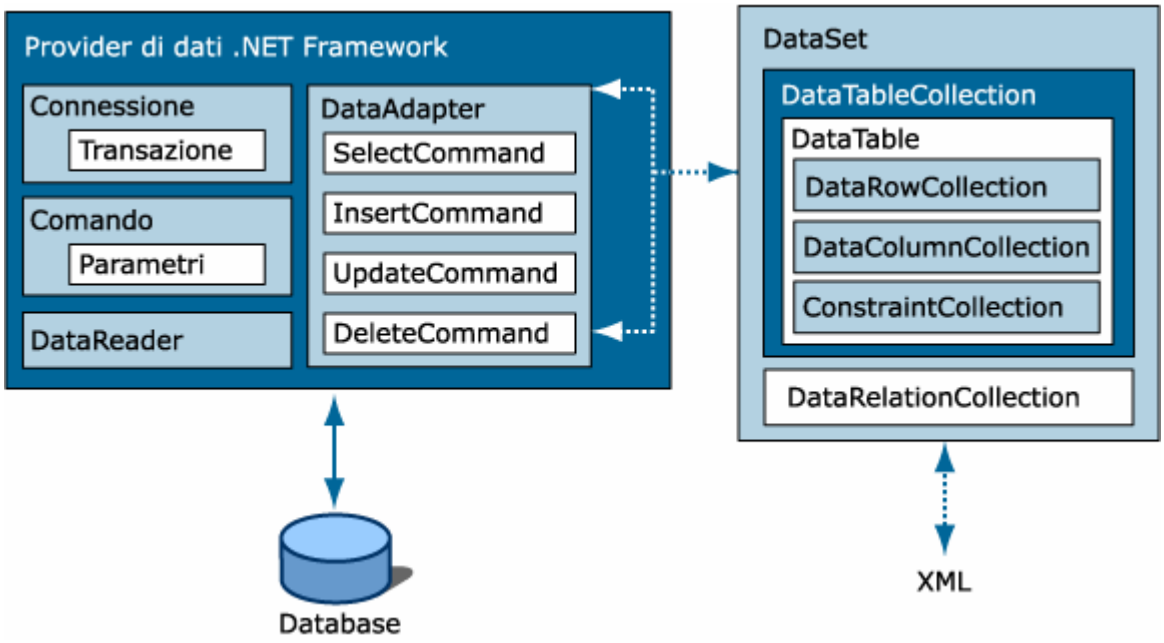

figura 2 Architettura di ADO.NET

5.3 - CONNESSIONE COL FRAMEWORK .NET A SQL SERVER

5.3.1 SCHEMA DEL DATABASE

Questa è una vista parziale del database, sono rappresentate soltanto le tabelle su cui si eseguiranno operazioni di esportazione/importazione dati. La tabella centrale del database è la tabella T_Pratiche_edili che identifica le pratiche edilizie di cui verranno esportati dati. Soggetti è un'associazione tra le tabelle T_Pratiche_edili, T_ruolo_soggetti e T_dati_anagrafici_soggetti ed è implementata attraverso una tabella che contiene i riferimenti ai dati anagrafici e ai ruoli dei soggetti (memorizzati nelle omonime tabelle) oltre che alla pratica. Nella tabella T_Pratiche_edili sono presenti anche i riferimenti alle località e alle vie riguardanti la pratica edilizia rispettivamente memorizzate nelle tabelle T_InterventoLocalita_Pratiche_edili e T_InterventoVia_Pratiche_edili. La tabella T_fogli_mappali contiene i dati riguardanti i fogli e i mappali della pratica, mentre la tabella T_costo_costruzione contiene i dati riguardanti i costi di costruzione per le pratiche. L'associazione tra la tabella T_costo_costruzione e la tabella T_Pratiche_edili è implementata dalla tabella T_Pratiche_edili_costi_costruzione_legame. Per finire vi sono una serie di tabelle contenenti dati inerenti i costi di costruzione che sono associate alla tabella T_costo_costruzione tramite vincoli di integrità referenziale.

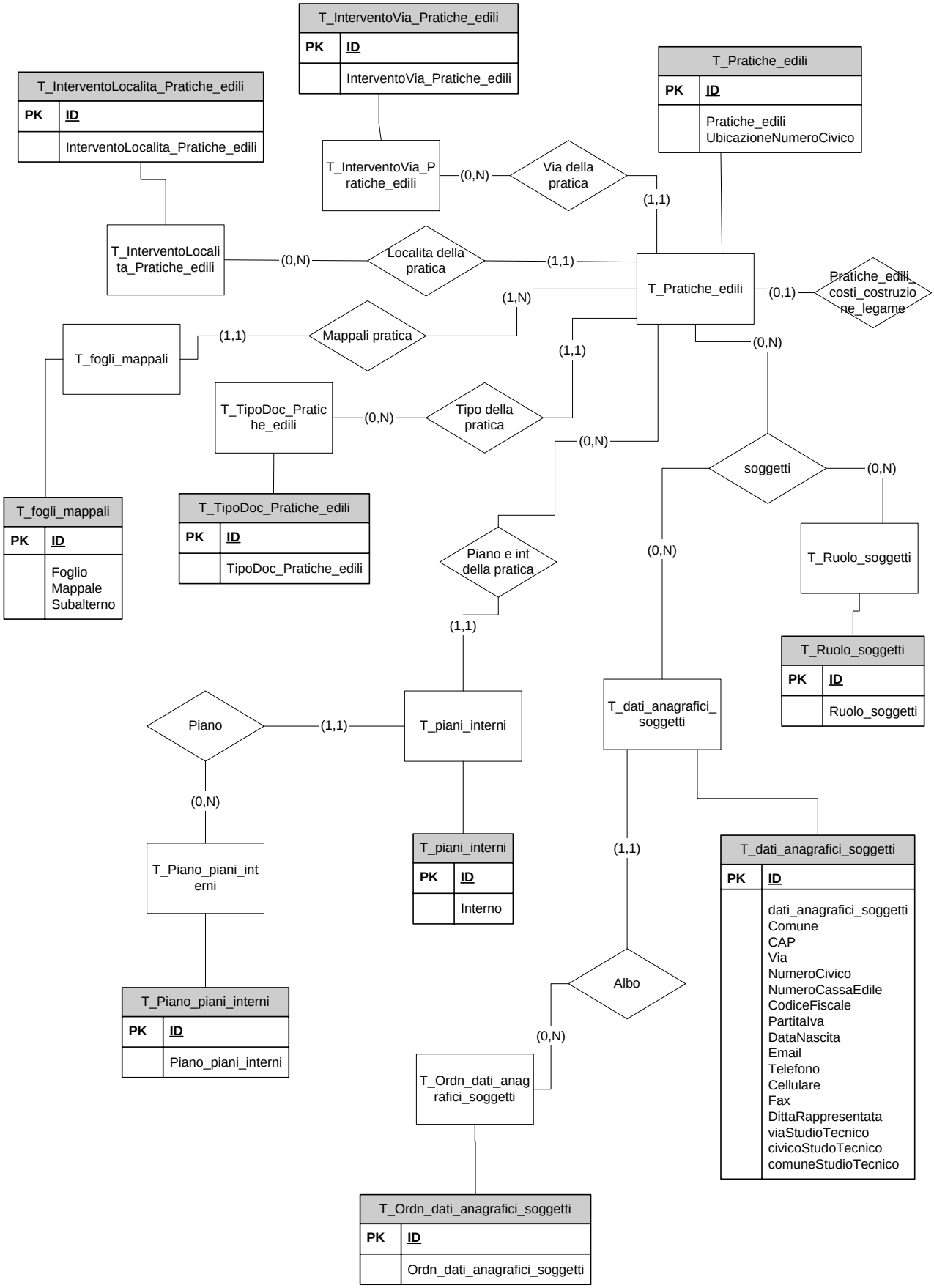

figura 3 schema del Database: prima parte

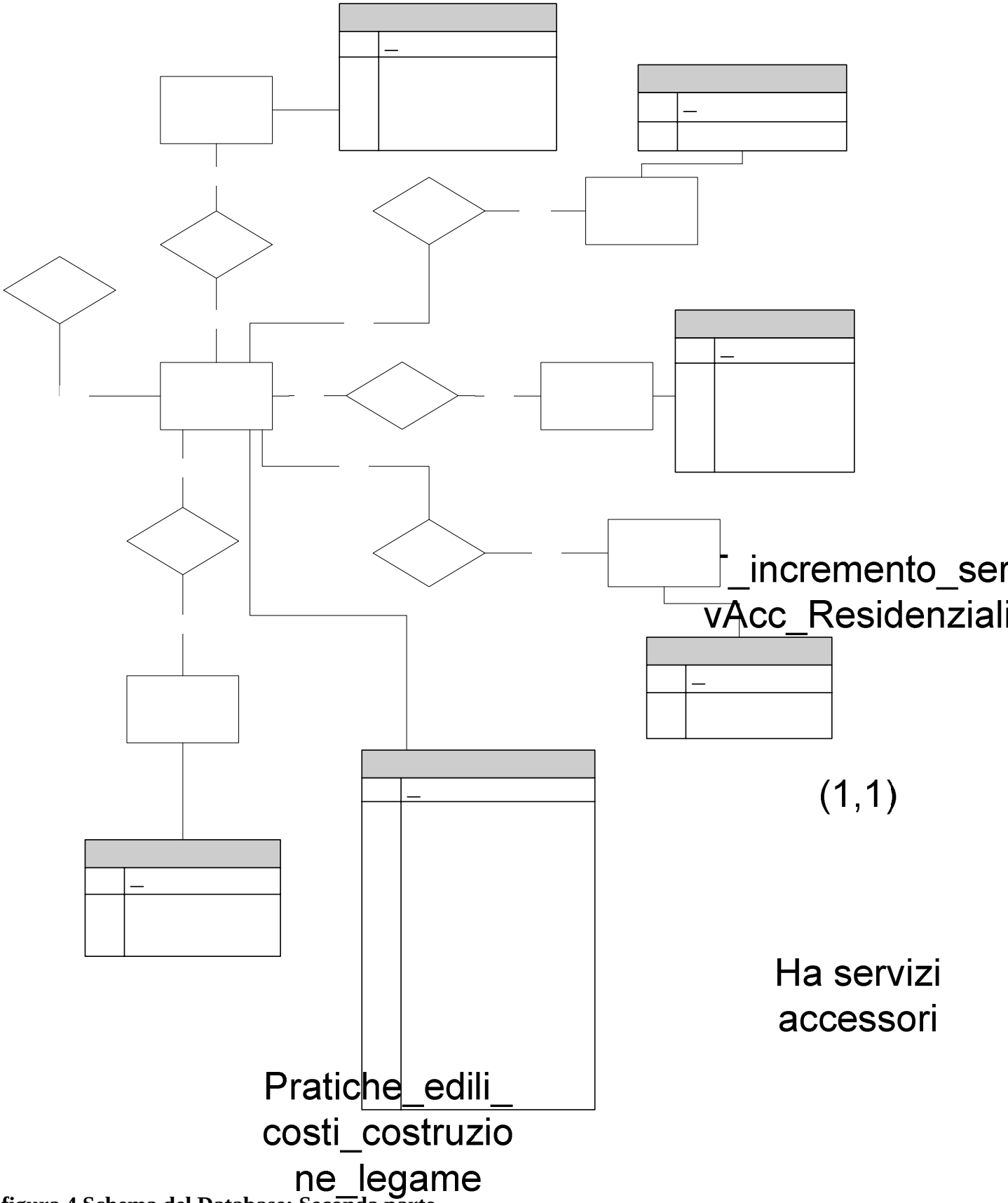

figura 4 Schema del Database: Seconda parte

5.3.2 - UTILIZZO DI OLEDB PER ESPORTARE DATI DA SQL SERVER

Per permettere l'accesso ai dati, il provider di dati .NET Framework per OLE DB si avvale dell'OLE DB nativo tramite l'interoperabilità COM. Il provider di dati .NET Framework per OLE DB supporta transazioni sia locali che distribuite. SQL Server supporta completamente OLE DB, si è deciso di utilizzare questa tecnologia anche se è un po' più lenta del provider nativo SqlClient nell'esecuzione di operazioni sul RDBMS perché a differenza di esso supporta anche vecchie versioni di SQL Server antecedenti alla versione 7.0 cosa che non fa il provider nativo; inoltre OleDb tramite il driver MSDAORA fornisce il provider OLE DB per Oracle consentendo così alle applicazioni di connettersi a questo importante RDBMS semplicemente cambiando il provider e il server nella stringa di connessione.
Vi è un'ulteriore precisazione da fare, i provider OLE DB che richiedono il supporto di interfacce di OLE DB 2.5 non funzioneranno correttamente con il provider di dati .NET Framework per OLE DB.
Lo spazio dei nomi "System.Data.OleDb" consente di effettuare una connessione a un database, eseguire comandi e recuperare risultati. Tali risultati sono elaborati direttamente o inseriti nel DataSet per essere poi processati una volta disconnessi dal database. La semplicità è un aspetto importante della progettazione del provider di dati .NET Framework, che consente di creare un livello minimo tra l'origine dati e il codice, migliorando quindi le prestazioni senza rinunciare ad alcuna funzionalità.

Per usare OLEDB .NET Data Provider nell'applicazione, si deve importare lo spazio dei nomi "System.Data.OleDb" tramite la direttiva:

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

La seconda direttiva consente di importare lo spazio dei nomi che ci permette di utilizzare gli oggetti di ADO.NET.

Tra i vari namespace da importare vi deve essere anche quello riguardante il file di configurazione che contiene la dichiarazione e definizione di vari oggetti utilizzati per riferirsi a tabelle e ad altri campi.

```
using Configurazione_Tabelle;
```

Vengono poi dichiarati gli oggetti che saranno utilizzati in seguito nel programma, tra cui l'oggetto conf di tipo configurazione che contiene la definizione di costanti utilizzate per la connessione al database, la definizione delle tabelle su cui saranno eseguite le query e altre variabili utili in fase di programmazione.

```
public class WriteXML_class
{
        configurazione conf = new configurazione();
        protected OleDbConnection cn;
        protected OleDbDataAdapter c_select,c_select2,….;
        protected string error;
        ….
        ….
}
```

Dopo è necessario creare una connessione che utilizzi i driver OleDb per connettersi al database, SQL Server in questo caso. La connessione dovrà indicare il provider utilizzato, in questo caso SQLOLEDB, il server su cui è fisicamente presente il database utilizzato, il nome del database a cui ci si connette specificato da Initial Catalog, per finire con l'ID e la password associate all'utente che si utilizza per connettersi al database.

Si crea il costruttore della classe in cui la connessione viene inizializzata in modo da poter essere utilizzata in seguito.

```
public WriteXML_class()
{
        cn = new OleDbConnection("provider=SQLOLEDB; user id="+conf.User+";
password="+conf.Pwd+"; server="+conf.Server+";Initial Catalog='"+conf.Database+"' ");
}
```

La connessione al database verrà poi aperta e chiusa utilizzando i metodi **Open( )** e **Close( )** della classe **OleDbConnection**.

Sarà ora possibile eseguire le query sul database dell'RDBMS SQL Server per reperire i dati utili che dovranno poi essere scritti in un file XML per essere poi selezionati e inseriti in un altro database.

Rispetto alle tipiche istruzioni SELECT di SQL 92, SQL Server 2000 presenta questa variante per l'esportazione di dati in formato XML, che però non sarà utilizzata nel programma.

```
[ FOR { BROWSE | XML |  { RAW | AUTO |  EXPLICIT }
      [ , XMLDATA ]
      [ , ELEMENTS ]
      [ , BINARY base64 ]
   }
]
```

Viene ora esposto un possibile utilizzo di SQL 92 embedded C#

Per recuperare i dati in questo caso si è deciso di utilizzare un oggetto **OleDbDataAdapter**, questo oggetto funziona come un ponte tra un oggetto **DataSet** e un'origine dati costituita da un database, tale funzione è ottenuta grazie al metodo **Fill** che permette di riempire il DataSet con il resultset di dati ritornato dalla query effettuata sul database o tramite il metodo Update che chiama le rispettive istruzioni INSERT, UPDATE o DELETE per ciascuna riga inserita, aggiornata o eliminata nell'oggetto DataSet. Per l'oggetto **OleDbDataAdapter** possono essere utilizzate varie proprietà tra cui **SelectCommand** che permette di ottenere o impostare un'istruzione SQL o una stored procedure per selezionare record nell'origine dati oppure **InsertCommand**, **UpdateComand** e **DeleteCommand** che servono rispettivamente per ottenere o impostare istruzioni SQL che inseriscono, aggiornano o eliminano record dal DataSet. All'OleDbDataAdapter si deve poi aggiungere oltre al comando la connessione.

Analizziamo ora la funzione che recupera i dati dal database e ritorna un oggetto DataSet che li contiene.

```
public DataSet Select_pratiche()
{

DataSet ds = new DataSet();

// viene aperta la connessione al database

cn.Open();

//vengono create le stringhe rappresentanti le query SQL ed in seguito vengono inizializzati gli
//oggetti OleDbDataAdapter con queste stringhe e l'oggetto OleDbConnection precedentemente
//creato

string SQL="SELECT T1.ID AS id,T1.ID AS chiave,T1.TipoDoc_Pratiche_edili AS
codiceintervento,T2.TipoDoc_Pratiche_edili AS descrizioneintervento,T1.Pratiche_edili AS
OggettoPratica ";
SQL += "FROM "+conf.TABELLA_PRATICHE_EDILI+" AS
T1,"+conf.TABELLA_TIPODOC_PRATICHE_EDILI+" AS T2 ";
SQL += "WHERE T1.TipoDoc_Pratiche_edili = T2.ID ";
SQL += "ORDER BY T1.ID ASC ";

c_select = new OleDbDataAdapter(SQL,cn);

string SQL2 ="SELECT T1.Pratiche_edili AS pratica,T2.Ruolo_soggetti AS
tipoSoggetto,T1.dati_anagrafici_soggetti AS Codice  ";
SQL2 += "FROM "+conf.TABELLA_SOGGETTI+" AS T1,"+conf.TABELLA_RUOLO_SOGGETTI+"
AS T2 ";
SQL2 += "WHERE T2.ID = T1.Ruolo_soggetti ";
SQL2 += "ORDER BY T1.Pratiche_edili ASC ";

c_select2 = new OleDbDataAdapter(SQL2,cn);

string SQL3="SELECT T1.ID AS Soggetto,T1.dati_anagrafici_soggetti AS Nome,T1.Via AS
Indirizzo,T1.NumeroCivico AS NumeroCiv,T1.Comune AS Località,T1.CAP,T1.NumeroCassaEdile
AS NrAlbo,T2.Ordn_dati_anagrafici_soggetti AS Albo,T1.CodiceFiscale AS
CodiceFisc,T1.PartitaIva AS PIVA,T1.DataNascita,T1.Email AS email,T1.Telefono AS
telefono,T1.Cellulare AS cellulare,T1.Fax AS fax,T1.DittaRappresentata AS
PersonaGiuridica,T1.viaStudioTecnico AS IndirizzoStudio,T1.civicoStudioTecnico AS
NumeroCivStudio,T1.comuneStudioTecnico AS LocalitàStudio  ";
SQL3 += "FROM "+conf.TABELLA_DATI_ANAGRAFICI_SOGGETTI+" AS
T1,"+conf.TABELLA_ORDN_DATI_ANAGRAFICI_SOGGETTI+" AS T2 ";
SQL3 += "WHERE T1.Ordn_dati_anagrafici_soggetti = T2.ID ";
SQL3 += "ORDER BY T1.ID ASC ";

c_select3 = new OleDbDataAdapter(SQL3,cn);

string SQL4= "SELECT T1.ID AS pratica,T2.InterventoVia_Pratiche_edili AS
DescrizioneVia,T1.UbicazioneNumeroCivico AS NumeroCiv,T3.Interno,T5.Piano_piani_interni AS
Piano,T4.InterventoLocalita_Pratiche_edili AS Località ";
SQL4 += "FROM "+conf.TABELLA_PRATICHE_EDILI+" AS T1 ";
SQL4 += "LEFT OUTER JOIN "+conf.TABELLA_PIANI_INTERNI+" AS T3 ON T1.ID =
T3.Pratiche_edili ";
SQL4 += "LEFT OUTER JOIN "+conf.TABELLA_PIANO_PIANI_INTERNI+" AS T5 ON T5.ID =
T3.Piano_piani_interni ";
SQL4 += "LEFT OUTER JOIN "+conf.TABELLA_INTERVENTOVIA_PRATICHE_EDILI+" AS T2
ON T1.InterventoVia_Pratiche_edili = T2.ID ";
```

```
SQL4 += "LEFT OUTER JOIN "+conf.TABELLA_INTERVENTOLOCALITA_PRATICHE_EDILI+"
AS T4 ON T1.InterventoLocalita_Pratiche_edili = T4.ID ";
SQL4 += "ORDER BY T1.ID ASC ";

c_select4 = new OleDbDataAdapter(SQL4,cn);


string SQL5="SELECT T3.Pratiche_edili AS pratica,T1.ID,T1.SupUtileAttivitaTuristicheCom AS
SupUtile,T1.Snr,T1.RapportoSnrSu AS Rapporto,T1.SnrRagguagliata AS snr60,T1.Sc,T1.Su AS
suNonRes,T1.Sa,T1.SaRagguagliata AS sa60,T1.St,T1.I,T2.classi_edifici_maggiorazioni AS
Classe,T1.Maggiorazione,T1.CostoCostruzioneMq AS A,T1.CostoCostruzioneMqMaggiorato AS
B,T1.CostoCostruzioneEdificio AS C,T1.Qpercentuale AS D,T1.Itre AS I3,T1.costo_costruzione AS
CostoCostruzione ";
SQL5 += "FROM "+conf.TABELLA_COSTO_COSTRUZIONE+" AS
T1,"+conf.TABELLA_CLASSI_EDIFICI_MAGGIORAZIONI+" AS
T2,"+conf.PRATICHE_EDILI_COSTI_COSTRUZIONE_LEGAME+" AS T3 ";
SQL5 += "WHERE T1.classi_edifici_maggiorazioni = T2.ID AND T3.costo_costruzione =T1.ID ";
SQL5 += "ORDER BY T3.Pratiche_edili ASC";

c_select5 = new OleDbDataAdapter(SQL5,cn);

string SQL6= "SELECT T2.ID AS CostoCostruzione,T1.classi_superficie AS Id,T1.Alloggi AS
NumAlloggi,T1.SupUtileAbitabile AS SupUtile,T1.Rapporto_Totale_Su AS
Rapporto,T1.Incremento_ClassiSup AS PercentualeIncremento,T2.Iuno AS I1 ";
SQL6 += "FROM "+conf.TABELLA_INCREMENTO_SUPERFICIE_UTILE+" AS
T1,"+conf.TABELLA_COSTO_COSTRUZIONE+" AS
T2,"+conf.PRATICHE_EDILI_COSTI_COSTRUZIONE_LEGAME+" AS T3 ";
SQL6 += "WHERE T2.ID = T1.costo_costruzione AND T2.ID = T3.costo_costruzione ";
SQL6 += "ORDER BY T2.ID ASC ";

c_select6 = new OleDbDataAdapter(SQL6,cn);

string SQL7= "SELECT T2.ID AS CostoCostruzione,T1.CodiceDestinazione AS
Id,T1.SuperficieNettaServAcc AS SupNettaAcc ";
SQL7 += "FROM "+conf.TABELLA_SUPERFICI_SERVIZI_ACCESSORI+" AS
T1,"+conf.TABELLA_COSTO_COSTRUZIONE+" AS
T2,"+conf.PRATICHE_EDILI_COSTI_COSTRUZIONE_LEGAME+" AS T3 ";
SQL7 += "WHERE T2.ID = T1.costo_costruzione AND T2.ID = T3.costo_costruzione ";
SQL7 += "ORDER BY T2.ID ASC ";

c_select7 = new OleDbDataAdapter(SQL7,cn);

string SQL8= "SELECT T2.ID AS CostoCostruzione,T1.DescrizioneIntervVariabilita AS
Id,T1.IntervalloRicorre AS IpotesiSelezionata,T2.Idue AS I2 ";
SQL8 += "FROM "+conf.TABELLA_COSTO_COSTRUZIONE+" AS T2 JOIN
"+conf.PRATICHE_EDILI_COSTI_COSTRUZIONE_LEGAME+" AS T3 ON T2.ID =
T3.costo_costruzione ";
SQL8 += "LEFT OUTER JOIN "+conf.TABELLA_INCREMENTO_SERVACC_RESIDENZIALI+" AS
T1 ON T2.ID = T1.costo_costruzione ";
SQL8 += "ORDER BY T2.ID ASC ";

c_select8 = new OleDbDataAdapter(SQL8,cn);

string SQL9= "SELECT T1.costo_costruzione AS CostoCostruzione,
COUNT(T1.FlagIpotesiRicorre)AS NumIpotesi ";
```

```
SQL9 += "FROM "+conf.TABELLA_INCREMENTO_PER_PARTICOLARI_CARATTERISTICHE+"
AS T1,"+conf.PRATICHE_EDILI_COSTI_COSTRUZIONE_LEGAME+" AS T2 ";
SQL9 += "WHERE T1.FlagIpotesiRicorre = 1 AND T1.costo_costruzione = T2.costo_costruzione
GROUP BY T1.costo_costruzione ";
SQL9 += "UNION ";
SQL9 += "SELECT T3.costo_costruzione AS CostoCostruzione,0 ";
SQL9 += "FROM "+conf.TABELLA_INCREMENTO_PER_PARTICOLARI_CARATTERISTICHE+"
AS T3,"+conf.PRATICHE_EDILI_COSTI_COSTRUZIONE_LEGAME+" AS T4 ";
SQL9 += "WHERE T3.costo_costruzione = T4.costo_costruzione AND ( SELECT
COUNT(FlagIpotesiRicorre) FROM T_incr_particolari_caratteristiche WHERE FlagIpotesiRicorre =
0 AND costo_costruzione = T3.costo_costruzione ) = 5 ";
SQL9 += "GROUP BY T3.costo_costruzione ";

c_select9 = new OleDbDataAdapter(SQL9,cn);

string SQL10= "SELECT Pratiche_edili AS pratica,ID AS Id,Foglio,Mappale,Subalterno AS Sub ";
SQL10 += "FROM "+conf.TABELLA_FOGLI_MAPPALI;
SQL10 += " WHERE Foglio IS NOT NULL ";
SQL10 += "ORDER BY Pratiche_edili ASC ";

c_select10 = new OleDbDataAdapter(SQL10,cn);
```

Ora viene riempito il DataSet tramite il metodo Fill con le tabelle create dalle query

```
try
{
        error="1";
        c_select.Fill(ds,"pratiche");

        error="2";
        c_select2.Fill(ds,"Soggetti");
        …….
        …….
}
```

Viene utilizzata la variabile error per individuare quale istruzione genera l'eccezione all'interno del try per poi lanciare l'opportuna eccezione all'interno del catch.

```
catch (Exception ex)
{
        switch (error)
        {
                case "1" :
                throw new ArgumentException(c_select.SelectCommand.CommandText+" Errore:
        "+ex.Message);
                case "2" :
                throw new ArgumentException(c_select2.SelectCommand.CommandText+" Errore:
                "+ex.Message);
        ……..
        ……..

        }
}
```

Alla fine si chiude la connessione al database

```
finally
{
        if (cn.State == ConnectionState.Open)
        {
                cn.Close()
        }
}
return ds;
} // Select_pratiche()
```

Si dichiara un **DataSet** in cui saranno caricati i dati tramite la funzione **Select_Pratiche** che riempie il DataSet con i dati provenienti dal database, dopodiché si definiranno le relazioni tra le varie **DataTable** contenute all'interno del DataSet e si devono applicare i vincoli sulle colonne delle DataTable. Ora il DataSet può essere visto come un database in memoria e non resta altro da fare che scriverlo in un file XML.

Una **DataSet** rappresenta una cache di dati in memoria nel nostro caso recuperati da un database, esso è costituito da un insieme di oggetti **DataTable** che possono essere messi in relazione tra loro attraverso delle **DataRelations**. Tramite **Constraints** è poi possibile accedere agli insiemi di vincoli applicabili alle singole tabelle del DataSet. Le tabelle sono contenute nell'insieme **DataTableCollections** ed è possibile accedere alle singole tabelle del DataSet mediante una sintassi di questo tipo **NomeDataSet.Tables["nomeTabella"]**, nell'accesso alle tabelle tramite il nome ci si deve ricordare che i nomi sono Case sensitive, al posto del nome della tabella si può utilizzare l'indice che la identifica. I dati e gli schemi del DataSet vengono letti e scritti come documenti XML, per questo sono presenti metodi come **WriteXml** e **WriteXmlSchema** per scrivere i dati del **DataSet** e lo schema in un file XML validato tramite XSD.

```
//Si dichiarano gli oggetti che verranno utilizzati
public class paginaSelectDB: Page
{
        protected WriteXML_class reader = new WriteXML_class();
        protected DataSet ds;
        protected DataView dvPratiche ;
        protected DataGrid dgTabella;
        ……
        ……

public void Page_Load(object o,EventArgs e)
{
        if (!IsPostBack)
                {
//Si caricano i dati tramite questa funzione
                        GeneraDati ();
                }
}// Page_Load()
public void GeneraDati()
{
// vengono caricati i dati nel DataSet
        ds = reader.Select_pratiche();
```

Si applicano i vincoli di univocità sui valori delle colonne rappresentanti le chiavi primarie delle tabelle e poi si stabilisce che tali colonne non possono avere valori nulli impostando la proprietà **AllowDBNull** su false.

```
ds.Tables[“pratiche”].Constraints.Add( new UniqueConstraint(ds.Tables[0].Columns["chiave"]));
ds.Tables[“CostoCostruzione”].Constraints.Add( new
UniqueConstraint(ds.Tables[4].Columns["ID"]));
….
….
ds.Tables[“pratiche”].Columns["chiave"].AllowDBNull = false;
ds.Tables[“CostoCostruzione”].Columns["ID"].AllowDBNull = false;
….
….
```

Ora si applicano i vincoli di integrità referenziale sulle tabelle, applicando così delle restrizioni sulle colonne tramite relazioni chiave primaria/chiave esterna. La/Le prima/e colonna/e rappresenta/no la chiave primaria, mentre la seconda rappresenta la ForeignKey che ad essa si riferisce.

```
ds.Tables[“Soggetti”].Constraints.Add( new
ForeignKeyConstraint(ds.Tables[“pratiche”].Columns["chiave"],ds.Tables[“Soggetti”].Columns["prati
ca"] ));

ds.Tables[“Ubicazione”].Constraints.Add( new
ForeignKeyConstraint(ds.Tables[“pratiche”].Columns["chiave"],ds.Tables[“Ubicazione”].Columns["p
ratica"] ));

ds.Tables[“Mappali”].Constraints.Add( new
ForeignKeyConstraint(ds.Tables[“pratiche”].Columns["chiave"],ds.Tables[“Mappali”].Columns["prati
ca"] ));

ds.Tables[“Tabella1”].Constraints.Add( new
ForeignKeyConstraint(ds.Tables[“CostoCostruzione”].Columns["ID"],ds.Tables[“Tabella1”].Column
s["CostoCostruzione"] ));

ds.Tables[“Tabella2”].Constraints.Add( new
ForeignKeyConstraint(ds.Tables[“CostoCostruzione”].Columns["ID"],ds.Tables[“Tabella2”].Column
s["CostoCostruzione"] ));
…….
…….
```

Perché i vincoli vengano applicati al DataSet si deve impostare la proprietà **EnforceConstraints** su true.

```
ds.EnforceConstraints = true;
…..
…..
GeneraVista();
}// GeneraDati()
```

A questo punto è possibile scrivere in un file XML i dati del DataSet e il realtivo schema, questo viene fatto in una funzione richiamata dalla pressione di un Button della pagina ASP.NET, utilizzando il metodo **WriteXml** che ci permette di scrivere sia i dati che lo schema nello stesso file impostando il parametro **XmlWriteMode**. I valori che **XmlWriteMode** può assumere sono tre:

DiffGram: Scrive l'intero oggetto DataSet come DiffGram, inclusi i valori originali e quelli correnti.

IgnoreSchema: Scrive il contenuto corrente dell'oggetto DataSet come dati XML, senza uno schema XSD.

WriteSchema: Scrive il contenuto corrente dell'oggetto DataSet come dati XML con la struttura relazionale di uno schema XSD all'interno dello stesso file XML, così che il file risulti autodescrittivo per i dati contenuti al suo interno.

```
public void ScriviDati(object o,EventArgs e)
{
        GeneraDati();
        string nomefile = "..\..\XML/PI_DG.xml";
        ds.WriteXml(nomefile,XmlWriteMode.WriteSchema);
        ….
        ….
}// ScriviDati()
```

Ora esaminiamo la funzione che si occupa della visualizzazione della tabella centrale tra quelle create nel DataSet, la tabella pratiche che rappresenta il documento centrale a cui tutte le altre tabelle direttamente o indirettamente si riferiscono. Questi dati verranno visualizzati nella pagina ASP.NET che richiamando tutti i metodi visti in precedenza permette di scrivere il documento XML. Per la visualizzazione dei dati della pratica si sono utilizzati gli oggetti **DataView** e **DataGrid**. Il **DataView** rappresenta un vista personalizzata associabile ai dati di una **DataTable**, mentre il **DataGrid** è un controllo elenco associato ai dati, in esso gli elementi dell'origine dati sono visualizzati in una tabella e permette la visualizzazione di una tabella in una pagina ASP.NET.

```
protected void GeneraVista()
{
// inserisco nel DataView la tabella pratiche contenuta nel DataSet ds
        dvPratiche = ds.Tables[“pratiche”].DefaultView;
        dvPratiche.Sort =”chiave ASC”;
        dgTabella.AutoGenerateColumns =true;
        dgTabella.AllowPaging=true;
        dgTabella.PageSize=20;
        dgTabella.PagerStyle.Mode=PagerMode.NumericPages;

// indico la vista come sorgente di dati per il DataGrid e la visualizzo a video tramite un Bind
        dgTabella.DataSource = dvPratiche;
        dgTabella.DataBind();
}// GeneraVista()
```

I record della vista vengono ordinati secondo il valore della colonna chiave in modo ascendente, poi si imposta a 20 il numero di record visualizzabili in una singola pagina e si impostano i parametri per la paginazione. Alla fine si crea la funzione che gestisce l'evento cambio pagina generato nella pagina ASP.NET dall'utente.

```
protected void PaginazioneDataGrid(object o, DataGridPageChangedEventArgs e)
{
        dgTabella.CurrentPageIndex = e.NewPageIndex;
        GeneraDati();
}// Fine della Paginazione
```

5.3.3 - STRUTTURA DEL FILE XML SCRITTO DAL DATASET

Il file XML generato dall'esempio precedente utilizzando gli oggetti messi a disposizione dal Framework .NET contiene lo schema XSD al suo interno così che i dati del documento XML possano essere validati. Lo schema importa il namespace xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" definito dalla Microsoft che comprende definizioni di elementi e di tipi del .NET. La struttura dei nodi è composta dall'elemento root <NewDataSet> che rappresenta il database virtuale, questo conterrà un numero non specificato di elementi figli del tipi definito nello schema, questi elementi hanno lo stesso nome delle tabelle contenute nel DataSet e saranno tanti quante sono le righe della tabella che porta il nome dell'elemento stesso; a loro volta questi elementi conterranno degli elementi figli i cui nomi saranno quelli delle colonne della tabella. Questi sono gli elementi che contengono i dati, i cui valori devono rispettare i vincoli sul tipo definiti dallo schema (string,int…). I campi che possono avere valore nullo sono rappresentati all'interno dello schema come elementi con l'attributo MinOccurs="0", difatti i valori NULL vengono in questo caso rappresentati con elementi mancanti all'interno dell'elemento padre nel documento XML; questa interpretazione è mantenuta anche in fase di caricamento del documento XML all'interno di un DataSet riuscendo così ad identificare i valori NULL. Nel caso in cui non sia specificato altrimenti l'attributo MinOccurs vale 1, in questo modo vengono rappresentati i campi che non possono avere valori nulli. All'interno dell'elemento root sono poi inseriti i vincoli di unicità e di integrità referenziale attraverso gli elementi unique e keyref, grazie a tutti questi controlli sono garantite le relazioni tra gli elementi e i range di valori che possono assumere, così che il documento XML possa essere interpretato come un piccolo database.

Struttura del file XML

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
    <xs:schema id="NewDataSet" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
        <xs:element name="NewDataSet" msdata:IsDataSet="true" msdata:Locale="it-IT">
            <xs:complexType>
                <xs:choice maxOccurs="unbounded">
                    <xs:element name="pratiche">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="id" type="xs:int" minOccurs="0"/>
                                <xs:element name="chiave" type="xs:int"/>
                                <xs:element name="codiceintervento" type="xs:int" minOccurs="0"/>
                                <xs:element name="descrizioneintervento" type="xs:string" minOccurs="0"/>
                                <xs:element name="OggettoPratica" type="xs:string" minOccurs="0"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="Soggetti">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="pratica" type="xs:int" minOccurs="0"/>
                                <xs:element name="tipoSoggetto" type="xs:string" minOccurs="0"/>
                                <xs:element name="Codice" type="xs:int" minOccurs="0"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="Indice">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="Soggetto" type="xs:int" minOccurs="0"/>
```

```
            <xs:element name="Nome" type="xs:string" minOccurs="0"/>
            <xs:element name="Indirizzo" type="xs:string" minOccurs="0"/>
            <xs:element name="NumeroCiv" type="xs:string" minOccurs="0"/>
            <xs:element name="Località" type="xs:string" minOccurs="0"/>
            <xs:element name="CAP" type="xs:string" minOccurs="0"/>
            <xs:element name="NrAlbo" type="xs:string" minOccurs="0"/>
            <xs:element name="Albo" type="xs:string" minOccurs="0"/>
            <xs:element name="CodiceFisc" type="xs:string" minOccurs="0"/>
            <xs:element name="PIVA" type="xs:string" minOccurs="0"/>
            <xs:element name="DataNascita" type="xs:dateTime" minOccurs="0"/>
            <xs:element name="email" type="xs:string" minOccurs="0"/>
            <xs:element name="telefono" type="xs:string" minOccurs="0"/>
            <xs:element name="cellulare" type="xs:string" minOccurs="0"/>
            <xs:element name="fax" type="xs:string" minOccurs="0"/>
            <xs:element name="PersonaGiuridica" type="xs:string" minOccurs="0"/>
            <xs:element name="IndirizzoStudio" type="xs:string" minOccurs="0"/>
            <xs:element name="NumeroCivStudio" type="xs:string" minOccurs="0"/>
            <xs:element name="LocalitàStudio" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Ubicazione">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="pratica" type="xs:int" minOccurs="0"/>
            <xs:element name="DescrizioneVia" type="xs:string" minOccurs="0"/>
            <xs:element name="NumeroCiv" type="xs:string" minOccurs="0"/>
            <xs:element name="Interno" type="xs:string" minOccurs="0"/>
            <xs:element name="Piano" type="xs:string" minOccurs="0"/>
            <xs:element name="Località" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="CostoCostruzione">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="pratica" type="xs:int" minOccurs="0"/>
            <xs:element name="ID" type="xs:int"/>
            <xs:element name="SupUtile" type="xs:decimal" minOccurs="0"/>
            <xs:element name="Snr" type="xs:decimal" minOccurs="0"/>
            <xs:element name="Rapporto" type="xs:decimal" minOccurs="0"/>
            <xs:element name="snr60" type="xs:decimal" minOccurs="0"/>
            <xs:element name="Sc" type="xs:decimal" minOccurs="0"/>
            <xs:element name="suNonRes" type="xs:decimal" minOccurs="0"/>
            <xs:element name="Sa" type="xs:decimal" minOccurs="0"/>
            <xs:element name="sa60" type="xs:decimal" minOccurs="0"/>
            <xs:element name="St" type="xs:decimal" minOccurs="0"/>
            <xs:element name="I" type="xs:decimal" minOccurs="0"/>
            <xs:element name="Classe" type="xs:string" minOccurs="0"/>
            <xs:element name="Maggiorazione" type="xs:decimal" minOccurs="0"/>
            <xs:element name="A" type="xs:decimal" minOccurs="0"/>
            <xs:element name="B" type="xs:decimal" minOccurs="0"/>
            <xs:element name="C" type="xs:decimal" minOccurs="0"/>
            <xs:element name="D" type="xs:decimal" minOccurs="0"/>
            <xs:element name="I3" type="xs:decimal" minOccurs="0"/>
            <xs:element name="CostoCostruzione" type="xs:string" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Tabella1">
    <xs:complexType>
        <xs:sequence>
```

```
                        <xs:element name="CostoCostruzione" type="xs:int" minOccurs="0"/>
                        <xs:element name="Id" type="xs:int" minOccurs="0"/>
                        <xs:element name="NumAlloggi" type="xs:decimal" minOccurs="0"/>
                        <xs:element name="SupUtile" type="xs:decimal" minOccurs="0"/>
                        <xs:element name="Rapporto" type="xs:decimal" minOccurs="0"/>
                        <xs:element name="PercentualeIncremento" type="xs:decimal" minOccurs="0"/>
                        <xs:element name="I1" type="xs:decimal" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Tabella2">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="CostoCostruzione" type="xs:int" minOccurs="0"/>
                        <xs:element name="Id" type="xs:string" minOccurs="0"/>
                        <xs:element name="SupNettaAcc" type="xs:decimal" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Tabella3">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="CostoCostruzione" type="xs:int" minOccurs="0"/>
                        <xs:element name="Id" type="xs:string" minOccurs="0"/>
                        <xs:element name="IpotesiSelezionata" type="xs:boolean" minOccurs="0"/>
                        <xs:element name="I2" type="xs:decimal" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Tabella4">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="CostoCostruzione" type="xs:int" minOccurs="0"/>
                        <xs:element name="NumIpotesi" type="xs:int" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Mappali">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="pratica" type="xs:int" minOccurs="0"/>
                        <xs:element name="Id" type="xs:int" minOccurs="0"/>
                        <xs:element name="Foglio" type="xs:string" minOccurs="0"/>
                        <xs:element name="Mappale" type="xs:string" minOccurs="0"/>
                        <xs:element name="Sub" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:choice>
    </xs:complexType>
    <xs:unique name="Constraint1">
        <xs:selector xpath=".//pratiche"/>
        <xs:field xpath="chiave"/>
    </xs:unique>
    <xs:unique name="CostoCostruzione_Constraint1" msdata:ConstraintName="Constraint1">
        <xs:selector xpath=".//CostoCostruzione"/>
        <xs:field xpath="ID"/>
    </xs:unique>
    <xs:keyref name="Mappali_Constraint1" refer="Constraint1" msdata:ConstraintName="Constraint1"
msdata:ConstraintOnly="true">
        <xs:selector xpath=".//Mappali"/>
        <xs:field xpath="pratica"/>
```

```
            </xs:keyref>
            <xs:keyref name="Tabella4_Constraint1" refer="CostoCostruzione_Constraint1"
msdata:ConstraintName="Constraint1" msdata:ConstraintOnly="true">
                <xs:selector xpath=".//Tabella4"/>
                <xs:field xpath="CostoCostruzione"/>
            </xs:keyref>
            <xs:keyref name="Tabella3_Constraint1" refer="CostoCostruzione_Constraint1"
msdata:ConstraintName="Constraint1" msdata:ConstraintOnly="true">
                <xs:selector xpath=".//Tabella3"/>
                <xs:field xpath="CostoCostruzione"/>
            </xs:keyref>
            <xs:keyref name="Tabella2_Constraint1" refer="CostoCostruzione_Constraint1"
msdata:ConstraintName="Constraint1" msdata:ConstraintOnly="true">
                <xs:selector xpath=".//Tabella2"/>
                <xs:field xpath="CostoCostruzione"/>
            </xs:keyref>
            <xs:keyref name="Tabella1_Constraint1" refer="CostoCostruzione_Constraint1"
msdata:ConstraintName="Constraint1" msdata:ConstraintOnly="true">
                <xs:selector xpath=".//Tabella1"/>
                <xs:field xpath="CostoCostruzione"/>
            </xs:keyref>
            <xs:keyref name="Constraint2" refer="Constraint1" msdata:ConstraintOnly="true">
                <xs:selector xpath=".//CostoCostruzione"/>
                <xs:field xpath="pratica"/>
            </xs:keyref>
            <xs:keyref name="Ubicazione_Constraint1" refer="Constraint1" msdata:ConstraintName="Constraint1"
msdata:ConstraintOnly="true">
                <xs:selector xpath=".//Ubicazione"/>
                <xs:field xpath="pratica"/>
            </xs:keyref>
            <xs:keyref name="Soggetti_Constraint1" refer="Constraint1" msdata:ConstraintName="Constraint1"
msdata:ConstraintOnly="true">
                <xs:selector xpath=".//Soggetti"/>
                <xs:field xpath="pratica"/>
            </xs:keyref>
        </xs:element>
    </xs:schema>
.....
XML
.....
</NewDataSet>
```

Si potrebbe sfruttare maggiormente la gerarchia dei dati fornita dall'XML. Per identificare i vincoli d'integrità referenziale veniva utilizzato keyref, ma nell'XML tali relazioni sarebbero meglio rappresentate da relazioni padre-figlio tra gli elementi, ponendo gli elementi dipendenti come figli degli elementi da cui dipendono rendendo così più visibile la struttura dei dati. Per fare ciò si può utilizzare la proprietà **Relations** del **DataSet**, che contiene l'insieme di tutte le relazioni tra le tabelle che vengono implementate da oggetti **DataRelation**, questi oggetti possiedono la proprietà **Nested** che se impostata su true permette la nidificazione delle righe figlie all'interno della colonna padre durante la scrittura come dati XML.
Per nidificare gli elementi precedentemente rappresentati nel DataSet si procede in questo modo:

```
DataRelation praticaMappaliRel;
//Si aggiunge la relazione col nome MappaliDipratica  al DataSet indicando che la colonna pratica
//della tabella Mappali si riferisce alla colonna chiave della tabella pratiche
praticaMappaliRel =
ds.Relations.Add("MappaliDipratica",ds.Tables["pratiche"].Columns["chiave"],ds.Tables["Mappali"].
Columns["pratica"]);
```

```
// si imposta la proprietà nested in modo da nidificare gli elementi
praticaMappaliRel.Nested =true;

ds.WriteXml(nomefile,XmlWriteMode.IgnoreSchema);
```

Ora il documento XML avrà una struttura di questo tipo:

```
<NewDataSet>
    <pratiche>
        <id>121</id>
        <chiave>121</chiave>
        <codiceintervento>5</codiceintervento>
        <descrizioneintervento>Dichiarazione di inizio attività</descrizioneintervento>
        <OggettoPratica>Costruzione di n. 3 garage in Comune di Roncofritto, Via Claudia,
        1235.</OggettoPratica>
        <Mappali>
            <pratica>121</pratica>
            <Id>7671</Id>
            <Foglio>1</Foglio>
            <Mappale>4</Mappale>
            <Sub>1</Sub>
        </Mappali>
    </pratiche>
    <Soggetti>
        <pratica>121</pratica>
        <tipoSoggetto>Richiedente</tipoSoggetto>
        <Codice>89</Codice>
    </Soggetti>
     ……
     ……
</NewDataSet>
```

5.4 - IMPORTAZIONE DI DATI DA UN FILE XML IN SQL SERVER TRAMITE OLEDB

5.4.1 - OBIETTIVO DEL PROGRAMMA

Lo scopo di questo programma eseguito in una pagina ASP.NET è quello di inserire o aggiornare le tabelle già esistenti del database descritto nei capitoli precedenti con i dati letti dal file XML precedentemente scritto, il database in cui verranno importati i dati ha la stessa struttura e le stesse tabelle del database da cui tali dati sono stati esportai. Si procede acquisendo i dati del file XML in un oggetto DataSet; i dati presenti nelle tabelle del DataSet verranno estratti ed inseriti in tabelle temporanee omologhe a quelle del database. Vengono poi invocati i metodi per aggiornare o insere nelle tabelle del database i dati delle tabelle temporanee. Si è proceduto creando un namespace che contiene la classe e i metodi necessari per la creazione delle tabelle per l'inserimento/aggiornamento del database, queste tabelle avranno lo stesso nome delle tabelle su cui andranno ad operare nel database, la stessa premessa vale per le colonne, inoltre vi dovrà essere corrispondenza tra i tipi di queste tabelle e i tipi delle tabelle nel database. Questa corrispondenza è mantenuta dal .NET attraverso le corrispondenze tra i tipi definiti nel DBMS e i tipi del .NET prima e tra i tipi .NET e i tipi XSD dopo per quanto riguarda la scrittura del file XML, il contrario avverrà per la lettura (per avere una visione più chiara delle corrispondenze tra tipi si rimanda alle tabelle in appendice).
Un altro namespace è stato creato per contenere la classe e i metodi che dovranno effettuare le operazioni di inserimento e aggiornamento del database con le tabelle precedentemente descritte, vi saranno sia metodi per l'inserimento/aggiornamento di intere tabelle che metodi per l'inserimento/aggiornamento di singole righe.
Una volta caricato il file XML nel DataSet l'inserimento parte dalle pratiche che ne sono la tabella centrale, tramite questi dati si và ad aggiornare/inserire dati nella corrispondente tabella del database (la tabella T_Pratiche_edili). Si è deciso di applicare la seguente regola nell'aggiornamento/inserimento delle pratiche nel database:

Nel campo chiave della tabella pratiche sono contenuti gli ID (chiave primaria) della tabella T_Pratiche_edili (tabella centrale delle pratiche edilizie) del database da cui sono stati esportati i dati, questo campo è stato inserito per mantenere i vincoli di integrità referenziale nelle altre tabelle esportate, le quali dipendono direttamente o indirettamente da questa tabella.
Il campo id della tabella pratiche contenuta nel DataSet in cui sono stati caricati i dati e lo schema del file XML può avere valori nulli e l'interpretazione di questi è la seguente: se nel campo id è contenuto un valore si deve tentare l'update della pratica edilizia (rappresentata nella tabella T_Pratiche_edili del databse) il cui id è lo stesso del campo sopra citato, se nella tabella non sono presenti record con tale id la pratica viene segnalata e non si procede all'inserimento/aggiornamento delle tabelle correlate con i dati esportati che si riferiscono a tale pratica. Se il campo id non è presente quindi NULL si procede all'inserimento nella tabella T_Pratiche_edili della pratica e all'aggiornamento/inserimento nelle tabelle correlate dei dati esportati che si riferiscono a tale pratica.

5.4.2 - INTEGRAZIONE DELL'XML NEL DATASET

Per riempire un **DataSet** con i dati provenienti da una sorgente XML viene utilizzato il metodo **ReadXml**, questo metodo permette di leggere dati da un file, un flusso o un **XmlReader**.

Per specificare la modalità con cui vengono letti gli schemi si utilizza l'argomento **XmlReadMode** che può assumere questi valori:
Auto: Consente di esaminare il documento o flusso XML e di selezionare l'opzione di lettura più appropriata.
ReadSchema: Consente di leggere gli schemi all'interno del documento XML e di caricare sia i dati che lo schema.
IgnoreSchema: Consente di ignorare gli schemi presenti nel documento XML e di caricarne i dati nello schema del DataSet già presente, se nel DataSet non è presente uno schema non viene caricato nessun dato.
InferSchema: Consente di ignorare parzialmente eventuali schemi inline e di integrare lo schema esistente del DataSet con lo schema XML aggiungendo nuove tabelle o nuove colonne, se si presenta un conflitto tra gli schemi viene generata un'eccezione.
DiffGram: Consente di leggere un DiffGram e di aggiungere i dati allo schema corrente.
Fragment: Consente di leggere più frammenti XML fino al raggiungimento della fine del flusso, i frammenti corrispondenti allo schema del DataSet vengono aggiunti alle tabelle, mentre quelli non corrispondenti vengono eliminati.

5.4.3 - IMPORTAZIONE DEI DATI

Oltre ai namespace da importare per utilizzare gli oggetti messi a disposizione dal .NET si importano il namespace **CreaTabelle_inserimento** contenente la classe **CreaTabelle** che implementa metodi per raggruppare i dati in tabelle temporanee con la stessa struttura delle tabelle del database su cui si effettueranno le operazioni di importazione dati; e il namespace **Inserimento_Tabelle** che contiene la classe **InserisciDati**, la quale implementa i metodi per l'inserimento/aggiornamento delle tabelle del database con le tabelle temporanee. Oltre a questi viene importato anche il namespace **Configurazione_Appl** che incorpora la classe **configurazione** contenente costanti utilizzate più volte nell'applicazione come il nome del database a cui connettersi ecc..

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Xml;
…
using CreaTabelle_inserimento;
using Configurazione_Appl;
using Inserimento_Tabelle;

namespace InserimentoTabelleInDB {

public class paginaInsertTable: Page
{
// SI DICHIARANO GLI OGGETTI UTILIZZATI NELL'APPLICAZIONE

                protected DataGrid  tabella;
                protected DataTable[] miaTab,DtTabella;
                protected DataSet DsSportUnico;
                protected Panel myPanel;
                protected OleDbConnection cn;
                protected OleDbCommand query;
                protected OleDbDataReader reader;
                protected ArrayList AlPratica,AlID;
                protected string nomeTab,SQL;
```

```
….
….
protected CreaTabelle generator;
protected InserisciDati insert;
protected configurazione conf;
protected string urlxml ="../../XML/PI_DG.xml";
```

A questo punto si controlla se il file XML è presente sul server su cui viene eseguita la pagina ASP.NET che richiama questa applicazione e in caso affermativo lo si carica nel DataSet tramite il metodo **ReadXml** precedentemente illustrato.

```
public void Page_Load(object o,EventArgs e) {
        if ( File.Exists(Server.MapPath(@urlxml)) )
        {
              try
              {
                 DsSportUnico.ReadXml(Server.MapPath(@urlxml), XmlReadMode.ReadSchema);
                 filePresente =true;
               }
               catch ( Exception exc) {
                 throw new ArgumentException("Errore durante la lettura del file XML:
"+exc.Message);
                 }

         }
        else {
                 filePresente =false;
        }
}
```

Viene ora creata la struttura delle due tabelle temporanee che conterranno i dati da inserire o utilizzare per un'aggiornamento della tabella T_Pratiche_edili che è la tabella centrale del database. Viene poi creata una terza tabella per contenere eventualmente i dati delle pratiche per cui le operazioni di inserimento/update non sono andate a buon fine. Per sintetizzare rappresentiamo solo la struttura della tabella usata per l'inserimento. Qui di seguito viene rappresentata la funzione che si occupa dell'importazione dei dati nel database.

```
public void CreaTabella_Pratiche_edili(object o,EventArgs e) {

// si istanziano le classi create per effettuare le operazioni di importazione dati nel database
generator = new CreaTabelle();
conf = new configurazione();
insert = new InserisciDati();
…
…
DtTabella = new DataTable[3];

DtTabella[0] = new DataTable("T_Pratiche_edili");
IdColumn = new DataColumn("ID",DsSportUnico.Tables["pratiche"].Columns["chiave"].DataType);
IdColumn.AllowDBNull = false;
DtTabella[0].Columns.Add(IdColumn);
colonna = new
DataColumn("TipoDoc_Pratiche_edili",DsSportUnico.Tables["pratiche"].Columns["codiceintervento
"].DataType);
colonna.AllowDBNull = false;
```

```
DtTabella[0].Columns.Add(colonna);
colonna = new
DataColumn("Pratiche_edili",DsSportUnico.Tables["pratiche"].Columns["OggettoPratica"].DataTyp
e);
DtTabella[0].Columns.Add(colonna);
colonna = new
DataColumn("UbicazioneNumeroCivico",DsSportUnico.Tables["Ubicazione"].Columns["NumeroCiv
"].DataType);
DtTabella[0].Columns.Add(colonna);
colonna = new
DataColumn("InterventoVia_Pratiche_edili",System.Type.GetType("System.Int32"));
DtTabella[0].Columns.Add(colonna);
colonna = new
DataColumn("InterventoLocalita_Pratiche_edili",System.Type.GetType("System.Int32"));
DtTabella[0].Columns.Add(colonna);
….
….
```

L'oggetto **OleDbCommand** rappresenta una istruzione SQL o una stored procedure, al Command deve essere associata una connessione (identificata dalla proprietà Connection) per poter essere eseguito. Attraverso la proprietà **CommandText** si ottiene o imposta l'istruzione SQL o la stored procedure da eseguire. Il Command comprende tre diversi metodi per eseguire comandi su un'origine dati:
ExecuteReader: Esegue comandi che restituiscono righe, tipicamente viene utilizzato per generare delle SELECT; in fase di esecuzione invia la proprietà CommandText alla proprietà Connection e genera un oggetto OleDbDataReader che leggerà i record ritornati dal comando eseguito.

ExecuteNonQuery: Esegue istruzioni SQL quali INSERT, DELETE, UPDATE, SET, creazione e modifica di tabelle ecc.. ritorna il numero di righe interessate dal comando nel caso si tratti di un inserimento/cancellazione/aggiornamento, mentre ritorna -1 per le altre istruzioni o se si verifica un'operazione di Rollback.

ExecuteScalar: Recupera un valore singolo per esempio un valore aggregato da un database, tipicamente esegue la query e ritorna la prima colonna della prima riga del gruppo di risultati ritornato dalla query.
L'oggetto **OleDbDataReader** permette di leggere un flusso di righe di dati in modalità forward-only da un'origine dati. Finche il DataReader è in uso l'oggetto Connection associato al reader è occupato e non è possibile eseguire altre operazioni/comandi su di esso fino a quando non viene richiamato il metodo **Close** del DataReader che rilascia la Connection. Il metodo **Read** permette di spostare il DataReader al record successivo in fase di lettura, per ottenere il valore di una specifica colonna del record su cui è posizionato in quel momento il DataReader si utilizza il metodo **GetValue**(numero ordinale della colonna a base 0).
Verrà fatto un largo uso in seguito degli oggetti appena descritti.

Ora si crea una connessione al database e tramite un oggetto **OleDbCommand** si esegue una SELECT sulla tabella T_Pratiche_edili per recuperarne gli ID, così da poterli confrontare con gli eventuali valori (se presenti) della colonna id della tabella pratiche di **DsSportUnico** per controllare se gli ID delle eventuali pratiche da aggiornare combaciano, nel caso non combacino tali pratiche verranno inserite nella tabella delle pratiche non andate a buon fine.

```
nomeTab="pratiche";
cn = new OleDbConnection("provider=SQLOLEDB; user id="+conf.utente+";password=
"+conf.Pwd+";server="+conf.nomeServer+";Initial Catalog="'+conf.nomeDataBase+"'");
```

```
SQL = "SELECT ID FROM "+conf.TABELLA_PRATICHE_EDILI+" ";

AlID = new ArrayList();
query = new OleDbCommand(SQL,cn);
try
{
        query.Connection.Open();
        reader = query.ExecuteReader();
        while ( reader.Read() )
        {
                id=reader.GetInt32(0);
                AlID.Add(id);
        }
        reader.Close();
}
catch(Exception ex) { throw new ArgumentException …. }
finally {
        query.Connection.Close();
}
```

Ora che gli ID sono memorizzati nell'array **AlID** si procede al confronto con i valori del campo id della tabella pratiche. Se tali valori non sono presenti si procede inserendo i dati della pratica, se sono presnti e combaciano con gli ID contenuti in AlID si procede all'aggiornamento di tali record con i valori della tabella pratiche, mentre se sono presenti ma non combaciano la pratica viene inserita nella tabella delle pratiche che presentano problemi e non viene fatto altro per quel record (che rappresenta una pratica). Queste operazioni vengono ripetute all'interno di un ciclo per tutte le righe della tabella pratiche.

```
for (int i=0;i < DsSportUnico.Tables[nomeTab].Rows.Count ;i++ )
{
        if ( DsSportUnico.Tables[nomeTab].Rows[i][ "id"] == DBNull.Value )
        {
        DrRiga = DtTabella[0].NewRow();
        DrRiga[0] = DsSportUnico.Tables["pratiche"].Rows[i]["chiave"];
        DrRiga[1] = DsSportUnico.Tables["pratiche"].Rows[i]["codiceintervento"];
        DrRiga[2] = DsSportUnico.Tables["pratiche"].Rows[i]["OggettoPratica"];
        DrRiga[3] = DBNull.Value;
        DrRiga[4] = DBNull.Value;
        DrRiga[5] = DBNull.Value;
        DtTabella[0].Rows.Add(DrRiga);
        …….
        …….
```

A questo punto richiamo il metodo che controlla sei i valori della colonna DescrizioneVia della tabella Ubicazione, associati alla pratica tramite la Foreign Key pratica, sono già presenti nella tabella T_InterventoVia_Pratiche_edili. In caso affermativo verrà restituita una tabella vuota, mentre se vi saranno delle vie non ancora inserite verrà restituita una tabella contenente i dati da inserire nella tabella delle vie. Lo stesso viene fatto tramite un altro metodo sulla tabella T_InterventoLocalita_Pratiche_edili, contenente la località della pratica, che dovrà eventualmente essere aggiornata con i dati della colonna Località della tabella Ubicazione.

```
miaTab[0] = new DataTable();
miaTab[0]=generator.CreaTabella_InterventoVia_Pratiche_edili(DsSportUnico,(int)DsSportUnico.T
ables["pratiche"].Rows[i]["chiave"]);
```

Una volta in possesso delle tabelle temporanee con gli eventuali dati da inserire nelle omonime tabelle del database, vengono invocati i metodi di inserimento sull'istanza della classe addetta alle operazioni di INSERT/UPDATE passandogli come argomento la tabella contenente i dati da inserire ed il nome della tabella in cui effettuare l'inserimento.

```
righeScritte=0;

righeScritte = insert.InserisciTabellaBis(miaTab[0], miaTab[0].TableName);
```

Ora si effettua una ricerca tra i valori della colonna NumeroCiv della tabella Ubicazione che si riferiscono alla pratica (tramite la colonna pratica) di cui si deve effettuare l'inserimento. Il valore trovato viene inserito nella colonna UbicazioneNumeroCivico della riga associata alla pratica nella tabella temporanea. Poichè nella tabella Ubicazione sono presenti più righe per la stessa pratica, e la via, la località ed il numero civico per una medesima pratica sono sempre gli stessi, mentre cambiano soltanto l'interno ed il piano, si dovranno raggruppare le righe della tabella Ubicazione in base ai valori delle colonne pratica e NumeroCiv. Una volta controllato che i dati non siano ripetuti si inserisce nella riga della tabella temporanea il valore del NumeroCiv estratto dalla tabella Ubicazione e poi si recuperano tramite Select gli ID che hanno nelle tabelle T_InterventoVia… e T_InterventoLocalita… del database le rispettive descrizioni DescrizioneVia e Localita della tabella Ubicazione e si aggiungono alla tabella temporanea rispettivamente nei campi InterventoVia_Pratiche_edili e InterventoLocalita_Pratiche_edili.

```
for (int v=0;v < DsSportUnico.Tables["Ubicazione"].Rows.Count ;v++ )
{
        if ( (int)DtTabella[0].Rows[row][0] ==
(int)DsSportUnico.Tables["Ubicazione"].Rows[v]["pratica"])
        {
//Si inserisce un solo dato per pratica quindi vengono scartate le pratiche i cui dati sono già stati
inseriti
                Boolean trovato=false;
                for (int j=0;j < AlPratica.Count ;j++ )
                {
                        if ( (int)AlPratica[j] ==
(int)DsSportUnico.Tables["Ubicazione"].Rows[v]["pratica"] )
                        {
                                trovato=true;
                                break;
                        }
                }
                if (trovato == false)
                {
// Il dato per la pratica in questione non è ancora stato inserito
                 AlPratica.Add( DsSportUnico.Tables["Ubicazione"].Rows[v]["pratica"] );
                 DtTabella[0].Rows[row][“UbicazioneNumeroCivico”] =
DsSportUnico.Tables["Ubicazione"].Rows[v]["NumeroCiv"];
// Si reperiscono gli ID delle località
                SQL = "SELECT ID FROM
"+conf.TABELLA_INTERVENTOLOCALITA_PRATICHE_EDILI+" WHERE
InterventoLocalita_Pratiche_edili = '"+DsSportUnico.Tables["Ubicazione"].Rows[v]["Località"]+"' ";
                query = new OleDbCommand(SQL,cn);
                query.Connection.Open();
```

```
try {
        reader = query.ExecuteReader();
        if ( reader.Read())
        {
          id= reader.GetInt32(0);
          DtTabella[0].Rows[row][“InterventoLocalita_Pratiche_edili”] = id;
```

Dopo si individua nello stesso modo l’ID della Via, a questo punto nella riga della tabella temporanea per l’inserimento sono presenti tutti i dati della pratica, si può quindi procedere invocando il metodo per l’inserimento e passando a questo come argomenti la tabella temporanea con i dati delle pratiche, il nome della tabella in cui inserire i dati e il numero della riga da inserire.

```
// inserimento di un record in Pratiche_edili

righeScritte=0;
righeScritte=insert.InserisciRiga( DtTabella[0],DtTabella[0].TableName,row);
row++;
//Controllo se l’inserimento è andato a buon fine
if (righeScritte == 1 )
{
//in caso affermativo si effettua una select per recuperare l’ID della pratica appena inserita

SQL = "SELECT ID FROM "+conf.TABELLA_PRATICHE_EDILI+" ORDER BY ID DESC ;";
Object value;
int pratica=0;
query = new OleDbCommand(SQL,cn);
try {
        query.Connection.Open();
        value = query.ExecuteScalar();
        if ( value != null )
        {
                pratica = Convert.ToInt32(value);
        …….
        …….
```

A questo punto si può procedere con l’inserimento dei dati presenti nelle altre tabelle caricate nel **DataSet DsSportUnico** che si riferiscono alla pratica appena inserita partendo dai costi di costruzione contenuti nella tabella **DsSportUnico.Tables[“CostoCostruzione”]**.
Imposto la variabile **inser** su true per specificare che si tratta di un inserimento, dopodiché chiamo il metodo per la creazione della tabella contenente i dati del costo di costruzione associato alla pratica appena inserita e l’ID di tale pratica, se la tabella ritornata non è vuota ne inserisco il contenuto nella tabella T_costo_costruzione del database.

```
inser = true;
miaTab=generator.CreaTabella_costo_costruzione(DsSportUnico,(int)DsSportUnico.Tables["pratic
he"].Rows[i]["chiave"],pratica,inser);
if (miaTab[0].Rows.Count > 0)
{
        int costoLoro = (int)miaTab[0].Rows[0][“ID”];
        righeScritte = insert.InserisciRiga(miaTab[0], miaTab[0].TableName,0);
```

Nel caso in cui l’inserimento sia andato a buon fine recupero l’ID del costo di costruzione appena inserito (come si è fatto per la pratica) memorizzandolo nella variabile costo e procedo con l’inserimento dei dati nelle tabelle associate alla tabella T_costo_costruzione. Per importare i dati vengono prima invocati i metodi che creano le tabelle temporanee contenenti i dati da inserire o con

cui aggiornare le tabella del database e dopo vengono passate queste tabelle ai metodi che effettuano l'INSERT o l'UPDATE. Per alcune tabelle si controlla se i dati relativi al costo di costruzione sono già presenti, in tal caso verranno aggiornati con i nuovi dati.

```
if (righeScritte == 1 )
{
SQL = "SELECT ID FROM "+conf.TABELLA_COSTO_COSTRUZIONE+" ORDER BY ID DESC ;";
….
….
// Si inserisce il legame tra il Costo di Costruzione e la pratica edilizia
// T_Pratiche_edili_costi_costruzione_legame
        miaTab[0]=generator.CreaTabella_Pratiche_edili_costi_costruzione_legame(
        DsSportUnico,pratica,costo );
        if ( miaTab[0].Rows.Count > 0)
        {
                righeScritte = insert.InserisciRigaConID(miaTab[0], miaTab[0].TableName,0);
        }
```

La variabile costo è passata come argomento ai metodo che creano le tabelle temporanee per l'inserimento così da poter controllare nelle tabelle del database se sono già presenti i dati relativi a quel particolare costo di costruzione. Nel caso in cui tali dati siano già presenti si procede all'UPDATE della tabella con i nuovi valori, altrimenti si procede con l'inserimento.
La variabile **costoLoro** contiene l'ID del costo di costruzione nel database da cui sono stati esportati i dati (DsSportUnico.Tables["CostoCostruzione"].Columns["ID"]) associato alla pratica in questione ed è stata utilizzata come Foreign Key per le tabelle Tabella1, Tabella2, Tabella3, Tabella4 del **DataSet DsSportUnico**. Grazie a costoLoro i metodi che creano le tabelle temporanee possono ricavare i record riguardanti il costo in questione.

```
// T_incremento_SupUtile

        miaTab=generator.CreaTabella_incremento_SupUtile(DsSportUnico,costoLoro,costo);
        if ( miaTab[0].Rows.Count > 0)
        {
                righeScritte = insert.InserisciTabella(miaTab[0], miaTab[0].TableName);
        }
        if ( miaTab[1].Rows.Count > 0)
        {
                righeScritte = insert.UpdateTabella(miaTab[1], miaTab[1].TableName);
        }
        …….
        …….
}// if (righeScritte == 1) costo_costruzione

}// if (miaTab[0].Rows.Count > 0) costo_costruzione
```

Una volta inseriti i dati riguardanti il costo di costruzione della pratica si procede inserendo i dati delle colonne Interno e Piano della tabella Ubicazione nelle tabelle T_piani_interni e T_Piano_piani_interni. Ora si può terminare inserendo i dati relativi ai Mappali e ai Soggetti della pratica in questione. I dati anagrafici dei soggetti sono contenuti nella tabella Indice del DataSet, per il loro inserimento viene adottata questa politica: se nella tabella T_dati_anagrafici_soggetti non sono presenti record con valori delle colonne dati_anagrafici_soggetti, CodiceFiscale e DataNascita uguali a quelli presenti nelle colonne Nome, CodiceFisc e DataNascita delle righe della tabella Indice riguardanti la pratica in questione si procede all'inserimento di tali dati, nel caso contrario si procede all'UPDATE della tabella coi nuovi valori. Per tutti i dati anagrafici inseriti o aggiornati

vengono recuperati gli ID dalla tabella T_dati_anagrafici_soggetti, poi tramite il metodo **CreaTabella_soggetti** si crea la tabella temporanea per l'inserimento nella tabella T_soggetti degli ID della pratica, dei soggetti legati alla pratica e dell'ordine di cui fanno parte i soggetti.

```
inser = true;
miaTab[0]=generator.CreaTabella_fogli_mappali(DsSportUnico,(int)DsSportUnico.Tables["pratiche
"].Rows[i]["chiave"],pratica,inser );
if ( miaTab[0].Rows.Count > 0)
{
    righeScritte = insert.InserisciTabella(miaTab[0], miaTab[0].TableName);
 }
// T_dati_anagrafici_soggetti
DataTable DtAnagrafica= new DataTable("ID_dati_Anagrafici");
colonna = new DataColumn("dati_anagrafici_Loro",System.Type.GetType("System.Int32"));
DtAnagrafica.Columns.Add(colonna);
colonna = new DataColumn("dati_anagrafici_Nostri",System.Type.GetType("System.Int32"));
DtAnagrafica.Columns.Add(colonna);

IDdatiAnagrafici=0;
righeScritte=0;

miaTab=generator.CreaTabella_dati_anagrafici_soggetti(DsSportUnico,(int)DsSportUnico.Tables["
pratiche"].Rows[i]["chiave"]);

// dati anagrafici da inserire
for (int j=0; j < miaTab[0].Rows.Count; j++ )
{
     miaRiga=DtAnagrafica.NewRow();
     miaRiga[0]=miaTab[0].Rows[j][“ID”];
     righeScritte += insert.InserisciRiga(miaTab[0], miaTab[0].TableName,j);
    SQL = "SELECT ID FROM "+conf.TABELLA_DATI_ANAGRAFICI_SOGGETTI+" ORDER BY
ID DESC ;";
    query = new OleDbCommand(SQL,cn);
    try {
           query.Connection.Open();
           Object value;
           value = query.ExecuteScalar();
           if ( value != null )
           {
               IDdatiAnagrafici = Convert.ToInt32(value); }
         }
     catch(Exception ex) {  throw new ArgumentException(query.CommandText+" Errore:
"+ex.Message);    }

      finally
      {
          query.Connection.Close();
       }
       miaRiga[1]= IDdatiAnagrafici;
       DtAnagrafica.Rows.Add(miaRiga);
 }

// dati anagrafici da aggiornare
for ( int j=0; j < miaTab[1].Rows.Count ;j++)
{
   miaRiga=DtAnagrafica.NewRow();
```

```
   miaRiga[0]=miaTab[1].Rows[j][0];
   miaRiga[1]=miaTab[1].Rows[j][0];
   DtAnagrafica.Rows.Add(miaRiga);
   righeScritte = insert.UpdateRiga(miaTab[1], miaTab[1].TableName,j);
 }
// T_soggetti
miaTab[0]=generator.CreaTabella_soggetti(DsSportUnico,DtAnagrafica,(int)DsSportUnico.Tables["
pratiche"].Rows[i]["chiave"],pratica);
if (miaTab[0].Rows.Count > 0)
{
   righeScritte=insert.InserisciTabella(miaTab[0],miaTab[0].TableName);
}
```

Quello che è stato appena illustrato è il caso dell'inserimento dei dati riguardanti una nuova pratica, se il campo id della tabella pratiche fosse stato NULL si sarebbe proceduto invece con l'UPDATE della tabella T_Pratiche_edili e di tutte le tabelle ad essa correlate. In questo elaborato non si analizza il caso dell'aggiornamento delle pratiche, viene così terminata la parte inerente l'importazione dei dati nel database.

5.4.4 - RAGGRUPPAMENTO DEI DATI IN TABELLE PER L'INSERIMENTO

Come si è visto nel paragrafo precedente i dati importati dal file XML in un DataSet venivano passati come argomenti a metodi che agendo sulle diverse tabelle importate, a seconda che si tentasse di effettuare un inserimento o un aggiornamento del database con i nuovi valori, ritornavano delle tabelle temporanee contenenti i dati da importare con la stessa struttura e lo stesso nome di quelle del database in cui tali dati venivano importati.
A titolo esplicativo andremo ad analizzare uno dei metodi utilizzati nel paragrafo 5.4.3 quello che si occupa della creazione delle tabelle temporanee contenenti i dati da inserire nella tabella dei costi di costruzione T_costo_costruzione. Il metodo in questione viene definito all'interno della classe CreaTabelle.

```
namespace CreaTabelle_inserimento {

public class CreaTabelle {
// Dichiarazione degli oggetti utilizzati
protected OleDbConnection cn;
protected OleDbCommand query;
protected OleDbDataReader reader;
protected DataTable[] DtTabella;
protected configurazione conf = new configurazione();
……
……
```

Al metodo sono passati come argomenti il **DataSet** in cui è stato caricato il file XML, la variabile **idpraticaNostro** che rappresenta l'ID della pratica nel nostro database, la variabile **idpratica** che rappresenta l'ID della pratica nel database da cui sono stati esportati i dati, questa è utilizzato come Foreign Key nella tabella CostoCostruzione e in altre tabelle del DataSet per implementare i vincoli di integrità referenziale verso la tabella pratiche.

```
public DataTable[] CreaTabella_costo_costruzione(DataSet DsSportello,int idpratica,int idpraticaNostro,Boolean inserimento)
{
DtTabella = new DataTable[2];
DataColumn IdColumn,IdColumn2;
int costoLegame=0;
cn = new OleDbConnection("provider=SQLOLEDB; user id="+conf.utente+";password=”+conf.Pwd+”;server="+conf.nomeServer+";Initial Catalog='"+conf.nomeDatabase+"'");

// Creazione struttura della tabella T_costo_costruzione per l'inserimento
DtTabella[0] = new DataTable("T_costo_costruzione");
IdColumn = new DataColumn("ID",DsSportello.Tables["CostoCostruzione"].Columns["ID"].DataType);
IdColumn.AllowDBNull = false;
DtTabella[0].Columns.Add(IdColumn);
colonna = new DataColumn("SupUtileAttivitaTuristicheCom",DsSportello.Tables["CostoCostruzione"].Columns["SupUtile"].DataType);
DtTabella[0].Columns.Add(colonna);
colonna = new DataColumn("Snr",DsSportello.Tables["CostoCostruzione"].Columns["Snr"].DataType);
DtTabella[0].Columns.Add(colonna);
colonna = new DataColumn("Idue",DsSportello.Tables["Tabella3"].Columns["I2"].DataType);
DtTabella[0].Columns.Add(colonna);
……
……
// Creazione della struttura della tabella T_costo_costruzione per l'update

DtTabella[1] = new DataTable("T_costo_costruzione");
IdColumn2 = new DataColumn("ID",DsSportello.Tables["CostoCostruzione"].Columns["ID"].DataType);
IdColumn2.AllowDBNull = false;
…..
…..
nomeTab="CostoCostruzione";
```

Nel caso in cui si stia eseguendo un inserimento della pratica e dei dati ad essa correlati (inserimento == true) si deve effettuare un INSERT del costo di costruzione nella tabella T_costo_costruzione. Per la creazione della tabella per l’inserimento si procede riempiendo la tabella temporanea con i dati di CostoCostruzione che si riferiscono alla pratica inserita, poi vengono aggiunti i dati presenti nelle tabelle Tabella1 e Tabella3 che si riferiscono al Costo di costruzione da inserire, ed in fine si recupera dalla tabella T_classi_edifici_maggiorazioni l’ID della classe da inserire come Foreign Key nella tabella T_costo_costruzione.

```
if ( inserimento == true )
{
  for (int i=0;i < DsSportello.Tables[nomeTab].Rows.Count ;i++ )
  {
        if ( (int)DsSportello.Tables[nomeTab].Rows[i]["pratica"]  == idpratica )
        {
        DrRiga = DtTabella[0].NewRow();
        DrRiga[0] = DsSportello.Tables["CostoCostruzione"].Rows[i]["ID"];
        DrRiga[1] = DsSportello.Tables["CostoCostruzione"].Rows[i]["SupUtile"];
        DrRiga[2] = DsSportello.Tables["CostoCostruzione"].Rows[i]["Snr"];
```

```
        DrRiga[3] = DsSportello.Tables["CostoCostruzione"].Rows[i]["Rapporto"];
        DrRiga[4] = DsSportello.Tables["CostoCostruzione"].Rows[i]["snr60"];
        DrRiga[5] = DsSportello.Tables["CostoCostruzione"].Rows[i]["Sc"];
        DrRiga[6] = DsSportello.Tables["CostoCostruzione"].Rows[i]["suNonRes"];
        DrRiga[7] = DsSportello.Tables["CostoCostruzione"].Rows[i]["Sa"];
        DrRiga[8] = DsSportello.Tables["CostoCostruzione"].Rows[i]["sa60"];
        DrRiga[9] = DsSportello.Tables["CostoCostruzione"].Rows[i]["St"];
        DrRiga[10] = DsSportello.Tables["CostoCostruzione"].Rows[i]["I"];
        DrRiga[12] = DsSportello.Tables["CostoCostruzione"].Rows[i]["Maggiorazione"];
        DrRiga[13] = DsSportello.Tables["CostoCostruzione"].Rows[i]["A"];
        DrRiga[14] = DsSportello.Tables["CostoCostruzione"].Rows[i]["B"];
        DrRiga[15] = DsSportello.Tables["CostoCostruzione"].Rows[i]["C"];
        DrRiga[16] = DsSportello.Tables["CostoCostruzione"].Rows[i]["D"];
        DrRiga[17] = DsSportello.Tables["CostoCostruzione"].Rows[i]["I3"];
        DrRiga[18] = DsSportello.Tables["CostoCostruzione"].Rows[i]["CostoCostruzione"];

        for (int k = 0;k < DsSportello.Tables["Tabella1"].Rows.Count ;k++ )
        {
           if ( (int)DsSportello.Tables["Tabella1"].Rows[k]["CostoCostruzione"] ==
          (int)DsSportello.Tables["CostoCostruzione"].Rows[i]["ID"] )
          {
            DrRiga[19] = DsSportello.Tables["Tabella1"].Rows[k]["I1"];
            break;
          }
        }
.....
.....
SQL = "SELECT ID FROM "+conf.TABELLA_CLASSI_EDIFICI_MAGGIORAZIONI+" WHERE
classi_edifici_maggiorazioni =
'"+DsSportello.Tables["CostoCostruzione"].Rows[i]["Classe"].ToString()+"' ";

query = new OleDbCommand(SQL,cn);
try {
        query.Connection.Open();
        reader = query.ExecuteReader();
        if ( reader.Read() )
        {
                id=reader.GetInt32(0);
                DrRiga[11] = id;
                reader.Close();
        }
    }
    catch (Exception ex){
        .....
    }
}
DtTabella[0].Rows.Add(DrRiga);
break;
}// if  == idpratica

}// for Rows di CostoCostruzione
}//  if ( inserimento == true )
```

Nel caso in cui inserimento == false si deve tentare l'UPDATE della pratica e delle tabelle ad essa associate, perciò si esegue una select sulla tabella T_Pratiche_edili_costi_costruzione_legame per verificare se la pratica in questione ha già un costo di costruzione associato, in tal caso si riempie la

tabella temporanea per l'aggiornamento di T_costi_costruzione. Nel caso in cui la pratica non abbia ancora un costo di costruzione associato nel database si procede riempiendo la tabella temporanea per l'inserimento con i dati del costo di costruzione (se presente) associato a tale pratica. Al termine il metodo ritorna l'array di tabelle contenente le due tabelle temporanee.

….
….

```
return DtTabella;
}// CreaTabella_costo_costruzione()  fine
```

5.4.5 – METODI DI INSERIMENTO/AGGIORNAMENTO

In questo paragrafo viene esaminato soltanto uno dei metodi di inserimento/aggiornamento per dare una visione di come funziona l'applicazione. Il metodo che andremo ad esaminare riguarda l'inserimento dei dati contenuti in una tabella. Gli argomenti passati al metodo sono una tabella contenente i dati da inserire, le cui colonne hanno gli stessi nomi delle colonne della tabella del database in cui si effettua l'inserimento e il nome della tabella in cui effettuare l'inserimento.
Innanzitutto viene inizializzata una **OleDbConnection** chiamata cn, dopodiché si inizializza un oggetto **OleDbCommand** chiamato insert, a cui viene associata la connessione precedentemente inizializzata e si apre la connessione al database grazie al metodo **Open** di cn. A questo punto per gestire l'inserimento in modo sicuro ed evitare incongruenze all'interno del database si inizializza un oggetto **OleDbTransaction** che rappresenta una transazione sulla connessione precedentemente aperta, invocando il metodo **BeginTransaction** dell'OleDbConnection, poi tramite la proprietà **Transaction** dell'OleDbCommand si imposta la transazione precedentemente inizializzata come transazione in cui eseguire il Command, per finire si inizializza la variabile **numrow** a zero, tale variabile conterrà il numero di righe inserite.

```
protected OleDbTransaction transazione;
protected OleDbConnection cn;
protected OleDbCommand insert;
protected configurazione conf =new configurazione;

public int InserisciTabella(  DataTable DtTabella,string NomeTab ) {

cn = new OleDbConnection("provider=SQLOLEDB; user
id="+conf.utente+";password="+conf.Pwd+";server="+conf.nomeServer+";Initial
Catalog='"+conf.nomeDataBase+"' ");
insert = new OleDbCommand("",cn);
insert.Connection.Open();
transazione = cn.BeginTransaction();
insert.Transaction = transazione;
numrow = 0;
```

A questo punto si crea in modo dinamico la stringa rappresentante il comando INSERT da eseguire tramite un ciclo for su tutte le righe della tabella temporanea, per ogni riga prima si cicla su tutte le colonne della tabella temporanea recuperandone i nomi ed inserendoli nella stringa, poi si recuperano i valori della riga controllando quelli che hanno valori NULL e quelli di tipo String che dovranno essere inseriti tra 'apici' nel comando. Oltre ai controlli sui NULL e sulle stringhe si effettuano dei controlli sul tipo dei dati da inserire, in quanto tipi come date e numeri decimali contengono virgole ed altri caratteri speciali che possono creare problemi durante l'inserimento. Per ovviare tale problema si è scelto di utilizzare dei parametri rappresentati dalla classe **OleDbParameter**, tali parametri devono essere aggiunti all'**OleDbCommand** nello stesso ordine in cui vengono inseriti sotto forma di ? nella stringa rappresentante il comando di INSERT.

L’insieme dei parametri relativi all’oggetto Command si ottiene tramite la proprietà **Parameters**, ed invocando il metodo **Add** i parametri vengono aggiunti al Command. La data e l’ora di aggiornamento e inserimento presenti in tutte le tabelle del database (anche se per motivi di spazio non lo si è specificato nello Schema del database) vengono a loro volta inserite grazie all’uso dei parametri. Una volta che la stringa di inserimento è pronta, la si associa al comando tramite la proprietà **CommandText**, poi si invoca il metodo **ExecuteNonQuery** per eseguire il comando, questo viene eseguito per tutte le righe della tabella.

```
try
{
for (int i=0;i < DtTabella.Rows.Count ;i++ )
{
        SQL = "INSERT INTO "+NomeTab+" (";
        for (int k=0;k < DtTabella.Columns.Count ;k++ )
        {
                SQL += " "+DtTabella.Columns[k].ColumnName+",";
        }
        SQL += "datainserimento,orainserimento,dataUltimaModifica,oraUltimaModifica ) VALUES ( ";
        for (int k=0;k < DtTabella.Columns.Count ;k++ )
        {
        if ( DtTabella.Columns[k].DataType ==  System.Type.GetType("System.String") )
        {
                if ( DtTabella.Rows[i][k] == DBNull.Value )
                {
                        SQL += "NULL ,";
                }
                else
                {
                        SQL += "'"+DtTabella.Rows[i][k].ToString()+"' ,";
                }
        }
        else if ( DtTabella.Columns[k].DataType == System.Type.GetType("System.DateTime") )
        {
                if ( DtTabella.Rows[i][k] == DBNull.Value )
                {
                        SQL += "NULL ,";
                }
                else
                {
                insert.Parameters.Add(new OleDbParameter("@data"+i.ToString()+k.ToString(),
OleDbType.Date, 4, DtTabella.Columns[k].ColumnName)).Value = DtTabella.Rows[i][k];
                SQL += "? ,";
                }
        }
```

vi erano altri controlli sul tipo dei dati da inserire non riportati per motivi di spazio

```
        ……..
        …….
        else
        {
                if ( DtTabella.Rows[i][k] == DBNull.Value )
                {
                        SQL += "NULL ,";
                }
                else
                {
                        SQL += " "+DtTabella.Rows[i][k].ToString()+" ,";
                }
        }

    }// for Columns
```

```
      SQL += "?,?,?,? )";
      insert.Parameters.Add(new OleDbParameter("@datainserimento", OleDbType.Date, 4,
"datainserimento")).Value = DateTime.Now;
    insert.Parameters.Add(new OleDbParameter("@orainserimento", OleDbType.Date, 4,
"orainserimento")).Value = DateTime.Now;
    insert.Parameters.Add(new OleDbParameter("@dataUltimaModifica", OleDbType.Date, 4,
"dataUltimaModifica")).Value = DateTime.Now;
    insert.Parameters.Add(new OleDbParameter("@oraUltimaModifica", OleDbType.Date, 4,
"oraUltimaModifica")).Value = DateTime.Now;
    insert.CommandText = SQL;

    numrow += insert.ExecuteNonQuery();

    OleDbParameterCollection parametrinsert;
    parametrinsert = insert.Parameters;
    parametrinsert.Clear();
}// for Rows
```

Ora si esegue il Commit della transazione e se non ci sono stati problemi i dati verranno inseriti e il metodo ritornerà il numero di righe inserite, mentre se sono state generate delle eccezioni queste saranno intercettate dal Catch all'interno del quale si eseguirà il Rollback della transazione.

```
transazione.Commit();
}// try fine
catch (Exception ex)  {
          transazione.Rollback();
          throw new ArgumentException(insert.CommandText+" Errore: "+ex.Message);
      }
finally   {
          insert.Connection.Close();
      }
return numrow;
}// InserisciTabella()  fine
```

I tipi che possono assumere i parametri sono specificati da **OleDbType**, per un enumerazione dei tipi dei parametri si rimanda all'appendice.

5.4.6 - VISIONE GENERALE DELL'APPLICAZIONE

In questo elaborato si è visto come utilizzare alcuni degli strumenti messia a disposizione da ADO.NET per operare con i database, in particolare si è visto come trasferire dati tra database tramite un file XML. I dati sono stati recuperati eseguendo delle query sul primo database sfruttando gli oggetti della libreria OleDb, per poi essere inseriti in un oggetto DataSet che rappresenta una cache in memoria dei dati recuperati. Si è poi applicata l'integrità dei dati nel DataSet grazie ad oggetti come ForeignKeyConstraint e UniqueConstraint. La classe DataSet ha funzionato da ponte tra il database e l'XML, sfruttando alcuni suoi metodi è stato possibile scrivere i sui dati ed il suo schema in un documento XML; a questo punto è stato possibile trasferire il documento XML, per esempio in un'applicazione Web. Il documento XML è stato poi caricato in un DataSet ricreando lo schema del DataSet da cui era stato esportato, a questo punto è stato possibile riorganizzare i dati delle pratiche edilizie in tabelle temporanee con le quali si è proceduto all'inserimento/aggiornamento del secondo database. Sfruttando la sicurezza mantenuta dal Framework .NET nell'esecuzione di applicazioni Web e la sicurezza fornita dal DBMS per l'accesso ai dati è stato possibile creare un'applicazione distribuita che permetta il trasferimento selettivo di dati tra database mantenendone l'integrità.

6 - INTERPRETE RELAZIONALE PER XML CON SUPPORTO PER INTERROGAZIONI XQUERY

6.1 - INTEGRAZIONE DI XML IN .NET E XQUERY

Il Framework .NET fornisce un ambiente di sviluppo integrato che supporta l'XML grazie ad una serie di classi contenute nel namespace **System.Xml**. Tramite queste classi viene mantenuta l'aderenza agli standard W3C come XML, schemi XSD (utili per convalidare i dati XML), spazio dei nomi, espressioni XPath, livelli 1 e 2 del DOM e trasformazioni XSLT. Oltre a questi standard gli sviluppatori del Framework .NET stanno lavorando ad una libreria contenente le classi per il supporto di XQuery, questo standard W3C è molto utile per eseguire delle query anche complesse su documenti XML. Al momento le classi che implementano XQuery non sono ancora contenute nel Framework .NET, ma è possibile scaricare una demo della Microsoft "xquery.msi" attraverso cui viene fornito un supporto parziale per XQuery. Nella versione rilasciata sono supportate solo alcune delle espressioni definite da XQuery, ma attraverso queste è comunque possibile eseguire una vasta gamma di interrogazioni su documenti XML; in seguito verrà specificato come è stato utilizzato questo strumento all'interno dell'applicazione che verrà descritta in questo capitolo. Verranno ora analizzate alcune tra le classi del Framework .NET che permettono di lavorare con l'XML.
La classe **XmlReader** fornisce un accesso in sola lettura e di tipo forward-only a un flusso di dati XML, il fatto di leggere da un flusso senza dover caricare prima l'intero documento XML in memoria la rende molto veloce. Questa classe legge in modo sequenziale dal documento XML spostandosi in avanti di nodo in nodo, per questo è particolarmente indicata quando si desidera eseguire un singolo passaggio su un documento XML, diventa meno efficace quando si deve navigare all'interno del documento spostandosi in modo non sequenziale tra gli elementi. La classe **XmlValidatingReader** consente di convalidare un documento XML attraverso DTD e schemi XSD. La classe **XmlSchemaCollection** viene utilizzata per caricare nella cache schemi XSD tramite il metodo Add, così da poterli utilizzare con **XmlValidatingReader** per la convalida dei documenti XML. Oltre a caricare in memoria schemi già generati il Framework .NET consente di crearne di nuovi grazie alla classe **XmlSchema**, questa contiene la definizione di uno Schema XSD e tramite i suoi metodi tale schema può essere generato a livello di codice e compilato. Una volta creati gli schemi e i documenti XML, li si può scrivere in un file o in un flusso grazie alla classe **XmlWriter**, questa come **XmlReader** viene implementata da altre classi, per esempio dalla classe **XmlTextWriter** che consente di generare un flusso o un file contenente dati XML conformi agli standard W3C.
I componenti di base del livello 1 e 2 del DOM (Document Object Model) di W3C (vedi (10) http://www.w3.org/TR/REC-DOM-Level-x con x=1,2) sono implementati dalla classe **XmlDocument**. Il modello DOM è una rappresentazione in cache della struttura di un documento XML e consente lo spostamento all'interno di tale documento e la sua modifica; le parti del documento XML sono rappresentate da classi di nodi, tali classi rappresentano elementi, attributi e così via. I nodi sono implementati dalla classe **XmlNode** e costituiscono l'oggetto base della struttura DOM, è importante sottolineare che ogni nodo possiede un unico nodo padre, quello di livello immediatamente superiore. Gli unici nodi privi di padre sono i nodi Document ovvero quelli di primo livello che contengono il documento stesso. La classe **XmlDataDocument** che deriva da **XmlDocument** permette di caricare dati relazionali e di manipolarli come se fossero dati XML mediante il DOM. In questo modo il Framework .NET permette l'accesso sincrono sia alla rappresentazione relazionale dei dati che a quella gerarchica mediante gli oggetti **DataSet** e **XmlDataDocument**; difatti quando si sincronizza un **DataSet** con un **XmlDataDocument**

entrambi gli oggetti operano sullo stesso insieme di dati ed una modifica apportata ad uno dei due oggetti viene riflessa anche sull'altro.
Tramite il metodo **CreateNavigator** della classe **XmlDocument** è possibile creare un **XPathNavigator**, questa classe è basata sul modello di dati XPath e consente di eseguire delle query XPath (vedi (9) raccomandazioni W3C http://www.w3.org/TR/xpath) su qualsiasi archivio dati, per esempio su un documento XML caricato in un **XmlDocument**, oppure su dei dati provenienti da un database e caricati in un **DataSet** che viene poi sincronizzato con un **XmlDataDocument**.
XPathNavigator inoltre supporta XSLT (vedi (11) raccomandazioni W3C http://www.w3.org/TR/xslt).

XQuery è un potente linguaggio per effettuare delle interrogazioni su un documento XML, per la sua capacità espressiva è stato paragonato all'SQL, ma a differenza di esso è stato progettato per eseguire query su un modello di dati gerarchico come l'XML. Le interrogazioni in XQuery sono costituite da un'espressione che legge una sequenza di nodi XML o un singolo valore e ritorna una sequenza di nodi XML o un singolo valore. Nell'applicazione presentata in questo capitolo si è prevalentemente utilizzato XQuery per eseguire interrogazioni su documenti XML con la struttura del documento XML presentato nel paragrafo 5.3.3 o simile, così da poter eseguire delle interrogazioni sui dati esportati dal database. Una trattazione approfondita della sintassi delle interrogazioni XQuery esula dallo scopo di questo elaborato, per un maggiore approfondimento si rimanda al sito del W3C (vedi (8) http://www.w3.org/TR/xquery-use-cases).
Per lo sviluppo dell'applicazione come piattaforma si è utilizzato un PC con sistema operativo Microsoft Windows XP Professional con Service Pack 1 e con IIS v5.1, su cui è installato .NET Framework SDK 1.0 in cui è stato installato il supporto per XQuery "xquery.msi" e si è registrata la DLL Microsoft.Xml.Xquery.dll.

6.2 – OBIETTIVO DELL'APPLICAZIONE

Questa applicazione Web si propone lo scopo di fornire un'interpretazione relazionale di un documento XML con una certa struttura e di eseguire delle query su tale documento. La struttura del documento XML è quella dei documenti creati dal .NET tramite il metodo **WriteXml** invocato da un oggetto **DataSet**. Il documento XML dovrà avere un elemento Root (di livello 0) il cui nome preferibilmente dovrebbe essere "NewDataSet", questo potrà contenere o meno al suo interno uno schema XSD. Gli elementi (di livello 1) figli dell'elemento Root saranno di tipo complesso e rappresenteranno le tabelle nella visione relazionale del documento XML, questi elementi non conterranno né nodi testo né attributi al loro interno, ma solamente altri elementi figli (di livello 2), i quali rappresenteranno le colonne della tabella. Gli elementi di livello 2 conterranno al loro interno i dati veri e propri delle tabelle. Il numero degli elementi di livello 1 con un certo nome sarà uguale al numero di righe della tabella con quel nome. L'applicazione funziona anche se il documento XML ha una struttura leggermente diversa da quella appena specificata, anche se per avere una corretta interpretazione relazionale dell'XML sarebbe meglio mantenere la struttura sopra dichiarata. L'applicazione una volta selezionato il file XML mostra le tabelle che vengono riconosciute in tale file, a questo punto si possono selezionare le tabelle su cui eseguire la query. Una volta selezionate le tabelle si può scegliere tra l'eseguire una query che ritorni il risultato di una funzione di aggregazione eseguita su una colonna di una tabella, o se eseguire una query senza funzioni di aggregazione, anche su più tabelle. Una volta fatta questa scelta e dopo aver selezionato le colonne delle tabelle o la colonna su cui eseguire la funzione di aggregazione si deve eventualmente specificare la clausola where se presente. Nella clausola where per identificare una colonna si deve utilizzare la seguente sintassi "NomeTabella/NomeColonna". A questo punto si eseguirà la query e verranno visualizzati i risultati o gli eventuali errori.

Esempio di documento XML

```
<NewDataSet>
    <pratiche>
        <id>452</id>
        <chiave>601</chiave>
        <codiceintervento>6</codiceintervento>
        <descrizioneintervento>Richiesta di permesso di costruire</descrizioneintervento>
        <OggettoPratica>ristrutturazione ed ampliamento di fabbricato rurale da adibirsi ad uso abitazione e servizi
agricoli</OggettoPratica>
    </pratiche>
    <Soggetti>
        <pratica>601</pratica>
        <tipoSoggetto>Coord.Sicur.in Progett.</tipoSoggetto>
        <Codice>143</Codice>
    </Soggetti>
    <Soggetti>
        <pratica>601</pratica>
        <tipoSoggetto>Istituto</tipoSoggetto>
        <Codice>145</Codice>
    </Soggetti>
</NewDataSet>
```

6.3 – IMPLEMENTAZIONE DELL'APPLICAZIONE

Questa applicazione viene implementata come una pagina ASP.NET con la logica scritta in C#, per poter eseguire delle query XQuery si è dovuto importare e registrare la libreria demo fornita dalla Microsoft Microsoft.Xml.XQuery.dll. In questa libreria sono contenute varie classi, le principali sono:

XQueryNavigator che espone una sorgente dati come un documento XML e permette di navigare all'interno di tale documento, grazie ad una serie di metodi che consentono di spostarsi tra i nodi del documento in questione. Alcuni metodi interessanti sono:

- MoveToRoot che consente di spostare il navigator sul nodo root.
- MoveToFirstChild che sposta il navigator sul primo elemento figlio del nodo corrente, ritorna true se tale operazione ha successo, mentre ritorna false se il nodo corrente non è un elemento o non ha nodi figli.
- MoveToParent che sposta il navigator sul nodo padre del nodo corrente, ritorna true se tale operazione riesce, mentre ritorna false in caso contrario o se ci si trova sul nodo root.
- MoveToNext che sposta il navigator sul nodo seguente dello stesso livello del nodo corrente, ritorna true se tale operazione riesce, in caso contrario o se ci si trova su un nodo attributo o namespace ritorna false.
- ToXml che scrive la rappresentazione XML dell'albero associato al navigator, tale rappresentazione può essere scritta in una stringa o in un TextWriter.

Di seguito vengono elencate alcune proprietà interessanti della classe XQueryNavigator:

- HasChildren che ritorna true se il nodo corrente ha nodi figli.
- NodeType che ritorna l'XPathNodeType corrispondente al tipo del nodo corrente, per esempio Element, Attribute, Text, Root, Comment ecc..
- LocalName che ritorna il nome locale del nodo corrente, cioè il nome del nodo corrente senza il prefisso del namespace.
- Value che per i nodi di tipo Element ritorna i valori concatenati del nodo e di tutti i relativi elementi figlio, per i nodi Text ritorna il loro contenuto, per i nodi Attribute ritorna il valore dell'attributo e così via.

XQueryNavigatorCollection che rappresenta una raccolta di oggetti XQueryNavigatr.

XQueryExpression che incorpora una query XQuery, questa classe presenta un metodo che esegue la query e ritorna un XQueryNavigator associato al risultato della query.
XQueryDocument permette di immagazzinare un archivio dati su cui eseguire delle query XQuery in modo ottimizzato, presenta un metodo per la creazione di un XQueryNavigator sull'archivio immagazzinato.
Andiamo ora ad analizzare l'applicazione, come al solito si importano i namespace utilizzati, poi si dichiarano gli oggetti usati nell'applicazione.

```
using System.Data;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Xml;
using System.Xml.XPath;
using Microsoft.Xml.XQuery;
…..
…..
namespace  Query_su_Xml{

public class paginaXQuery: Page
{
protected DataGrid  tabella =new DataGrid();
protected DataTable  myTable =new DataTable();
protected Panel myPanel;
protected Label LbErrore =new Label();
protected String query,nomeFile,DirCorrente;
protected TextBox where =new TextBox();
protected PlaceHolder PhColonne =new PlaceHolder();
protected ArrayList AlTabelle;
protected ArrayList[] AlColonne;
protected ListBox LbxFile =new ListBox();
protected ListBox LbxTabelle = new ListBox();
protected ListBox LbxFunzioni = new ListBox();
protected ListBox[] LbxColonne;
……
protected XQueryNavigator navigator;

public void Page_Load(object o,EventArgs e)
{
        LbxTabelle.SelectionMode = ListSelectionMode.Multiple;
        ….
        ….
        DirCorrente =Server.MapPath(@"../../XML");
        LbxFile.AutoPostBack =true;
        if (!IsPostBack)
        {
                TrovaFile(DirCorrente,ref LbxFile);
                LbxFunzioni.Items.Add(new ListItem("COUNT ( )","count("));
                LbxFunzioni.Items.Add(new ListItem("AVG ( )","avg("));
                LbxFunzioni.Items.Add(new ListItem("SUM ( )","sum("));
                LbxFunzioni.Items.Add(new ListItem("MAX ( )","max("));
                LbxFunzioni.Items.Add(new ListItem("MIN ( )","min("));
        }
        if (IsPostBack)
        {
                if (LbxFile.SelectedItem != null)
                {
                        if (LbxFile.SelectedItem.Text.Substring(0,4) == "DIR " || LbxFile.SelectedItem.Text
== ".. " )
                        {
```

```
                        TrovaFile(LbxFile.SelectedItem.Value.ToString(),ref LbxFile);
                    }
                    else
                    {
                        Session["pathFile"] =LbxFile.SelectedItem.Value.ToString();
                        nomeFileTb.Text =LbxFile.SelectedItem.Value.ToString();
                    }
                }
            }
        }// Page_Load
    }// classe fine
}// namespace fine
```

Al caricamento della pagina si riempie il **ListBox** contenente le funzioni di aggregazione, poi si imposta la directoy corrente in cui cercare i file XML, dopodiché con l'istruzione **LbxFile.AutoPostBack=true** si indica che si verificherà un postback automatico al server quando l'utente modificherà la selezione del ListBox contenente i file XML tra cui scegliere il file su cui eseguire la query. La funzione **TrovaFile** riempie il **ListBox** che le viene passato con i file XML e le directory contenute nella directory che le viene passata come argomento. Se la pagina è caricata in risposta ad un postback si controlla se l'utente ha selezionato un file XML o un direttorio, nel primo caso il percorso assoluto del file viene memorizzato in una variabile di sessione, mentre nel secondo si invoca la funzione **TrovaFile** sulla directoy selezionata dall'utente. Una volta selezionato il file XML su cui eseguire le query si attende che l'utente richiami la funzione che interpreta in modo relazionale il file XML con la pressione del bottone "Seleziona Tabelle".
Descriviamo ora le funzioni utilizzate nell'applicazione:

La funzione **SelezionaTabelle_query** non fa altro che riempire il **ListBox LbxTabelle** con i nomi degli elementi di livello 1 del documento XML selezionato se presenti. In particolare controlla se l'elemento contenitore che può chiamarsi o meno NewDataSet e il cui nome è memorizzato in una variabile di sessione, presenta degli elementi figli diversi dall'elemento "schema" contenente lo schema XSD del documento XML. Nel caso tali elementi siano presenti vengono inseriti nel ListBox. Nel documento XML d'esempio illustrato nel paragrafo 6.2 tali elementi sono gli elementi <pratiche> e <Soggetti>.

La funzione **SelezionaFunzioni_query** riempie il **ListBox LbxColonneFunz** con le colonne della tabella selezionata e imposta la variabile di sessione "Funz" su true, per specificare che si intende eseguire una query che ritorna i risultati di una funzione di aggregazione. Nella funzione si procede in questo modo, innanzitutto si controlla se vi è almeno una tabella selezionata in **LbxTabelle** (se vi è più di una tabella selezionata si seleziona soltanto la prima della lista, in quanto si è stabilito che la funzione di aggregazione può essere eseguita su un'unica tabella) poi si riempie **LbxColonneFunz** con i nomi degli elementi del documento XML figli degli elementi col nome della tabella selezionata. Tali elementi vengono considerati nell'interpretazione relazionale come le colonne della tabella. Per finire si aggiunge il ListBox ad un controllo **PlaceHolder** della pagina ASP .NET.

La funzione **SelezionaColone_query** imposta la variabile di sessione "Funz" su false per indicare che la query non conterrà funzioni di aggregazione, poi crea tanti oggetti **ListBox** quante sono le tabelle selezionate. Ciascuno di questi oggetti viene riempito con i nomi delle colonne della tabella e un elemento di nome "Tabella" che se selezionato rappresenta la selezione di tutte le colonne della tabella. Le colonne sono identificate dagli elementi figli degli elementi con i nomi delle tabelle. Una volta creati i ListBox li si aggiunge dentro una tabella (non più di quattro per riga) nel controllo PlaceHolder.

La funzione **SelezionaWhere_query** genera la stringa che identifica la query XQuery sulla base delle tabelle, funzioni di aggregazione e colonne selezionate e della clausola Where se presente. Una volta generata la query viene passata alla funzione incaricata di eseguirla, che poi ne visualizza il risultato o gli errori.

```
public void SelezionaWhere_query(object o,EventArgs evt)
{
// contiene la clausola WHERE inserita nel TesxBox where
String expression =where.Text;
String funzSelezionata ="";
String listaColonne="";
String[] Colonne;
NameValueCollection valPagina;
XQueryNavigatorCollection collect = new XQueryNavigatorCollection();
int indice=0;
// stringa rappresentante l'espressione RETURN della query XQuery
string returnStr="";
….
// si recupera il nome del file XML dal percorso assoluto
fileInDirectory =(Session["pathFile"].ToString()).Split('\\');
nomeFile =fileInDirectory[fileInDirectory.Length -1];
if (Convert.ToBoolean(Session["Funz"]) == false )
{
        returnStr="RETURN <Qr>{ ";
// si richiama la funzione che genera le ListBox contenenti le colonne in quanto tali dati non restano
//memorizzati e devono pertanto essere ricreati
        SelezionaColone_query(o,evt);
}
else    if (LbxFunzioni.SelectedItem != null)
        {
                returnStr="RETURN  ";
                SelezionaFunzioni_query(o,evt);
                funzSelezionata =LbxFunzioni.SelectedItem.Value.ToString();
        }
// si riempie l'array con i nomi delle tabelle selezionate
AlTabelle =new ArrayList();
for (int i=0;i < LbxTabelle.Items.Count ;i++ )
{
        if (LbxTabelle.Items[i].Selected)
        {
                AlTabelle.Add(LbxTabelle.Items[i].Value);
        }
}
AlColonne =new ArrayList[AlTabelle.Count];
// tra tutti i controlli del PlaceHolder si selezionano solo i ListBox contenenti le colonne
for (int i=0;i < PhColonne.Controls.Count ;i++ )
{
        if ( PhColonne.Controls[i].GetType().ToString() == "System.Web.UI.WebControls.ListBox" )
        {
                if ( ((ListBox)PhColonne.Controls[i]).Items.Count > 0)
                {
                        AlColonne[indice] =new ArrayList();
                        for (int j=0;j < ((ListBox)PhColonne.Controls[i]).Items.Count ;j++ )
                        {
// il primo elemento dell'array AlColonne sarà il nome della tabella e verrà seguito dai nomi delle colonne
                         AlColonne[indice].Add(((ListBox)PhColonne.Controls[i]).Items[j].Value.ToString());
                        }
                        indice++;
                }
        }
}
```

```
valPagina =Request.Form;
query ="";
// Si inseriscono le espressioni FOR nella query per ogni tabella
//corrisponde alla specificazione delle tabelle nel FROM in SQL
for (int i=0;i < AlTabelle.Count ;i++ )
{
    query +="FOR $a"+i.ToString()+" in
document(\""+nomeFile+"\")/"+Session["RootElement"].ToString()+"/"+AlTabelle[i].ToString()+" ";
}
for (int i=0;i < AlTabelle.Count ;i++ )
{
// si riempie la stringa con tutte le colonne selezionate separate tra loro dal carattere “,”
        listaColonne =valPagina[AlTabelle[i].ToString()];
        if (listaColonne != null )
        {
// l’array di stringhe Colonne viene riempito con i nomi delle colonne selezionate
                Colonne =listaColonne.Split(',');
                if (Colonne.Length > 0)
                {
// Se si è selezionato “Tabella” nel ListBox devono essere scelte tutte le colonne
                        if (Colonne[0] == AlTabelle[i].ToString())
                        {
                                for (int j=0;j < indice ;j++ )
                                {
                                        if ( AlColonne[j][0].ToString() == AlTabelle[i].ToString())
                                        {
// nella clausola where si sostituisce NomeTab/NomeColonna con $variabileTabella/NomeColonna
                                                for (int k=1;k < AlColonne[j].Count ;k++ )
                                                {
                                                expression=expression.Replace(AlTabelle[i].ToString()+"/"+
                                                AlColonne[j][k].ToString(),"$a"+i.ToString()+"/"+AlColonne[j][
                                                k].ToString());
// si aggiunge la colonna alla clausola RETURN corrisponde alla SELECT nomeColonna in SQL
                                                returnStr+="$a"+i.ToString()+"/"+AlColonne[j][k].ToString()+",";
                                                }
                                                break;
                                        }

                                }
                        }
                        else
                        {
                                for (int j=0;j < Colonne.Length ;j++ )
                                {
                                        returnStr +="$a"+i.ToString()+"/"+Colonne[j]+",";
                                }
                                for (int j=0;j < indice ;j++ )
                                {
                                        if ( AlColonne[j][0].ToString() == AlTabelle[i].ToString())
                                        {
                                                for (int k=1;k < AlColonne[j].Count ;k++ )
                                                {
                                                expression=expression.Replace(AlTabelle[i].ToString()+"/"+
                                                AlColonne[j][k].ToString(),"$a"+i.ToString()+"/"+AlColonne[j][
                                                k].ToString());
                                                }
                                                break;
                                        }

                                }
                        }
```

```
                }// if (Colonne.Length > 0)
        }
}
if ( expression == "")
{
        query +=" "+returnStr;
        query =query.Substring(0,(query.Length - 1));

        if (Convert.ToBoolean(Session["Funz"]) == false)
        {
                query +=" }</Qr>";
        }
}
else
{
// se la clausola WHERE è presente la si aggiunge alla query
        query +=" WHERE "+expression+" "+returnStr;
        query =query.Substring(0,(query.Length - 1));
        if (Convert.ToBoolean(Session["Funz"]) == false)
        {
                query +=" }</Qr>";
        }
}
// Se la query contiene una funzione di aggregazione la struttura del doc XML ritornato sarà diversa
if ( Convert.ToBoolean(Session["Funz"]) == true)
{
        if (LbxFunzioni.SelectedItem != null)
        {
                query ="<CONTENITORE_DATASET>{ <RESULT_QUERY>{ "+funzSelezionata+"
"+query+" ) } </RESULT_QUERY> }</CONTENITORE_DATASET>";
        }
}
// si richiama la funzione che esegue la query passandogli il testo della query, il percorso assoluto ed il nome
// del fileXML
GeneraData(query,Session["pathFile"].ToString(),nomeFile);

}// SelezionaWhere_query
```

La funzione **GeneraData** esegue la query, dopodiché se non sono stati segnalati errori si invoca la funzione che crea la struttura della tabella temporanea in cui inserire i risultati della query; una volta creata la tabella si invoca la funzione che la riempie con i dati risultanti dalla query. Alla fine tale tabella viene indicata come sorgente dati per un oggetto **DataGrid** tramite il quale i dati vengono visualizzati nella pagina ASP .NET. Per sintetizzare si illustra soltanto come la query viene eseguita.

```
XQueryNavigator navigator;
XQueryNavigatorCollection collection = new XQueryNavigatorCollection();
// Si aggiunge alla collezione un nuovo navigator sul file col nome fileName
// identificato dal path Session[“pathFile”]
collection.AddNavigator(Session["pathFile"].ToString(),fileName);
// si esegue la query memorizzata nella stringa richiesta
XQueryExpression interrogazione = new XQueryExpression(richiesta);
try {
// si esegue la query e viene ritornato un oggetto XQueryNavigator che punta al documento XML risultato
// dell’interrogazione XQuery
        navigator =interrogazione.Execute(collection);
        ….
        ….
    }
```

La funzione **NavigaNodi** grazie all'oggetto **XQueryNavigator** che punta al documento XML risultato della interrogazione XQuery crea la struttura della tabella temporanea. Sfruttando i metodi messi a disposizione dalla classe **XQueryNavigator** ci si può spostare tra i nodi del documento XML, in questo modo vengono aggiunte alla tabella le colonne identificate dagli elementi di tipo semplice contenenti i dati. Questi elementi non contengono altri elementi al loro interno, ma soltanto nodi Text, perciò si possono vedere come i nodi foglia dell'albero rappresentato dal documento XML.

La funzione **RiempiTabella** grazie all'**XQueryNavigator** naviga tra i nodi del documento XML e riempie le righe della tabella temporanea con i valori dei nodi foglia. Tale operazione è ripetuta per tutti gli elementi contenenti nodi foglia. La struttura dei documenti XML ritornati dalle query XQuery è di questi due tipi:

Documento XML risultante da query senza funzioni di aggregazione

```
<Qr>
   <NomeColonna1>Valore</NomeColonna1>
   <NomeColonna2>Valore</NomeColonna2>
   ……
   <NomeColonnaN>Valore</NomeColonnaN>
</Qr>
```

Documento XML risultante da query con funzione di aggregazione

```
<CONTENITORE_DATASET>
   <RESULT_QUERY>Risultato funzione di aggregazione</RESULT_QUERY>
</CONTENITORE_DATASET>
```

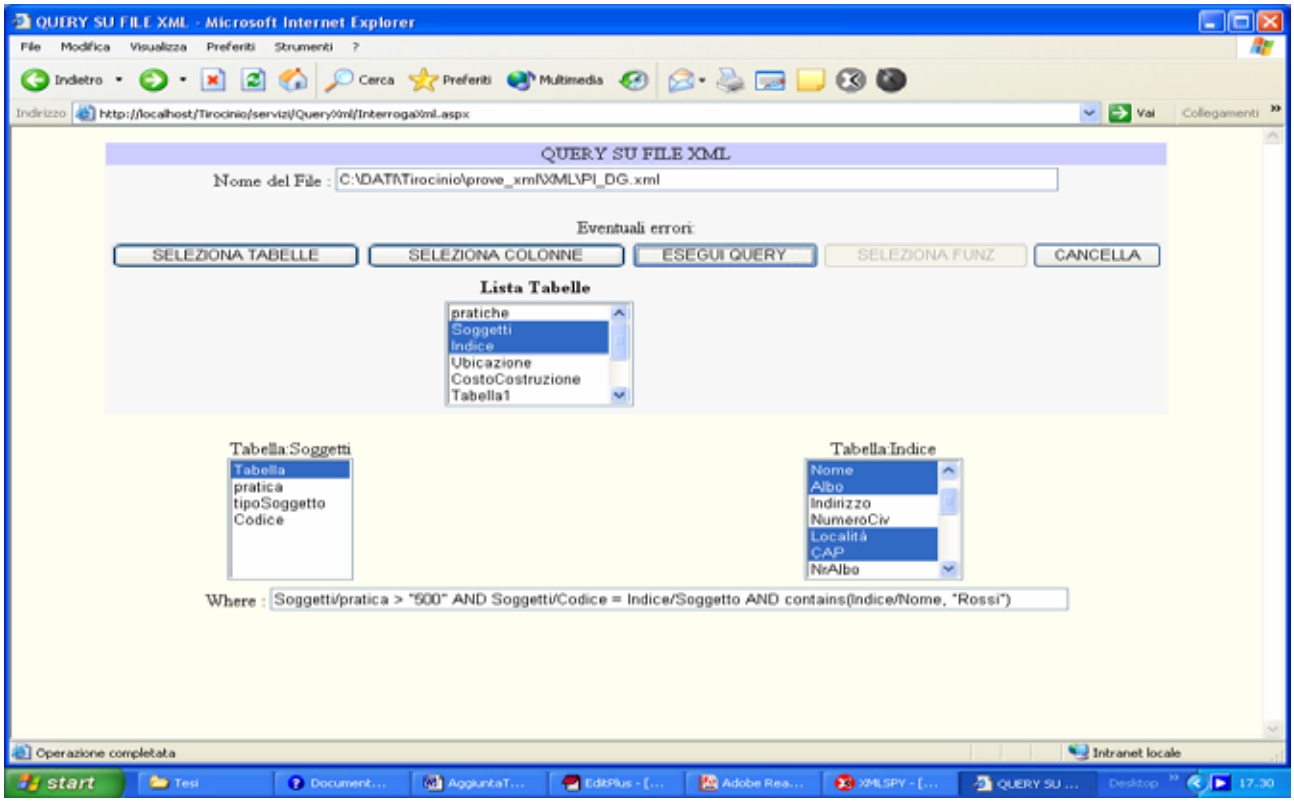

figura 5 Pagina ASP .NET che esegue query su documento XML: definizione query

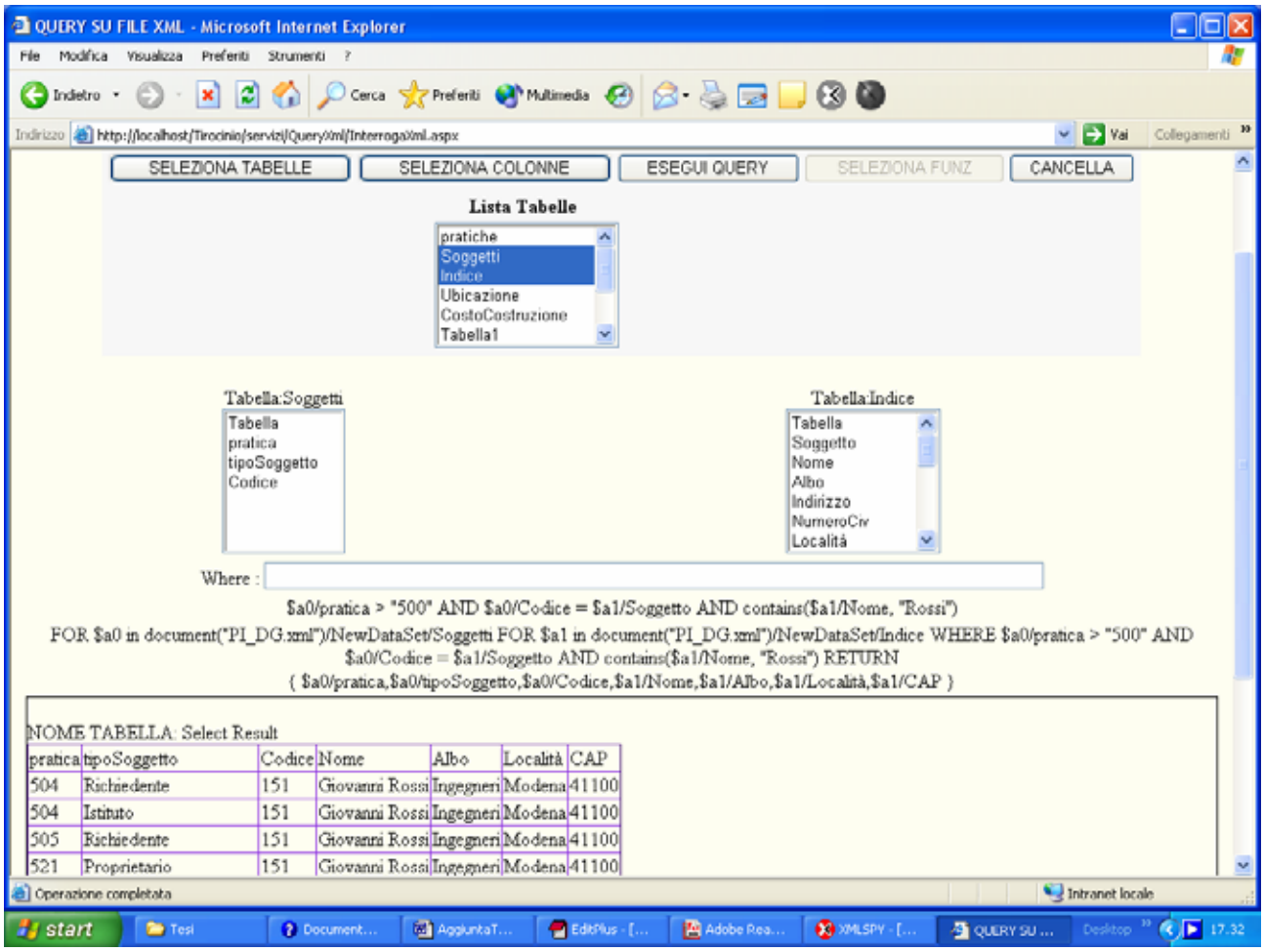

figura 6 Pagina ASP .NET che esegue query su documento XML: risultato query

CONCLUSIONE

In questo elaborato si è analizzato come trasferire in modo selettivo dei dati tra differenti database, installati su server diversi che possono trovarsi in luoghi fisicamente distanti. Per trasferire i dati è stato utilizzato un documento XML validato mediante uno schema XSD, grazie a questa tecnologia è stato possibile mantenere l'integrità dei dati trasferiti e le relazioni tra di essi. XML Schema Definition presenta numerosi tipi di dati semplici, grazie a questi, uniti alle facets ad essi applicabili nei tipi complessi è possibile mantenere il tipo dei dati esportati tra i database, mantenendo così anche il range di valori che i dati possono assumere. Oltre a questo è stato possibile mantenere i vincoli di chiavi primarie, unicità ed integrità referenziale tra i dati esportati per mezzo di vari strumenti messi a disposizione dagli Schemi XSD come ad esempio keyref che implementa i vincoli di integrità referenziale. Oltre ai vari vincoli è importante la possibilità fornita da XSD Schema di mantenere informazioni sui valori nulli, identificati dall'assenza dell'elemento o da un elemento vuoto con l'attributo nil="true". Per operare con i database e i documenti XML sono stati utilizzati gli strumenti messi a disposizione dal Framework .NET, grazie alla libreria ADO .NET che fornisce numerosi strumenti per eseguire istruzioni su tutti gli RDBMS generalmente utilizzati (come Oracle o SQL Server) è stato semplice operare con i database mantenendo sicurezza ed integrità dei dati. Il Framework .NET ha fornito anche numerosi strumenti per operare sui documenti XML, in particolare è stata abbastanza importante la libreria demo Microsoft.Xml.XQuery.dll fornita dalla Microsoft che implementa la tecnologia XQuery. Questa libreria essendo una demo non fornisce un supporto completo per XQuery, comunque sono state implementate tutte le espressioni più comunemente utilizzate e grazie ad esse è stato possibile generare tutte le query più frequentemente eseguite sul documento XML, riuscendo così ad avere a disposizione uno strumento potente quanto l'SQL 92 per eseguire delle query sull'interpretazione relazionale del documento XML contenente una selezione dei dati esportati dal database. Si è visto così anche come utilizzare un documento XML non solo come mezzo per esportare dati da database, ma come un piccolo database che mantiene i vincoli tra i dati che lo compongono e su cui è possibile eseguire delle interrogazioni. Non sono state implementate le transazioni e le Stored Procedure, inoltre le operazioni sul documento XML sono più lente di quelle eseguite da un RDBMS sulle sue tabelle, ma per piccole applicazioni Webdatabase che non devono gestire una grossa mole di dati ed accedono al database solo come fonte di dati è possibile utilizzare i documenti XML come piccoli database tramite applicazioni implementate con gli strumenti forniti dal Framework .NET.

APPENDICE

Queste tabelle rappresentano un elenco delle **Facets** messe a disposizione da XSD e dei tipi semplici a cui possono essere applicate.

Simple Types	Facets					
	length	**minLength**	**maxLength**	**pattern**	**enumeration**	**whiteSpace**
string	SI	SI	SI	SI	SI	SI
normalizedString	SI	SI	SI	SI	SI	SI
token	SI	SI	SI	SI	SI	SI
byte				SI	SI	SI
unsignedByte				SI	SI	SI
base64Binary	SI	SI	SI	SI	SI	SI
hexBinary	SI	SI	SI	SI	SI	SI
integer				SI	SI	SI
positiveInteger				SI	SI	SI
negativeInteger				SI	SI	SI
nonNegativeInteger				SI	SI	SI
nonPositiveInteger				SI	SI	SI
int				SI	SI	SI
unsignedInt				SI	SI	SI
long				SI	SI	SI
unsignedLong				SI	SI	SI
short				SI	SI	SI
unsignedShort				SI	SI	SI
decimal				SI	SI	SI
float				SI	SI	SI
double				SI	SI	SI
boolean				SI		SI
time				SI	SI	SI
dateTime				SI	SI	SI
duration				SI	SI	SI
date				SI	SI	SI
gMonth				SI	SI	SI
gYear				SI	SI	SI
gYearMonth				SI	SI	SI
gDay				SI	SI	SI
gMonthDay				SI	SI	SI
Name	SI	SI	SI	SI	SI	SI
QName	SI	SI	SI	SI	SI	SI
NCName	SI	SI	SI	SI	SI	SI
anyURI	SI	SI	SI	SI	SI	SI
language	SI	SI	SI	SI	SI	SI
ID	SI	SI	SI	SI	SI	SI
IDREF	SI	SI	SI	SI	SI	SI
IDREFS	SI	SI	SI		SI	SI
ENTITY	SI	SI	SI	SI	SI	SI
ENTITIES	SI	SI	SI		SI	SI
NOTATION	SI	SI	SI	SI	SI	SI
NMTOKEN	SI	SI	SI	SI	SI	SI

NMTOKENS	SI	SI	SI		SI	SI

Simple Types	Facets					
	max Inclusive	max Exclusive	min Inclusive	min Exclusive	totalDigits	fractionDigits
byte	SI	SI	SI	SI	SI	SI
unsignedByte	SI	SI	SI	SI	SI	SI
integer	SI	SI	SI	SI	SI	SI
positiveInteger	SI	SI	SI	SI	SI	SI
negativeInteger	SI	SI	SI	SI	SI	SI
nonNegativeInteger	SI	SI	SI	SI	SI	SI
nonPositiveInteger	SI	SI	SI	SI	SI	SI
int	SI	SI	SI	SI	SI	SI
unsignedInt	SI	SI	SI	SI	SI	SI
Long	SI	SI	SI	SI	SI	SI
unsignedLong	SI	SI	SI	SI	SI	SI
short	SI	SI	SI	SI	SI	SI
unsignedShort	SI	SI	SI	SI	SI	SI
decimal	SI	SI	SI	SI	SI	SI
float	SI	SI	SI	SI		
double	SI	SI	SI	SI		
time	SI	SI	SI	SI		
dateTime	SI	SI	SI	SI		
duration	SI	SI	SI	SI		
date	SI	SI	SI	SI		
gMonth	SI	SI	SI	SI		
gYear	SI	SI	SI	SI		
gYearMonth	SI	SI	SI	SI		
gDay	SI	SI	SI	SI		
gMonthDay	SI	SI	SI	SI		

Utilizzando gli oggetti forniti dal Framework .NET e importando dati da documenti XML convalidati tramite schemi XSD, si deve mantenere la compatibilità tra i tipi del .NET e i tipi XSD Schema. Per questo di seguito è fornita una tabella che mostra in che tipi del Framework .NET vengono codificati i tipi XSD Schema.

Tipo XML Schema (XSD)	Tipo .NET Framework
anyURI	System.Uri
base64Binary	System.Byte[]
Boolean	System.Boolean
Byte	System.SByte
Date	System.DateTime
dateTime	System.DateTime

decimal	System.Decimal
Double	System.Double
duration	System.TimeSpan
ENTITIES	System.String[]
ENTITY	System.String
Float	System.Single
gDay	System.DateTime
gMonthDay	System.DateTime
gYear	System.DateTime
gYearMonth	System.DateTime
hexBinary	System.Byte[]
ID	System.String
IDREF	System.String
IDREFS	System.String[]
int	System.Int32
integer	System.Decimal
language	System.String
long	System.Int64
month	System.DateTime
Name	System.String
NCName	System.String
negativeInteger	System.Decimal
NMTOKEN	System.String
NMTOKENS	System.String[]
nonNegativeInteger	System.Decimal
nonPositiveInteger	System.Decimal
normalizedString	System.String
NOTATION	System.String
positiveInteger	System.Decimal
QName	System.Xml.XmlQualifiedName
short	System.Int16
string	System.String
time	System.DateTime
timePeriod	System.DateTime
token	System.String
unsignedByte	System.Byte
unsignedInt	System.UInt32
unsignedLong	System.UInt64
unsignedShort	System.UInt16

Dovendo trasferire dati tra RDBMS utilizzando gli strumenti messi a disposizione dal Framework .NET si deve mantenere la compatibilità tra i tipi del RDBMS ed i tipi del Framework .NET. Per questo motivo si fornisce una tabella che descrive la codifica proprietaria del Framework per i tipi di dato di SQL Server.

Server SQL nativo	SqlDbType .NET Framework	Tipo .NET Framework e Descrizione
Bigint	**BigInt**	**Int64** Un valore integer con segno a 64 bit.

binari	**Binary**	**Array di tipo Byte** Flusso di dati binari di lunghezza fissa compresi tra 1 e 8.000 byte.
bit	**Bit**	**Boolean** Valore numerico senza segno che può essere 0, 1 o riferimento Null (**Nothing** in Visual Basic).
Char	**Char**	**String** Flusso di caratteri non Unicode di lunghezza fissa compresi tra 1 e 8.000 caratteri.
datetime	**DateTime**	**DateTime** Dati relativi a data e ora compresi nell'intervallo tra il 1° gennaio 1753 e il 31 dicembre 9999 con un'approssimazione di 3,33 millisecondi.
decimal	**Decimal**	**Decimal** Valore numerico a precisione fissa e in scala compreso tra -10^{38} -1 e 10^{38} -1.
Float	**Float**	**Double** Numero a virgola mobile compreso nell'intervallo da –1,79E +308 a 1,79E +308.
image	**Image**	**Array di tipo Byte** Flusso di dati binari di lunghezza variabile compresi tra 0 e 2^{31} -1 (o 2.147.483.647) byte.
Int	**Int**	**Int32** Un valore integer con segno a 32 bit.
Money	**Money**	**Decimal** Valore di valuta compreso nell'intervallo tra -2^{63} (o -922.337.203.685.477,5808) e 2^{63} -1 (o +922.337.203.685.477,5807) con un'approssimazione pari a dieci millesimi di un'unità di valuta.
nchar	**NChar**	**String** Flusso di caratteri Unicode di lunghezza fissa compresi tra 1 e 4.000 caratteri.
Ntext	**NText**	**String** Flusso di dati Unicode di lunghezza variabile con una lunghezza massima di 2^{30} - 1 (o 1.073.741.823) caratteri.
nvarchar	**NVarChar**	**String** Flusso di caratteri Unicode di lunghezza variabile compresi tra 1 e 4.000 caratteri. **Nota** La conversione implicita non riesce se la stringa è maggiore di 4.000 caratteri. Impostare in modo esplicito l'oggetto quando si utilizzano stringhe maggiori di 4.000 caratteri.
Real	**Real**	**Single** Numero a virgola mobile compreso nell'intervallo da –3,40E +38 a 3,40E +38.
smalldatetime	**SmallDateTime**	**DateTime** Dati relativi a data e ora compresi nell'intervallo tra il 1° gennaio 1900 e il 6 giugno 2079 con un'approssimazione di un minuto.
smallint	**SmallInt**	**Int16** Un valore integer con segno a 16 bit.

smallmoney	**SmallMoney**	**Decimal** Valore di valuta compreso nell'intervallo tra –214.748,3648 e +214.748,3647 con un'approssimazione pari a dieci millesimi di un'unità di valuta.
text	**Text**	**String** Flusso di dati non Unicode di lunghezza variabile con una lunghezza massima di 2^{31} -1 (o 2.147.483.647) caratteri.
timestamp	**Timestamp**	**Array di tipo Byte** Numeri binari generati automaticamente, di cui è garantita l'univocità all'interno di un database. **timestamp** è in genere utilizzato come meccanismo per indicare la versione delle righe di tabella. Le dimensioni di archiviazione sono 8 byte.
tinyint	**TinyInt**	**Byte** Valore integer senza segno a 8 bit.
uniqueidentifier	**UniqueIdentifier**	**Guid** Identificatore univoco globale o GUID.
varbinary	**VarBinary**	**Array di tipo Byte** Flusso di dati binari di lunghezza variabile compresi tra 1 e 8.000 byte. **Nota** La conversione implicita non riesce se la matrice di byte è maggiore di 8.000 byte. Impostare in modo esplicito l'oggetto quando si utilizzano matrici di byte maggiori di 8.000 byte.
varchar	**VarChar**	**String** Flusso di caratteri non Unicode di lunghezza variabile compresi tra 1 e 8.000 caratteri.
sql_variant	**Variant**	**Object** Speciale tipo di dati in cui possono essere contenuti dati numerici, stringhe, dati binari o di data, nonché valori Empty e Null di SQL Server, che vengono utilizzati se nessun altro tipo viene dichiarato.

BIBLIOGRAFIA

(1) Titolo: Progetto di Basi di Dati Relazionali “lezioni ed esercizi”
Autore/i: Domenico Beneventano, Sonia Bergamaschi, Maurizio Vincini
Editore: Pitagora Editrice Bologna

(2) Titolo: Programmazione in Visual C# .NET
Autore/i: Harold Davis
Editore: McGrawHill

(3) “Visual C# .NET”
http://msdn.microsoft.com/vcsharp

(4) “Dispense corso Tecnologia Database”
Teoria_XMLNEW.pdf

(5) “XML Schema Part 0”
http://www.w3.org/TR/xmlschema-0

(6) “XML Schema Part 1”
http://www.w3.org/TR/xmlschema-1

(7) “XML Query Requirements”
http://www.w3.org/TR/xmlquery-req

(8) “XML Query Use Cases”
http://www.w3.org/TR/xquery-use-cases

(9) “XML Path Language (XPath)”
http://www.w3.org/TR/xpath

(10) “Document Object Model (DOM)”
http://www.w3.org/TR/REC-DOM-Level-x con {x=1,2}

(11) “XSLT”
http://www.w3.org/TR/xslt