**ADO.NET**

http://www.antoniosammartino.it/ebook/Default.aspx?Id=212

1 - Introduzione ADO.NET

2 - Sql Connection Class

3 - Sql Command Class

4 - SqlParameter Class

5 - SqlDataReader Class

6 - Uso della SqlConnection

7 - Classe Base

8 - Classe Derivata

9 - Esempio CRUD

10 - Paginazione con ado.net

**1 - Introduzione ad ADO.NET**

ADO.NET è una libreria di classi il cui fine è di consentire una interazione con Database, file testo, file XML, ecc.

**Data Provider**

ADO.NET consente ad un'applicazione web di scambiare informazioni con diversi tipi di fonti dati o Database. Tuttavia, non vi è un insieme unico di classi, per riferirsi ai diversi database, ma per ogni tipo di database è stato implementato un diverso **Data Provider**, cioè librerie di classi, che definiscono un modo comune per interagire con specifiche sorgenti dati o protocolli. I pricipali provider sono i seguenti:

|  |  |  |
| --- | --- | --- |
| **Provider Name** | **API prefix** | **Data Source Description** |
| ODBC Data Provider | Odbc | Sorgenti dati con interfaccia ODBC |
| OleDb Data Provider | OleDb | Sorgenti dati che espone interfaccie OleDb (Access o Excel) |
| Oracle Data Provider | Oracle | Per database Oracle |
| **SQL Data Provider** | **Sql** | **Per interfaccia con Microsoft SQL Server.** |
|  |  |  |

**Namespace**

Per utilizzare gli oggetti ADO.NET associati al database Sql-Server occorre dichiarare i seguenti namespace:

using System;

using System.Data;

using System.Data.SqlClient;

**ADO.NET Objects**   
ADO.NET comprende diversi oggetti che possono essere utilizzati per interfacciarsi con un  database SQL Server. I più importanti sono:

* SqlConnection Object
* SqlCommand Object
* SqlParameter Object
* SqlDataReader Object
* SqlTransaction Object
* DataSet Object Sql
* DataAdapter Object

**2 - Sql Connection Class**

Per interrogare un base dati è necessario disporre di una connessione al database. Il fine è di indicare Server, Nome del database, nome utente, password ed altri parametri necessari per il collegamento al database. L’oggetto connessione viene utilizzato dai comandi per specificare su quale database occorre eseguire i comandi.

Una connection reference list è disponibile sul sito:  <http://www.connectionstrings.com/>

**Le Properties** della Sql Connection classa sono:  
ConnectionString    Connection string al database  
ConnectionTimeout Timeout in secondi della connessione   
Database                Nome del database   
DataSource             Nome del Server   
PacketSize             Dimensione dei packets usati per comunicare con il Server  
ServerVersion          Version of SQL   
State                      Stato della connessione   
WorkstationId          Identificatore NetBIOS della machine che ospita il Web form

**I Metodi** della Sql Connection classa sono:  
BeginTransaction()  Pone la connessione in una transazione  
ChangeDatabase()  Consente di associare ad una connessione un differente database   
Close()                   Chiude la connessione   
CreateCommand()   Ritorna un nuovo command object   
Open()                    Apre una connessione al database

**Connection String**   
Il codice che segue è un esempio di connection string per il database SqlEspress 2005

SqlConnection conn = new SqlConnection();   
conn.ConnectionString = "Data Source=.\SQLExpress;Integrated security=True;   
         User Instance=True;AttachDBFilename=|DataDirectory|Idea05.mdf”;

Il significato dei diversi parametri è il seguente:

* Data Source è il machine name del computer su cui è running sql server
* Integrated security indica che occorre usare le attuali credenziali degli utenti di windows (username e password) per accedere a sql server. I possibili valori sono true, false e sspi (indica connessione sicura).
* User Instance, durante la connessione crea una nuova local SQL Server Express instance. Funziona solo quando si usa l’autenticazione di Windows. Lo scopo è di poter creare una full rights SQL Server instance per un utente con limitati diritti amministrativi sul computer.
* AttachDBFilename, Path completo del nome del principale file del database

**Connection Timeout**E’ una property read-only che contiene il timeout in secondi della connessione. Il valore di default è 30 sec. Per modificare questo parametro occorre indicare il valore nella connection string.

SqlConnection conn = new SqlConnection( ConnString );

msg.Text = conn.ConnectionTimeout;

**Database**

E’ una property read-only che contiene il nome del database. Per cambiare il valore occorre usare il metodo ChangeDatabase() oppure modificare la connection string.

SqlConnection conn = new SqlConnection( ConnString );

msg.Text = conn.Database;

**State**

La property State consente di ottenere lo stato della connessione. Sono possibili i seguenti valori: Broken, Closed, Connecting, Executing, Fetching, and Open. Questi valori sono definite nel namespace System.Data.ConnectionState

SqlConnection conn = new SqlConnection( ConnString );

msg.Text = conn.State.ToString();

**Transaction**

Consente di eseguire diverse istruzioni SQL in modo consistente. Ad esempio, a volte occorre aggiornare diverse tabelle in un database (con Insert, update, ecc.) in modo consistente, nel senso che se si verifica un errore occorre annullare tutte le operazioni.

SqlConnection conn = new SqlConnection( ConnString );

Conn.Open();

SqlTransaction tx = conn.BeginTransaction();

*// Esegue le diverse operazioni …*

tx.Commit();

conn.Close();

**ChangeDatabase**

Consente di cambiare database senza dover creare una nuova connessione.

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

conn.ChangeDatabase("dbNew");

conn.Close()

**CreateCommand**

Ritorna una nuova instanza di un oggetto SqlCommand

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = conn.CreateCommand();

cmd.CommandText = "DELETE FROM Prodotti";

**3 - Sql Command Class**

Comunicare con un database significa indicare le operazioni che si desiderano fare, mediante la classe SqlCommand. Questa classe viene usata per eseguire operazioni su un database o per ottenere informazioni. In altri termini, si utilizza un Object Command per inviare istruzioni SQL alla base di dati. Un oggetto comando utilizza l'oggetto connessione, per conoscere con quale database comunicare.   
È possibile utilizzare SqlCommand, per eseguire un comando direttamente oppure mediante un riferimento ad un oggetto SqlDataAdapter, in cui sono memorizzati una serie di comandi da eseguire su un insieme di dati.

Properties   
CommandText       Contiene il testo di un commando sql   
CommandTimeout  Timeout in secondi per la query   
CommandType       Specifica il tipo di commando da eseguire   
Connection            Specifica la connessione al database   
Parameters            Specifica i parametri della query   
SQL Transaction    Specifica un transaction object

Methods   
Cancel()                     Annulla la query   
CreateParameter()      Ritorna un nuovo SQL parameter   
ExecuteNonQuery()    Esegue il commando indicato in CommandText, ma non ritorna dati   
ExecuteReader()        Esegue il commando e ritorna i dati in un oggetto DataReader  
ExecuteScalar()         Esegue il commando  e ritorna un singolo valore  
ExecuteXmlReader()   Esegue il commando  e ritorna i dati in un oggetto XMLDataReader  
ResetCommandTimeout() Reset la proprietà CommandTimeout della query

**CommandText**

La property CommandText è una string che specifica la query da eseguire

SqlCommand cmd = new SqlCommand();

cmd.CommandText = "SELECT \* FROM Clienti";

SqlCommand.CommandTimeout

**CommandTimeout**

La property CommandTimeout specifica il timeout della query. Il valore di default è di 30 sec.

SqlCommand cmd = new SqlCommand();

cmd.CommandTimeout = 50;

**CommandType**

La property CommandType specifica il tipo di query da eseguire. Il CommandType è definito nel namespace System.Data. Può essere StoreProcedure, TableDirect o Text.

SqlCommand cmd = new SqlCommand();

cmd.CommandType = CommandType.StoreProcedure;

**Connection**

La property Connection contiene un valido SqlConnection object su cui bisogna eseguire la query.

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand();

cmd.Connection = conn;

**Parameters**

La property Parameters contiene i nomi di un gruppo di parametri SQL che devono essere usati in una istruzione SQL oppure quando si invoca una Stored Procedure che richiede parametri di Input/Output.

*Nell'esempio che segue dopo aver aperto una connessione al database, viene creato un comando per eseguire una stored procedure. Successivamente vendono creati i parametri di input, output ed infine viene eseguito il comando:*

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("AddNewClient", conn);

cmd.CommandType = CommandType.StoredProcedure;

// ---------------------------------------------------------Configura I parametric di input

SqlParameter param = new SqlParameter();

param = cmd.Parameters.Add(new SqlParameter("@Name", SqlDbType.NVarChar, 20));

param.Direction = Parameter.Direction.Input;

param.Value = sName.Text;

// ---------------------------------------------------------Configura I parametric di output

param = cmd.Parameters.Add(new SqlParameter("@retval", SqlDbType.Int, 4));

param.Direction = ParameterDirection.Output;

cmd.ExecuteNonQuery();

**Transaction**

La property Transaction contiene l’eventuale oggetto transaction per l’attuale serie di comandi

* Il metodo BeginTransaction dell'oggetto Conn segna  l'inizio della transazione
* Vengono Eseguiti i comandi
* Infine viene attivato  il metodo Commit per completare la transazione o il metodo Rollback per annullare la transazione

 SqlConnection conn = new SqlConnection(ConnString);

conn.Open();

SqlTransaction trans = conn.BeginTransaction(IsolationLevel.Serializable, "RemoveClients");

cmd = new SqlCommand("", conn);

cmd.Transaction = trans;

……

trans.Commit();

**Cancel()**

Il metodo Cancel() rimuove una query attiva.

SqlConnection conn = new SqlConnection(ConnString);

conn.Open();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Clienti", conn);

cmd.ExecuteNonQuery();

cmd.Cancel();

**CreateParameter()**

Il metodo CreateParameter() ritorna una nuova istanza di un SQL parameter.

SqlParameter param = cmd.CreateParameter();

param.ParameterName = "@CustomerID";

param.DbType = SqlDbType.NVarChar;

param.Size = 5;

param.Value = "123";

param.Direction = ParameterDirection.Input;

cmd.Parameters.Add(param);

**ResetCommandTimeout()**

Ripristina il valore di default (30 sec.) della proprietà CommandTimeout.

cmd.ResetCommandTimeout();

**ExecuteNonQuery()**

Il metodo ExecuteNonQuery() esegue il commando indicato ella proprietà Command Text. E’ un metodo ottimizzato per le query che non ritornano informazioni.

È possibile utilizzare il metodo **ExecuteNonQuery** per ottenere informazioni sulla struttura di un database, per creare oggetti del database (per esempio, tabelle), modificare i dati contenuti in un database mediante le istruzioni UPDATE, INSERT o DELETE. Il valore restituito da queste istruzioni indica il numero di righe interessate dal comando, mentre per tutte le altre istruzioni ritorna il valore -1.

SqlConnection conn = new SqlConnection(ConnString);

conn.Open();

SqlCommand cmd = new SqlCommand("DELETE FROM Autori

         WHERE LastName='Samanton'", conn);

cmd.ExecuteNonQuery();

**ExecuteReader()**

Il metodo ExecuteReader() esegue il commando indicato nella property command text. Ritorna un oggetto SqlDataReader che è il risultato della query.

SqlConnection conn = new SqlConnection(ConnString);

conn.Open();

SqlCommand cmd = new SqlCommand("SELECT \* FROM Customers", conn);

SqlDataReader reader = cmd.ExecuteReader();

**ExecuteScalar()**

ExecuteScalar è un efficiente metodo dell'oggetto Command che può essere usato quando una query deve ritornare un solo valore.

SqlConnection conn = new SqlConnection(ConnString);

conn.Open();

SqlCommand cmd = new SqlCommand("SELECT count(\*) FROM Client", conn);

int32 NumOfClient = (int32)cmd.ExecuteScalar();

msg.Text = "Vi sono "+ NumOfClient.ToString()+" client nel database.";

**4 - SqlParameter Class**

Questa classe rappresenta un parametro di un oggetto SqlCommand. Viene usato per passare parametri alle Stored Procedure o a comandi.

DbType      Contiene il data type del Sql Parameter   
Direction    Definisce il parametro di input o output   
IsNullable   Definisce quando un parametro può essere null (true o false)   
Precision    Contiene la precision del parametro   
Scale         Contiene la scala del parametro   
Size           Contiene il size del parametro   
Value         Contiene il valore del parametro   
ParameterName Contiene il nome del parametro

 Per settare le property usare le seguenti istruzioni

SqlParameter param = new SqlParameter();

param.DbType = SqlDbType.NVarChar;

param.Direction = ParameterDirection.Input;

param.IsNullable = true;

Per usare l’oggetto SqlParameter occorre:

1.     Costruire la stringa di comando con i parametri

2.     Dichiarare un SqlParameter object e assegnarli i valori

3.     Assegnare il SqlParameter object alla property del SqlCommando

Il primo step è costruire una command string che contiene i parametri formali, preceduti dal simbolo @, che saranno successivamente sostituiti con i valori effettivi quando sarà eseguito il SqlCommand.

SqlCommand cmd = new SqlCommand(

"SELECT \* FROM Client WHERE NomeAzienda = @Nome", conn);

Il second step consiste nel dichiarare tutti parametric indicate nella stringa di comando

SqlParameter param = new SqlParameter();

param.ParameterName = "@Nome";

param.Value = "IdeaSoftware";

Il terzo step consiste nell’aggiungere il parametro all’oggetto command

cmd.Parameters.Add(param);

Un esempio di utilizzo della classe SqlParameter è il seguente:

        public void ReadWithParam( int inputId )

{

SqlDataReader dr = null;

string ConnString = ConfigurationManager.AppSettings["dbIDEA"];

SqlConnection conn = newSqlConnection(ConnString);

conn.Open();

//-------------------------------------------------------------------

SqlCommand cmd = newSqlCommand("SELECT \* FROM tPost where Id = @Id", conn);

SqlParameter param = newSqlParameter(); // 2. Definizione parametri

param.ParameterName = "@Id";

param.Value = inputId;

cmd.Parameters.Add(param); // 3. Aggiunge I parametri al Commando

dr = cmd.ExecuteReader();

string s = "";

while(dr.Read()) {

s += Convert.ToString(dr["Titolo"]) + "<br />";

}

Titolo.Text = s;

if (dr != null)

dr.Close();

if (conn != null)

conn.Close();

}

**5 - SqlDataReader Class**

Fornisce un metodo per la lettura di un flusso di righe in modo forward-only da un database SQL. Ciò significa che è possibile estrarre i dati dal flusso, solo in modo sequenziale, mentre se bisogna manipolare i dati nelle diverse righe occorre usare un DataSet.  
Per creare un oggetto SqlDataReader, invece di utilizzare un costruttore, occorre attivare il metodo ExecuteReader dell'oggetto SqlCommand,.   
Mentre l'oggetto SqlDataReader è in uso, l'oggetto SqlConnection associato è occupato a rendere disponibile l'oggetto SqlDataReader, non è possibile eseguire altre operazioni  sull'oggetto SqlConnection fino a quando non viene chiamato il metodo Close dell'oggetto SqlDataReader.

**Properties**FieldCount Contiene il numero di fields ottenuti dalla query   
IsClosed    vale True se il SqlDataReader object è chiuso  
Item          Contiene una collection di valori accessibili per nome o per numero ordinale   
RecordsAffected  Ritorna il numero di records influenzati da una executed query   
  
**Methods**Close()           Chiude il SqlDataReader object   
GetBoolean()   Ritorna un valore di tipo Boolean (esiste un metodo per ogni tipo di dati)  
IsDBNull()       Ritorna True se il campo contiene Null  
Read()            Lettura del successive valore dal result set into memory

     string queryString = "SELECT ID, Nome FROM dbo.Client;" ;

using (SqlConnection Conn = new SqlConnection( connString )) {

SqlCommand cmd = new SqlCommand(queryString, conn);

Conn.Open();

SqlDataReader dr = cmd.**ExecuteReader**();

while (dr.Read()) {

int val= Convert.ToInt32(dr[0]);

string s= Convert.ToString(dr ["Nome"]);

}

dr.Close();

Conn.Close();

}

**FieldCount**

Ritorna il numero di fields nel current record

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("Select id, lastname, firstname from Client", conn);

SqlDataReader dr = cmd.ExecuteReader();

FieldCount = dr.FieldCount;

**IsClosed**

vale True se il SqlDataReader object è chiuso

sqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("Select id, lastname, firstname from Client", conn);

SqlDataReader dr = cmd.ExecuteReader();

If (reader.IsClosed != true) {

// Esegue le operazioni con il datareader

}

**Item**

La property Item property ritorna il valore di una colonna nel suo format native

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("Select id, lastname, firstname from Client", conn);

SqlDataReader dr = cmd.ExecuteReader();

string msg = "";

while(reader.Read()){

msg = msg + dr["employee\_id"] + " " + dr["lastname"] +

", " + dr["firstname"] + "<br>";

}

dr.Close();

**RecordsAffected**

The RecordsAffected property contains the number of records affected by the query.

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("Select id, lastname, firstname from Client", conn);

SqlDataReader dr = cmd.ExecuteReader();

int RecordsAffected = dr.RecordsAffected;

**Close()**

Il metodo Close() consente di chiudere il SqlDataReader object.

SqlDataReader dr = cmd.ExecuteReader();

dr.Close();

**GetBoolean(Int32 i)**

Il metodo GetBoolean() ritorna il valore di una specificata colonna di Booleano.

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("Select fieldname, FROM Table", conn);

SqlDataReader dr = cmd.ExecuteReader();

dr.Read();

bool value = dr.GetBoolean(0);

dr.Close();

conn.Close();

**IsDBNull()**

Il metodo IsDBNull() ritorna True se la colonna specificata contiene null.

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("Select fieldname FROM Table", conn);

SqlDataReader reader = cmd.ExecuteReader();

while(reader.Read()){

if(reader.IsDBNull(0) != true)

value = reader.GetValue(0);

}

reader.Close();

conn.Close();

**Read()**

Il metodo Read() avanza il data reader al successive record. Ritorna True se vi sonono recods, diversamente ritorna False

SqlConnection conn = new SqlConnection( ConnString );

conn.Open();

SqlCommand cmd = new SqlCommand("Select fieldname from aTable", conn);

SqlDataReader dr = cmd.ExecuteReader();

string value;

while(dr.Read()) {

value = dr.GetValue(0);

}

dr.Close();

conn.Close();

**6 - Uso della SqlConnection**

**Definizione in web.config**

Un possibile modo per definire i parametri necessari per aprire una connessione è quello di definire, nel web.Config  una **Key** nella sezione  **appSetting** nel seguente modo**:**

< appSettings >

< add key = "dbIdea"value="Data Source=.\SQLExpress;Integrated

Security=True;User Instance=True;

AttachDBFilename=|DataDirectory|Idea05.mdf " />

  </ appSettings >

**Uso della SqlConnection classa**

La prima cosa che occorre fare quando si devono effettuare operazioni su un database è di creare una connessione. La connessione consente di identificare il database su cui si vuole operare e gestisce tutta la logica di basso livello per uno specifico protocollo. La connessione deve essere chiusa dopo aver effettuato le operazioni sul database.

La sequenza delle operazioni che occorre fare per operare su un database è la seguente:

1.     Instanziare SqlConnection

2.     Aprire la connessione

3.     Fornire la connessione agli altri oggetti ADO.NET

4.     Eseguire le operazioni sul database con gli altri oggetti ADO.NET

5.     Chiudere la connessione

Un  completo esempio è il seguente:

using System;  
using System.Data;  
using System.Data.SqlClient;

//--------------------------------------- 1. Instanziare la connection  
 string ConnString = ConfigurationManager.AppSettings["dbIDEA"];

SqlConnection conn = new SqlConnection( ConnString );

        SqlDataReader dr = null;

        try {  
              //----------------------------------- 2. Open della connection  
             conn.Open();

             //----------------------------------- 3. Passa la connection a un command object  
             SqlCommand cmd = new SqlCommand("select \* from Client", conn);

             //----------------------------------- 4. Use the connection

             dr = cmd.ExecuteReader();

               string s = "";

             while (dr.Read()) {  
               s+= Convert.ToString(dr[0]) + "-";  
             }  
         }  
         finally {  
         //--------------------------- 5a. Chiusura del datareader  
               if (dr != null)

                 dr.Close();

                 //---------------------------- 5b. Chiusura della connection  
                if (conn != null)

                conn.Close();

          }

Un oggetto SqlCommand consente di specificare quale tipo di operazione si desidera eseguire sul database. Ad esempio, è possibile eseguire comandi per: Selezionare, Inserire, Modificare e Cancellare righe di dati in una tabella del database.   
L'oggetto SqlCommand può anche essere utilizzato per la gestione dei dati nella modalità disconnessa, mediante l’impiego del **SqlDataAdapter**.

**Creare un SqlCommand object**

Per creare una instanza di un SqlCommand si usa la dichiarazione di instanza **new**:

string QueryString= "SELECT Title FROM tTableName",

SqlCommand cmd = new SqlCommand(QueryString, conn);

SqlCommand richiede un parametro stringa che contiene il comando che si desidera eseguire e un **riferimento** a un **oggetto**SqlConnection. SqlCommand ha diversi overloads.

**Querying Data**

Per leggere un set di record da un database si utilizza un comando SQL Select. Per eseguire questa operazione con un oggetto SqlCommand, si può utilizzare il metodo ExecuteReader, che restituisce un oggetto SqlDataReader. La sequenza delle operazioni è la seguente:

1. Instanzia un nuovo commando con una query string e una connection

string QueryString= "SELECT Title FROM tTableName",

SqlCommandcmd = newSqlCommand(QueryString, conn);

2. Per ottenere il risultato della query si esegue il metodo ExecuteReader

SqlDataReader dr = cmd.ExecuteReader();

**Inserting Data**  
Per inserire dati in un database si utilizza il metodo ExecuteNonQuery dell’object SqlCommand.

 string cmdString =   
                   @"INSERT INTO tPost (Title, Post) VALUES ('Titolo Post', 'Contenuto Post')";

 SqlCommandcmd = newSqlCommand(cmdString, conn); // Instanza del commando

 cmd.ExecuteNonQuery(); // Invio del commando

Con l’insert command non occorre inserire anche la chiave primaria Id in quando è aggiunta in automatic dal DBMS. Per eseguire questo comando si attiva il metodo ExecuteNonQuery sull’instanza cmd del SqlCommand.

**Updating Data**Il metodo ExecuteNonQuery viene anche usato per aggiornare i dati nel database. Sono disponibili diversi tipi di costruttori di SqlCommand. In questo esempio si evidenzia come sia possibile cambiare la connessione su un qualsiasi oggetto comando.

 string updateString = @"UPDATE tPost SET Title = 'NuovoTitolo' WHERE Id = 1";

 SqlCommand cmd = new SqlCommand(updateString);  // 1. Instanza di un nuovo commando

 cmd.Connection = conn;   // 2. Set della Connection property

 cmd.ExecuteNonQuery(); // 3. Esecuzione del comando

**Deleting Data**Il metodo ExecuteNonQuery viene anche usato per cancellare record. In questo esempio si può notare che è possibile cambiare sia il CommandText che la Connection.

string deleteString = @"DELETE FROM tPost WHERE Id = '1"; // Command string

SqlCommand cmd = new SqlCommand(); // 1. Instanza di un nuovo commando

cmd.CommandText = deleteString;   // 2. Set della Property CommandText

cmd.Connection = conn;   // 3. Set the Connection property

cmd.ExecuteNonQuery();  // 4. Esecuzione del comando

**Lettura di un singolo valore**Alcune volte si ha la necessità di leggere un singolo valore dal database, ad esempio un conteggio, una somma, una media, ecc. Utilizzare il metodo ExecuteReader per calcolare un valore, spesso non è efficiente. Una scelta migliore è di effettuare l’elaborazione nel database e ritornare un singolo valore mediante il metodo ExecuteScalar.

 SqlCommand cmd = new SqlCommand("SELECT COUNT(\*) FROM tPost", conn);

 int count = (int)cmd.ExecuteScalar();

La query nel costruttore SqlCommand ritorna il numero di tutti i record dalla tabella tPost. Siccome il valore di ritorno di ExecuteScalar è di tipo object, occorre effettuare un cast per convertire il valore in int.

**7 - Implementazione di una Classe Base**

In C# tutto il codice eseguibile deve essere definito in una classe, in quanto non è possibile definire funzioni o variabili globali, nel senso tradizionale del termine.   
Quando definiamo una classe è probabile che successivamente verrà utilizzata per creare oggetti o per derivare altre classi, mediante il meccanismo dell’ereditarietà. In quest’ultimo caso, la classe è detta Classe Base, in quanto viene utilizzata come fondamento per una nuova definizione. In altri termini, tutte le funzionalità della classe base vengono trasferite nella nuova classe, che è detta Classe Derivata. Ciò significa che la classe base deve comprendere solo funzionalità generiche che saranno successivamente specializzate, attraverso il processo di ereditarietà. Se funzionalmente la classe base non deve essere utilizzata per istanziare oggetti (perché incompleta), ma solo per derivare altri classi, occorre dichiararla con l’attributo abstract. Ciò consente al compilatore di generare un errore se la classe viene utilizzata per istanziare oggetti.

Questa premessa consente di dedurre che è opportuno, quando si realizza una qualsiasi applicazione, isolare le diverse funzionalità in almeno tre logical tiers:

* Interface Layer, in cui vi è il codice necessario per visualizzare le informazioni
* Business Logic, in cui vi è la logica associata all’applicazione
* Data Access Layer, in cui vi è il codice per accedere al database

Il primo passo è quindi quello di costruire la classe astratta, in cui viene incapsulato il codice di base che sarà condiviso da tutte le applicazioni e da cui successivamente saranno derivate le diverse classi associate al Data Access Layer.

E’ preferibile NON usare istruzioni SQL inline, ma incapsulare le istruzioni Transact-Sql nelle Stored Procedure. Ciò offre i seguenti vantaggi:

* aumento delle perfmance
* modifica del codice Sql di accesso ai dati senza dover modificare il codice nelle classi

**La Classe sqlDA**   
Questa classe comprende due property, il costruttore con un parametro che rappresenta la stringa di connessione, necessaria per creare una istanza SqlConnection. Viene creata la connessione, ma non viene aperta. Seguono i seguenti metodi:

* IsConnectionOpen(), usata per verificare se la connessione è aperta
* CloseConnection() , usata per chiudere una connessione
* ActiveStoredProc(), sono una serie di metodi, mediante i quali vengono eseguite le Stored Procedure.

public abstract classa sqlDA  
{  
       protected string ConnString { get ; set ; }         // Stringa di Connessione   
       protected SqlConnection Conn { get ; set ; }     // Connessione Sql

       /\*  **1**. Crea l'istanza di un ogetto sqlDA  
       ----------------------------------------------------------------------------------------------------------------\*/   
       public sqlDA( string connStringName)  
       {  
              ConnString = ConfigurationManager .AppSettings[connStringName];  
              Conn= new SqlConnection (ConnString);  
        }

       /\*  **2**. Verifica se la connessione è Open  
       ----------------------------------------------------------------------------------------------------------------\*/  
       public bool IsConnectionOpen()  
       {  
              bool bOpen = false ;  
              if (Conn!= null ) {  
                    if (Conn.State == ConnectionState .Open)  
                          bOpen = true ;  
               }  
              return bOpen;  
        }

       /\*  **3**. Chiude la connessione al database, se Open  
       ----------------------------------------------------------------------------------------------------------------\*/  
       public void CloseConnection()   
         {  
                if (Conn!= null ) {   
                       if (Conn.State == ConnectionState .Open)   
                              Conn.Close();  
                 }  
          }

       /\*  **4**. Attiva una Stored Proc. Per eseguire query Insert, Update, Delete. Ritorna   
               un int per indicare successo (oppure un codice di errore).  
       ----------------------------------------------------------------------------------------------------------------\*/   
       protectedintActiveStoredProc(stringspName,   
                              IDataParameter[] dataParam, outintnumRowsAffected)  
       {  
                  Conn.Open();  
                  SqlCommandcmd = newSqlCommand(spName, Conn);   
                  cmd.CommandType = CommandType.StoredProcedure;  
                  foreach(SqlParameterparameter indataParam) {  
                        cmd.Parameters.Add(parameter);  
                  }  
                  cmd.Parameters.Add(newSqlParameter("ReturnValue", SqlDbType.Int, 4,  
                                         ParameterDirection.ReturnValue,false, 0, 0, string.Empty,  
                                         DataRowVersion.Default, null));  
                  numRowsAffected = cmd.ExecuteNonQuery();   
                  intresult = (int)cmd.Parameters["ReturnValue"].Value;  
                  Conn.Close();  
                  returnresult;  
        }

        /\*  5. Attiva una Stored Proc. Ritorna l'istanza di object SqlDataReader che   
               contiene  il risultato della SP.  
        ----------------------------------------------------------------------------------------------------------------\*/   
        protected SqlDataReader ActiveStoredProc( string spName,   
                                    IDataParameter [] dataParam)  
          {  
                   Conn.Open();   
                   SqlCommand cmd = new SqlCommand (spName, Conn);   
                   cmd.CommandType = CommandType .StoredProcedure;   
                   foreach ( SqlParameter parameter in dataParam) {   
                           cmd.Parameters.Add(parameter);   
                  }  
                  cmd.CommandType = CommandType .StoredProcedure;   
                   SqlDataReader sqlDR = cmd.ExecuteReader();   
                   return sqlDR;   
           }   
  
          /\*    6. Crea un DataSet e memorizza il risultato della query/proc in tableName  
          ----------------------------------------------------------------------------------------------------------------\*/  
          protectedDataSetActiveStoredProc(stringspName,   
                                     IDataParameter[] dataParam, stringtableName )  
          {  
                        DataSetds = newDataSet();  
                        Conn.Open();  
                        SqlCommandcmd = newSqlCommand(spName, Conn);  
                        cmd.CommandType = CommandType.StoredProcedure;   
                        foreach(SqlParameterparameter indataParam) {  
                                 cmd.Parameters.Add(parameter);  
                         }  
                        SqlDataAdapterda = newSqlDataAdapter();  
                        da.SelectCommand = cmd;  
                        da.Fill( ds, tableName );  
                        Conn.Close();  
                        returnds;   
           }

          /\*    7.  Memorizza in un DataSet il risultato della query/proc in tableName  
          ----------------------------------------------------------------------------------------------------------------\*/   
          protected void ActiveStoredProc( string spName, IDataParameter [] dataParam,   
                                     DataSet dataSet, string tableName )  
          {  
                  Conn.Open();   
                  SqlCommand cmd = new SqlCommand (spName, Conn);   
                  cmd.CommandType = CommandType .StoredProcedure;   
                  foreach ( SqlParameter parameter in dataParam)  {  
                        cmd.Parameters.Add(parameter);   
                  }  
                  cmd.Parameters.Add( new SqlParameter ( "ReturnValue" ,   
                                SqlDbType .Int, 4, ParameterDirection .ReturnValue,   
                                false , 0, 0, string .Empty, DataRowVersion .Default, null ));  
                  SqlDataAdapter da = new SqlDataAdapter ();   
                        da.SelectCommand = cmd;  
                        da.Fill( dataSet, tableName );   
                        Conn.Close();   
           }  
}

**8 - Classe Derivata**

Una classe derivata è essenzialmente un’estensione di una classe base, da cui eredita property e metodi dichiarati public o protected, al fine di personalizzare, espandere le funzionalità, oppure modificare il comportamento dei metodi della classe base.   
Da una singola classe base è possibile derivare più classi, mentre C# non supporta l'ereditarietà multipla, ossia la possibilità per una classe di ereditare da più classi base. Il costruttore della classe derivata invoca, in modo implicito, quello della classe base.

In una classe è possibile, mediante l’overloding, assegnare a metodi diversi lo stesso nome, a condizione che abbiano una firma diversa, cioè numero e/o tipo di argomenti diversi nella dichiarazione.   
Le classi in C# sono polimorfiche, cioè è possibile definire metodi e property con lo stesso nome, in modo che una classe derivata possa ridefinire un metodo della classe base, assegnandogli lo stesso nome. Per usare il polimofismo occorre segnare il metodo, nella classe base, con la parola riservata virtual, al fine di indicare che è possibile avere, in una classe derivata, un metodo con lo stesso nome, mentre nella classe derivata il metodo, con il nome uguale a quello della classe base, deve essere segnato con la parola riservata override. Nella classe derivata si può nascondere il metodo della classe base, se il termine override è sostituito dalla parola riservata new. In ogni caso, nella classe derivata, è possibile attivare il metodo della classe base se all’identificatore si associa la parola riservata base. L'esempio che segue mostra l'utilizzo della classe base e di quella derivata:

      /\*                  Instanza della classe Cliente  
      --------------------------------------------------------------------------------------------------\*/   
 int id= 1;  
 Cliente cliente = new Cliente();  
      InfoClienti info = cliente.Read( id );  
      tCliente.Text = info.NomeCliente;

      /\*                  Definizione Dati Classe Cliente  
      --------------------------------------------------------------------------------------------------\*/   
      public classa InfoCliente   
      {   
            public int Id;   
            public string NomeCliente;  
            . . . .    
      }

      /\*              Definizione Classe Cliente derivata da quella base  
      --------------------------------------------------------------------------------------------------\*/    
   public classa Cliente : sqlDA

   {

public Cliente()

{

}

/\*       Lettura dati con Stored Procedure   
            --------------------------------------------------------------------------------------------\*/   
            public InfoCliente Read( int id )   
            {

       InfoCliente info = new InfoCliente();   
                   try {

            OpenConnection();

            SqlParameter[] parameters = { new SqlParameter ("@Id", SqlDbType.Int) };  
                        parameters[0].Value = id;   
                        DataSet ds = ActiveStoredProc ( "spCliente", parameters, "tdsCliente" );   
                        DataTable dt = ds.Tables["tdsCliente"];   
                        if ( dt.Rows.Count>0 ) {   
                              DataRow row = dt.Rows[0];   
                              info.Id = Convert.ToInt32(row["Id"]);   
                              info.Cliente = Convert.ToString( row["NomeCliente"] );   
                              . . . .   
                       }   
                  }   
                  catch (Exception ex) {   
                        MsgHandler.SetErrorMessage(ex.Message);   
                  }   
                  CloseConnection();   
                  return(info);   
            }

}

      /\*                  Definizione Stored Procedure  
      --------------------------------------------------------------------------------------------------\*/   
      USE [dbProgetto]   
      GO   
      CREATE PROCEDURE dbo.spCliente   
      @Id int AS   
      SELECT Id, NomeCliente, ….   
      FROM tClienti   
      WHERE Id=@Id GO

**9 - Esempi Read, Write, Insert, Update**

/\*-------------------------------------------------------------

INSERIMENTO DATI

--------------------------------------------------------------\*/

public void Insertdata()

{

string ConnString;

     ConnString= ConfigurationManager.AppSettings["dbIDEA"];

SqlConnection conn = newSqlConnection(ConnString);

conn.Open();

string insertString = @"INSERT INTO tPost”

+ (Titolo, Post, Autore, Data)"

+ " VALUES('Secondo Post', 'Inserimento dati',”

+ “ 'Antonio', '11/11/1960')" ;

SqlCommand cmd = newSqlCommand(insertString, conn);

cmd.ExecuteNonQuery();

if (conn != null)

conn.Close();

}

/\*-------------------------------------------------------------

UPDATE DATI

-------------------------------------------------------------\*/

public void UpdateData()   
{   
    string ConnString;   
 ConnString = ConfigurationManager.AppSettings["dbIDEA"];   
 SqlConnection conn = newSqlConnection(ConnString);   
      conn.Open();   
 string updateString =   
 "UPDATE tPost SET Titolo=Primo Post=Aggiornato WHERE Id=1";   
 SqlCommand cmd = newSqlCommand(updateString);   
 cmd.Connection = conn;   
 cmd.ExecuteNonQuery();   
 if (conn != null)   
 conn.Close();  
}

/\*------------------------------------------------------------

DELETE

------------------------------------------------------------\*/

public void DeleteData()

{

string ConnString;

   ConnString = ConfigurationManager.AppSettings["dbIDEA"];

SqlConnection conn = newSqlConnection(ConnString);

conn.Open();

string deleteString = @"DELETE FROM tPost WHERE Id=2";

SqlCommand cmd = newSqlCommand();

cmd.CommandText = deleteString;

cmd.Connection = conn;

cmd.ExecuteNonQuery();

if (conn != null)

conn.Close();

}

/\*-------------------------------------------------------------

-------------------------------------------------------------\*/

public void ReadWithParam(int inputId)

{

SqlDataReader dr = null;

    SqlCommand cmd;  
    string ConnString;

ConnString = ConfigurationManager.AppSettings["dbIDEA"];

SqlConnection conn = newSqlConnection(ConnString);

conn.Open();

cmd= newSqlCommand("SELECT \* FROM tPost where Id = @Id",conn);

SqlParameter param = newSqlParameter();

param.ParameterName = "@Id";

param.Value = inputId;

cmd.Parameters.Add(param);

dr = cmd.ExecuteReader();

string s = "";

while(dr.Read()) {

s += Convert.ToString(dr["Titolo"]) + "<br />";

}

Titolo.Text = s;

if (dr != null)

dr.Close();

if (conn != null)

conn.Close();

}

/\*-------------------------------------------------------------

Lettura dati

-------------------------------------------------------------\*/

public void ReadData()

{

string ConnString;  
     ConnString= ConfigurationManager.AppSettings["dbIDEA"];

SqlConnection conn = newSqlConnection(ConnString);

conn.Open();

//------------------------------------------------------

string queryString = "select Titolo from tPost";

SqlCommand cmd = newSqlCommand(queryString, conn);

SqlDataReader dr = cmd.ExecuteReader();

string s = "";

while (dr.Read()) {

s = s + Convert.ToString(dr[0]) + "<br>";

}

Titolo.Text = s;

if (dr != null)

dr.Close();

if (conn != null)

conn.Close();

}

/\*--------------------------------------------------------------

--------------------------------------------------------------\*/

public void RunStoredProc()

{

SqlDataReader dr = null;

int IdPost = 1;

string ConnString = ConfigurationManager.AppSettings["dbIDEA"];

SqlConnection conn = newSqlConnection(ConnString);

conn.Open();

SqlCommand cmd = newSqlCommand("spReadTitle", conn);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.Add(newSqlParameter("@Id", IdPost));

dr = cmd.ExecuteReader();

string s = "";

while (dr.Read()) {

s += Convert.ToString(dr["Titolo"]) + "<br />";

}

Titolo.Text = s;

if (dr != null)

dr.Close();

if (conn != null)

conn.Close();

}

**10 - Paginazione con ADO.NET**

In ADO.NET 2.0 è stato aggiunto una nuova funzionalità per realizzare il paging lato client, mediante il metodo ExecutePageReader della classe SqlDataReader:

       SqlCommand cmd;   
       SqlDataReader dr;   
       sqlConn.Open();

       cmd = new SqlCommand("Select \* from tChannel", sqlConn);   
       dr = cmd.ExecutePageReader(CommandBehavior.CloseConnection, 1, 10);

Il parametro opzionale CommandBehavior.CloseConnection chiude la connessione al database dopo la chiusura del datareader.