



ADO.NET

- Tutte le applicazioni moderne hanno bisogno di accedere ad un qualche tipo di "base di dati"
 - Database server (SQL Server, Oracle, DB2, ecc.)
 - Documenti XML
 - File di testo (txt, csv, ecc.)
 - File indicizzati (.mdb, .dbf, ecc.)
- La modalità di accesso ai dati dovrebbe essere il più indipendente possibile
 - dal tipo di base di dati
 - dal tipo di applicazione
 - Applicazioni Windows
 - Pagine web dinamiche
 - Web Service

Laboratorio di Ingegneria del Software L-A

8.2

ADO.NET

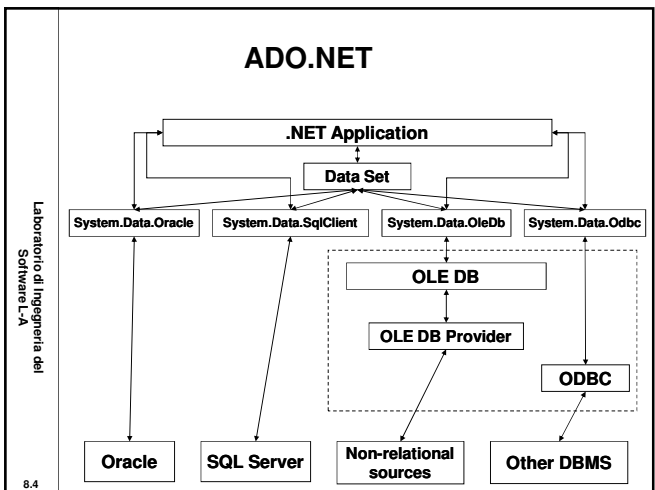
- ActiveX Data Objects
- Insieme di tipi (classi, strutture, interfacce, ...) divisi, dal punto di vista funzionale, in due gruppi fondamentali
 - **Data Set** – tipi che gestiscono la rappresentazione dei dati
 - **Data Provider** – tipi che gestiscono la comunicazione con un *data store*

```

    graph TD
      DS[Data Store] --> DP[Data Provider]
      DP --> DS
      DP --> DS_2[Data Set]
      DS_2 --> DP
      DS_2 --> DC[Data Consumer]
      DC --> DS_2
  
```

Laboratorio di Ingegneria del Software L-A

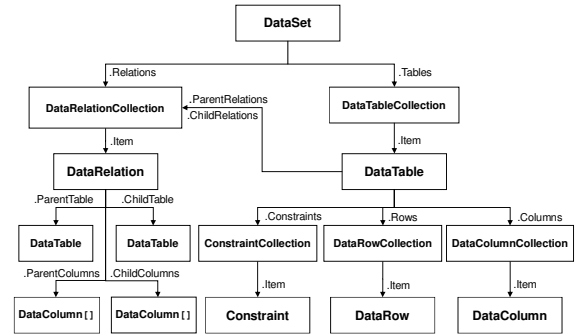
8.3



Data Set

- Permette di gestire i dati come se si disponesse di un semplice database relazionale
 - Residente in memoria di lavoro
 - Disconnesso dalla sorgente dati esterna
- Comprende vari tipi di oggetti:
 - **DataSet** – rappresenta il database in memoria e comprende:
 - Un insieme di tabelle
 - Un insieme di relazioni tra tabelle
 - **DataTable** – rappresenta una tabella
 - **DataColumn** – rappresenta una colonna di una tabella
 - **DataRow** – rappresenta una riga di una tabella
 - **Constraint** – rappresenta un vincolo di integrità su una tabella
 - **DataRelation** – rappresenta una relazione tra due tabelle

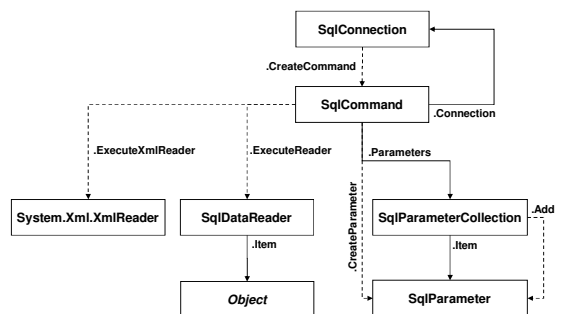
Data Set Object Model

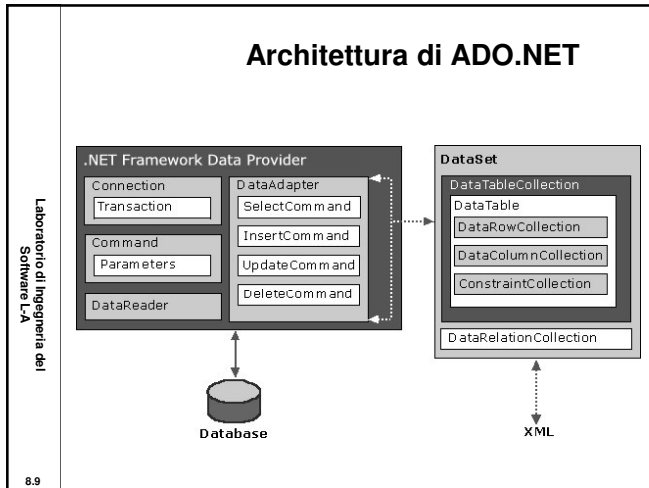


Data Provider

- Permette di ottenere/inviare dati da/a un qualsiasi tipo di sorgente dati
 - Relazionale
 - XML
 - *Custom*
- Specifico per una sorgente dati
- Comprende vari tipi di oggetti:
 - **Connection** – rappresenta la connessione fisica con la sorgente dati
 - **Command** – rappresenta un comando da eseguire sulla sorgente dati (ad es. un'istruzione SQL)
 - **DataReader** – permette di ottenere in modo veloce, *read-only* e *forward-only* i dati dalla sorgente dati
 - **DataAdapter** – fornisce il meccanismo di interazione tra la connessione e il data set

SqlClient .NET Data Provider Object Model





- System.Data.DataSet**
- Non dipende dalla sorgente dati
 - Le sue tabelle
 - possono essere popolate con dati provenienti da sorgenti differenti
 - utilizzano tipi di dati .NET, indipendenti dalla sorgente
 - Può essere costruito
 - completamente in automatico
 - completamente via codice
 - parte in automatico e parte via codice
 - Può essere utilizzato come *data store* indipendente
- Laboratorio di Ingegneria del Software L-A
- 8.10

- System.Data.DataSet**
- Può essere **"typed"** o **"untyped"**
 - Nel caso di *untyped DataSet*, si scrive:
 - `Biblioteca.Tables["Libri"].Rows[0]["Id"]`
 - Nel caso di *typed DataSet*, si può anche scrivere:
 - `Biblioteca.Libri[0].Id`
 - Per creare un *untyped DataSet*:
 - Drag dalla toolbox del DataSet
 - Inserimento delle tabelle
 - Per creare un *typed DataSet*:
 - Add new item → Data Set
 - Drag dalla toolbox di un Element per ogni tabella
- Laboratorio di Ingegneria del Software L-A
- 8.11

- System.Data.DataTable**
- Il vero contenitore dei dati in memoria
 - La sua struttura è descritta da una collezione di **DataColumn**
 - proprietà `Columns`
 - I suoi dati sono contenuti in una collezione di **DataRow**
 - proprietà `Rows`
 - Consente la definizione di una chiave primaria, composta da una o più colonne
 - proprietà `PrimaryKey`
 - Consente la creazione di colonne calcolate
 - proprietà `Expression` di `DataColumn`
- Laboratorio di Ingegneria del Software L-A
- 8.12

System.Data.DataTable

```
// Creazione di una nuova DataTable
DataTable table = new DataTable("Libri");
// Inserimento della DataTable nel DataSet
Biblioteca.Tables.Add(table);
// Definizione della struttura della DataTable
DataColumn columnId = new DataColumn("Id", typeof(int));
table.Columns.Add(columnId);
table.Columns.Add("Titolo", typeof(string));
table.PrimaryKey = new DataColumn[] { columnId };
// Aggiunta di una riga alla DataTable
DataRow dataRow = table.NewRow();
dataRow["Id"] = 1;
dataRow["Titolo"] = "Primo libro";
table.Rows.Add(dataRow);
```

Laboratorio di Ingegneria del Software L-A

8.13

System.Data.DataTable

- **UniqueConstraint** consente di definire **vincoli di univocità** su una o più colonne
 - `dt.Constraints.Add(new UniqueConstraint(dt.Columns[...]))`;
- **ForeignKeyConstraint** consente di definire **foreign key** su una o più colonne
 - `dt.Constraints.Add(new ForeignKeyConstraint(dtmaster.Columns[...], dt.Details.Columns[...]))`;
 - Le proprietà `DeleteRule` e `UpdateRule` consentono di ottenere la *Cascading Referential Integrity*
 - La proprietà `AcceptRejectRule` consente di propagare le modifiche fatte su una tabella "master" a una o più "details"

Laboratorio di Ingegneria del Software L-A

8.14

System.Data.DataColumn

DataColumn
+ <property> AllowDBNull : System.Boolean
+ <property> AutoIncrement : System.Boolean
+ <property> AutoIncrementSeed : System.Int64
+ <property> AutoIncrementStep : System.Int64
+ <property> Caption : System.String
+ <property>.ColumnName : System.String
+ <property> DataType : System.Type
+ <property> DefaultValue : System.Object
+ <property> Expression : System.String
+ <property> MaxLength : System.Int32
+ <property> Ordinal : System.Int32
+ <property> ReadOnly : System.Boolean
+ <property> Table : System.Data.DataTable
+ <property> Unique : System.Boolean
+ DataColumn ()
+ DataColumn ([in] columnName : System.String)
+ DataColumn ([in] columnName : System.String , [in] dataType : System.Type)
+ DataColumn ([in] columnName : System.String , [in] dataType : System.Type , [in] expr : System.String)

Laboratorio di Ingegneria del Software L-A

8.15

System.Data.DataRow

DataRow
+ <property> RowError : System.String
+ <property> RowState : System.Data.DataRowState
+ <property> Table : System.Data.DataTable
+ <property> Item(System.Int32) : System.Object
+ <property> Item(System.String) : System.Object
+ <property> Item(System.Data.DataColumn) : System.Object
+ <property> Item(System.Int32,System.Data.DataRowVersion) : System.Object
+ <property> Item(System.String,System.Data.DataRowVersion) : System.Object
+ <property> Item(System.Data.DataColumn,System.Data.DataRowVersion) : System.Object
+ <property> ItemArray : System.Object[]
+ <property> HasErrors : System.Boolean
DataRow ([in] builder : System.Data.DataRowBuilder)
+ Delete ()
+ GetColumnsInError () : System.Data.DataColumn[]
+ GetChildRows ([in] relationName : System.String) : System.Data.DataRow[]
+ GetChildRows ([in] relation : System.Data.DataRelation) : System.Data.DataRow[]
+ GetParentRow ([in] relationName : System.String) : System.Data.DataRow
+ GetParentRow ([in] relation : System.Data.DataRelation) : System.Data.DataRow
+ GetParentRows ([in] relationName : System.String) : System.Data.DataRow[]
+ GetParentRows ([in] relation : System.Data.DataRelation) : System.Data.DataRow[]

Laboratorio di Ingegneria del Software L-A

8.16

Creazione di un DataSet da codice

- Il **DataSet** è ottimo per costruire strutture dati in grado di mantenere in cache locale le informazioni
 - Creo il **DataSet**

```
DataSet custDS = new DataSet("CustomerOrders");
```
 - Creo una o più **DataTable**

```
DataTable ordersTable = new DataTable("Orders");
ordersTable.Columns.Add("OrderID", typeof(Int32));
ordersTable.Columns.Add("Quantity", typeof(Int32));
ordersTable.Columns.Add("CustID", typeof(Int32));
ordersTable.PrimaryKey =
    new DataColumn[] {ordersTable.Columns["OrderID"]};
...
```
 - Aggiungo al **DataSet** le **DataRelation** tra le tabelle

```
custDS.Relations.Add("CustOrders",
    custDS.Tables["Customers"].Columns["CustID"],
    custDS.Tables["Orders"].Columns["CustID"]);
```

Laboratorio di Ingegneria del Software-LA

8.17

DataRelation e navigazione tra i dati

- Le **DataRelation**, oltre a gestire l'integrità referenziale, consentono di navigare tra i dati
- Da una **DataRow** di una tabella è possibile ottenere le righe della tabella collegata mediante i metodi
 - **GetChildRows**
 - **GetParentRow**
 - **GetParentRows**

```
DataRelation dataRel = custDS.Relations.Add("CustOrders",
    custDS.Tables["Customers"].Columns["CustID"],
    custDS.Tables["Orders"].Columns["CustID"]);
foreach (DataRow custRow in custDS.Tables["Customers"].Rows)
{
    Console.WriteLine(custRow["CustID"]);
    foreach (DataRow orderRow in custRow.GetChildRows(dataRel))
        Console.WriteLine(orderRow["OrderID"]);
}
```

Laboratorio di Ingegneria del Software-LA

8.18

Modifiche ai dati

- Ogni **DataRow** ha uno stato che è interrogabile mediante la proprietà **RowState**
- **DataRow newRow = dataTable.NewRow();** crea una nuova **DataRow** in uno stato **Detached**
- **dataTable.Add(dataRow);** modifica lo stato di **dataRow** in **Added**
- **dataRow["Name"] = "Pippo";** modifica lo stato di **dataRow** in **Modified**
- **dataRow.Delete();** modifica lo stato di **dataRow** in **Deleted**
- Una **DataTable** può mantenere più versioni della stessa **DataRow**
- Le modifiche ad una **DataRow** vengono eseguite sulla versione **Current** della **DataRow**

```
<enumeration>
DataRowState
Detached = 1
Unchanged = 2
Added = 4
Deleted = 8
Modified = 16
```

```
<enumeration>
DataRowVersion
Original = 256
Current = 512
Proposed = 1024
Default = 1536
```

Laboratorio di Ingegneria del Software-LA

8.19

Modifiche ai dati

- Se durante la modifica di una riga si è riscontrato un errore, è possibile associare un messaggio di errore
 - all'intera riga

```
dataRow.RowError = messaggioDiErrore
```
 - ad una colonna specifica

```
dataRow.SetColumnError(colonna, messaggioDiErrore)
```
- È possibile verificare se una riga contiene degli errori
 - **dataRow.HasErrors**
- È possibile ottenere i messaggi di errore associati
 - all'intera riga

```
dataRow.RowError
```
 - ad una colonna specifica

```
dataRow.GetColumnError(colonna)
```
- È possibile cancellare tutti gli errori associati ad una riga
 - **dataRow.ClearErrors()**

Laboratorio di Ingegneria del Software-LA

8.20

Modifiche ai dati

- Le modifiche rimangono nella cache locale e verranno propagate alla sorgente dati (ad es. al database) mediante **DataAdapter**
- Il metodo **AcceptChanges** di **DataSet**, **DataTable** e **DataRow**
 - Esegue la **commit** di tutte le modifiche nella cache locale
 - Riporta allo stato **Unchanged** le righe inserite e modificate
 - Rimuove le eliminate.
- Il metodo **RejectChanges** di **DataSet**, **DataTable** e **DataRow**
 - Esegue un **rollback** delle modifiche
 - Le righe tornano allo stato originale

8.21

Selezionare e ordinare i dati

- Mediante il metodo **Select** è possibile
 - selezionare le righe di una **DataTable**,
 - eventualmente, ordinare le righe selezionate e
 - ottenere come risultato un'array di **DataRow**

```
DataRow[] Select(string filterExpression)
DataRow[] Select(string filterExpression, string sort)
```

libri.Select("id > 10", "titolo DESC")

8.22

System.Data.DataView

- Costruita su una **DataTable** per
 - sorting, filtering, searching, editing, and navigation
- Permette di selezionare le righe della **DataTable**, ma non permette di selezionare le colonne
- Utilizzata principalmente per poter mostrare i dati della stessa tabella in modo diverso
 - Data Binding e Ordinamenti
- Contiene una collezione di oggetti di tipo **DataRowView**
- Permette di inserire, modificare e cancellare righe della **DataTable** sulla quale è costruita

8.23

System.Data.DataView

DataView	
+ «event»	ListChanged : System.ComponentModel.ListChangedEventHandler
+ «property»	AllowDelete : System.Boolean
+ «property»	AllowEdit : System.Boolean
+ «property»	AllowNew : System.Boolean
+ «property»	Count : System.Int32
+ «property»	RowFilter : System.String
+ «property»	RowStateFilter : System.Data.DataRowViewRowState
+ «property»	Sort : System.String
+ «property»	Table : System.Data.DataTable
+ «property»	Item(System.Int32) : System.Data.DataRowView
+ DataView ([in] table : System.Data.DataTable)	
+ AddNew () : System.Data.DataRowView	
+ Delete ([in] index : System.Int32)	
+ Find ([in] key : System.Object) : System.Int32	
+ Find ([in] key : System.Object[]) : System.Int32	
+ FindRows ([in] key : System.Object) : System.Data.DataRowView[]	
+ FindRows ([in] key : System.Object[]) : System.Data.DataRowView[]	

8.24

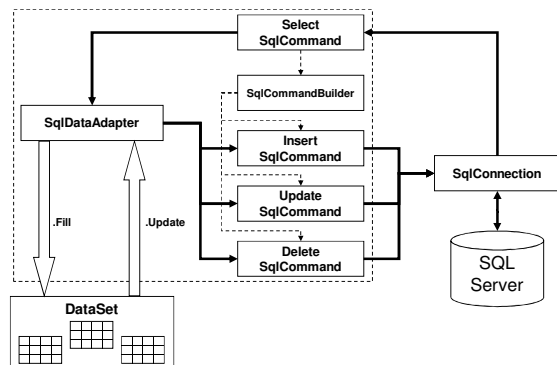
System.Data.DataRowView

DataRowView	
+ «property»	DataView : System.Data.DataView
+ «property»	Item(System.Int32) : System.Object
+ «property»	Item(System.String) : System.Object
+ «property»	Row : System.Data.DataRow
+ «property»	RowVersion : System.Data.DataRowVersion
+ «property»	IsNew : System.Boolean
+ «property»	IsEdit : System.Boolean
+	CancelEdit ()
+	EndEdit ()
+	BeginEdit ()
+	Delete ()

DataAdapter

- Fornisce il meccanismo di interazione tra
 - la parte "connessa" alla sorgente dei dati
 - la parte "disconnessa" del **DataSet**
- È costituito
 - da un **insieme di comandi** e
 - da una **connessione** alla sorgente dati
 che vengono utilizzati per
 - riempire il **DataSet** con i dati letti dalla sorgente
 - aggiornare la sorgente per tenere conto di operazioni di
 - inserimento
 - modifica
 - cancellazione
 effettuate sul **DataSet**
- Tipicamente si utilizza un **DataAdapter** per **DataTable**

Architettura del DataAdapter



Metodo Fill

- Accetta come parametri
 - un **DataSet** e il nome di una tabella, oppure
 - una **DataTable**
- Retrieves rows from the data source using the **SELECT** statement specified by an associated **SelectCommand** property
 - The connection object associated with the **SELECT** statement must be valid, but it does not need to be open
 - If the connection is closed before **Fill** is called, it is opened to retrieve data, then closed
 - If the connection is open before **Fill** is called, it remains open
- Then adds the rows to the specified destination **DataTable** object in the **DataSet**, creating the **DataTable** object if it does not already exist
 - When creating a **DataTable** object, the **Fill** operation normally creates only column name metadata. However, if the **MissingSchemaAction** property is set to **AddWithKey**, appropriate primary keys and constraints are also created

Metodo Update

- Accetta come parametri
 - un **DataSet** e il nome di una tabella, oppure
 - una **DataTable**
- Propaga al database le modifiche effettuate sulla **DataTable**
 - Invoca rispettivamente il comando **INSERT**, **UPDATE**, o **DELETE** per ogni riga inserita, modificata o cancellata dalla **DataTable**
- Per poter invocare il metodo **Update**, è necessario avere già creato le tre istanze di **Command** con **INSERT**, **UPDATE** e **DELETE**
 - automaticamente in design
 - mediante un **CommandBuilder**
 - manualmente da codice

Esempio

```
DataSet ds = new DataSet(dataSetName);
SqlConnection nwindConn = new SqlConnection(
    "Data Source=localhost; Integrated Security=SSPI;
    Initial Catalog=Northwind");
SqlDataAdapter custDA = new SqlDataAdapter(
    "SELECT * FROM Customers", nwindConn);
SqlCommandBuilder custCB = new SqlCommandBuilder(custDA);
custDA.MissingSchemaAction = MissingSchemaAction.AddWithKey;
// Crea la tabella, aggiunge i vincoli e la riempie
custDA.Fill(ds, "Customers");
// Modifico la tabella "Customers"
...
// Eseguo l'update - senza custCB, l'update fallirebbe
custDA.Update(ds, "Customers");
```

Esempio

```
// Phoenix - Metodo di configurazione di un sqlDataAdapter
// La configurazione avviene in modo (semi-)automatico
SqlCommandBuilder commandBuilder =
    new SqlCommandBuilder(sqlDataAdapter);
SqlCommand insertCommand = commandBuilder.GetInsertCommand();
if (insertCommand.CommandText != null)
    insertCommand.CommandText = InsertCommandText;
SqlCommand updateCommand = commandBuilder.GetUpdateCommand();
if (updateCommand.CommandText != null)
    updateCommand.CommandText = UpdateCommandText;
SqlCommand deleteCommand = commandBuilder.GetDeleteCommand();
if (deleteCommand.CommandText != null)
    deleteCommand.CommandText = DeleteCommandText;
commandBuilder.DataAdapter = null;
commandBuilder.Dispose();
sqlDataAdapter.InsertCommand = insertCommand;
sqlDataAdapter.UpdateCommand = updateCommand;
sqlDataAdapter.DeleteCommand = deleteCommand;
```

Come funziona Update

- Di ogni riga la **DataTable** tiene traccia
 - sia dei valori originali delle colonne
 - sia dei valori correnti delle colonne
- Ogni **DataRow** tiene traccia
 - del suo stato
- Il **DataAdapter** è in grado di
 - individuare le righe inserite, modificate o cancellate
 - inviare gli aggiornamenti pendenti, utilizzando il comando corrispondente

```
<enumeration>
DataRowVersion
Original = 256
Current = 512
Proposed = 1024
Default = 1536
```

```
<enumeration>
DataRowState
Detached = 1
Unchanged = 2
Added = 4
Deleted = 8
Modified = 16
```


SqlConnection

- Rappresenta una connessione fisica verso SQL Server
- Viene passato a oggetti **SqlCommand** o **SqlDataAdapter**
- Permette di creare **transazioni locali**
 - Il metodo **BeginTransaction** restituisce un oggetto **SqlTransaction** che rappresenta il contesto della transazione

SqlConnection	
+ «event»	InfoMessage : System.Data.SqlClient.SqlInfoMessageEventHandler
+ «event»	StateChange : System.Data.StateChangeEventHandler
+ «property»	ConnectionString : System.String
+ «property»	ConnectionTimeout : System.Int32
+ «property»	Database : System.String
+ «property»	Packetsize : System.Int32
+ «property»	ServerVersion : System.String
+ «property»	State : System.Data.ConnectionState
+ SqlConnection ()	
+ SqlConnection ([in] connectionString : System.String)	
+ Open ()	
+ Close ()	
+ BeginTransaction () : System.Data.SqlClient.SqlTransaction	
+ CreateCommand () : System.Data.SqlClient.SqlCommand	

SqlConnection

- Per utilizzare la modalità "connessa" occorre aprire e chiudere la connessione
 - Implicitamente con **SqlCommand** e **SqlDataReader**
 - Esplicitamente con i metodi **Open/Close** (o **Dispose**)
- Ogni oggetto **SqlConnection** mantiene riferimenti a risorse **unmanaged** che vanno rilasciate appena possibile pertanto, in caso di apertura esplicita:
 - non basarsi sul GC
 - chiudere esplicitamente la connessione appena possibile utilizzando il metodo **close** o il metodo **Dispose**
 - quando possibile, utilizzare la **keyword using**
- Utilizza **connection pooling**
 - per migliorare la scalabilità
 - abilitato di default, ma può essere configurato

Gestione degli errori

- Eventuali errori che avvengono durante l'esecuzione di operazioni sulla sorgente dati vengono intercettati e convertiti in eccezioni
- Ogni provider implementa le proprie eccezioni
- **SqlException**
 - Contiene almeno una istanza di tipo **SqlError**
 - **SqlError** restituisce le informazioni specifiche dell'errore

SqlException	
+ «property»	Errors : System.Data.SqlClient.SqlErrorCollection
+ «property»	Class : System.Byte
+ «property»	LineNumber : System.Int32
+ «property»	Message : System.String
+ «property»	Number : System.Int32
+ «property»	Procedure : System.String
+ «property»	Server : System.String
+ «property»	State : System.Byte
+ «property»	Source : System.String

SqlError	
+ «property»	Source : System.String
+ «property»	Number : System.Int32
+ «property»	State : System.Byte
+ «property»	Class : System.Byte
+ «property»	Server : System.String
+ «property»	Message : System.String
+ «property»	Procedure : System.String
+ «property»	LineNumber : System.Int32
+ ToString ()	: System.String

Gestione dei messaggi informativi

- Eventuali messaggi informativi generati dalla sorgente dati possono essere intercettati gestendo l'evento **InfoMessage**
- **SqlInfoMessageEventHandler**
- **SqlInfoMessageEventArgs**

SqlInfoMessageEventArgs	
+ «property»	Errors : System.Data.SqlClient.SqlErrorCollection
+ «property»	Message : System.String
+ «property»	Source : System.String
+ ToString ()	: System.String

Esempio

```
using (SqlConnection conn = new SqlConnection("..."))
{
    // Associo un handler all'evento InfoMessage
    conn.InfoMessage += new SqlInfoMessageEventHandler(myHandler);
    try
    {
        conn.Open();
        ...
    }
    catch (SqlException e)
    {
        for (int i = 0; i < e.Errors.Count; i++)
        {
            // Accedo alle informazioni sull'errore i-esimo
            ... e.Errors[i].ToString() ...
        }
    }
}
```

Laboratorio di Ingegneria del Software L-A

8.37

Esempio

```
public static void myHandler(object conn,
    SqlInfoMessageEventArgs e)
{
    for (int i = 0; i < e.Errors.Count; i++)
    {
        // Accedo al messaggio informativo i-esimo
        ... e.Errors[i].ToString() ...
    }
}
```

Laboratorio di Ingegneria del Software L-A

8.38

SqlCommand

- Rappresenta un comando da eseguire su SQL Server
- Opera attraverso una connessione ad esso associata
 - in fase di creazione dell'oggetto o successivamente
- Permette di eseguire operazioni di tipo diverso
 - query ad-hoc
 - stored procedure

```
SqlCommand
+ <property> Connection : System.Data.SqlClient.SqlConnection
+ <property> Transaction : System.Data.SqlClient.SqlTransaction
+ <property> CommandText : System.String
+ <property> CommandType : System.Data.CommandType
+ <property> CommandTimeout : System.Int32
+ <property> Parameters : System.Data.SqlClient.SqlParameterCollection
+ SqlCommand ( )
+ SqlCommand ( [in] cmdText : System.String )
+ SqlCommand ( [in] cmdText : System.String, [in] connection : System.Data.SqlClient.SqlConnection )
+ SqlCommand ( [in] cmdText : System.String, [in] connection : System.Data.SqlClient.SqlConnection, [in] transaction : System.Data.SqlClient.SqlTransaction )
+ ExecuteScalar ( ) : System.Object
+ ExecuteNonQuery ( ) : System.Int32
+ Cancel ( )
+ Prepare ( )
+ ResetCommandTimeout ( )
+ CreateParameter ( ) : System.Data.SqlClient.SqlParameter
+ ExecuteXmlReader ( ) : System.Xml.XmlReader
+ ExecuteReader ( ) : System.Data.SqlClient.SqlDataReader
```

Laboratorio di Ingegneria del Software L-A

8.39

SqlCommand

- Sfrutta tutte le caratteristiche espone da SQL Server
 - Es. esecuzione di query FOR XML
- Implementa l'interfaccia **IDbCommand**
 - **ExecuteReader**
 - Da utilizzare quando è previsto un result set come ritorno
 - **ExecuteScalar**
 - Da utilizzare per aggregazioni o risultati di calcoli
 - Restituisce la prima colonna della prima riga
 - **ExecuteNonQuery**
 - Ottimizzato per query che non restituiscono result set ma solo parametri di ritorno o numero di record modificati
 - **ExecuteXmlReader**
 - Da utilizzare insieme alla clausola FOR XML

Laboratorio di Ingegneria del Software L-A

8.40

Esempio

```
int rowsAffected;
using (SqlConnection sqlConn = new SqlConnection(myConnString))
{
    try
    {
        // Creo il comando
        SqlCommand sqlCmd = new SqlCommand();
        // specifico il tipo di comando
        sqlCmd.CommandType = CommandType.Text;
        sqlCmd.CommandText =
            "Insert Customers (Alias, CustomerName) Values ('myAlias', 'myName')";
        sqlCmd.Connection = sqlConn;
        // apro la connessione
        sqlConn.Open();
        // eseguo il comando, vengono restituite le righe inserite
        rowsAffected = sqlCmd.ExecuteNonQuery();
    }
    catch (SqlException e)
    {
        // gestisco l'eccezione ...
    }
}
```

Laboratorio di Ingegneria del
Software-LA

8.41

SqlCommand e query parametriche

- Utilizzare la collezione **Parameters** per passare i parametri di input e di output
 - nei comandi
 - nelle chiamate a stored procedure
- I parametri sono nominali

Laboratorio di Ingegneria del
Software-LA

8.42

Esempio

```
// Create a new command object
SqlCommand sqlCmd = new SqlCommand();
// Specify the stored procedure and connection
sqlCmd.CommandType = CommandType.StoredProcedure;
sqlCmd.CommandText = "InsertCustomer";
sqlCmd.Connection = sqlConn;
// Define and add a parameter to the parameters collection
SqlParameter param = sqlCmd.Parameters.Add(
    new SqlParameter("@p", SqlDbType.NVarChar, 100);
param.Direction = ParameterDirection.Input;
// Set the parameter value
param.Value = "xyz";
// Add remaining parameters
...
// Open the connection
sqlConn.Open();
// Execute the command
rowsAffected = sqlCmd.ExecuteNonQuery();
```

Laboratorio di Ingegneria del
Software-LA

8.43

SqlDataReader

- Fornisce funzionalità di accesso *read-only* e *forward-only* su uno *stream* di record restituiti dall'esecuzione di un comando sul database
- Viene creato al momento dell'esecuzione del metodo **ExecuteReader** di un oggetto **SqlCommand**
- Il metodo **Read**
 - legge il record successivo nello *stream*
 - restituisce **false** alla fine del *result set*
- Esiste un solo record alla volta in memoria
 - Aumenta performance e scalabilità delle applicazioni se usato correttamente

Laboratorio di Ingegneria del
Software-LA

8.44

SqlDataReader

- Accesso ai valori nelle singole colonne mediante
 - indice o nome del campo
 - metodi `Get` specifici per i tipi di dato contenuti
 - `GetDateTime`
 - `GetDouble`
 - `GetInt32`
 - ecc.

```
SqlDataReader myReader = myCommand.ExecuteReader();
while (myReader.Read())
{
    ... myReader.GetInt32(0) ... // valore del 1° campo
    ... myReader.GetString(1) ... // valore del 2° campo
}
myReader.Close();
```