

## Introduzione

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

ADO.NET 2

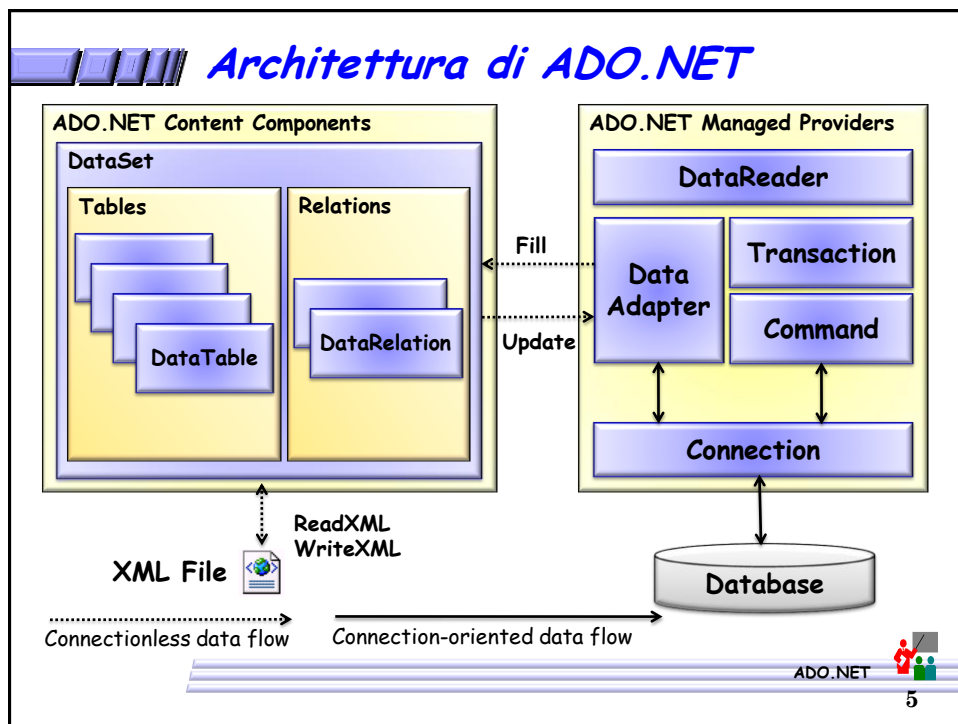
## Universal Data Access

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

ADO.NET 3


## Data Provider


ADO.NET 4



## Connection-oriented vs Connectionless



- Due diverse modalità di accesso ai dati:
  - **Connection-oriented**
    - mantiene **attiva la connessione** al database;
    - i **dati** sono sempre **aggiornati**;
    - utile per applicazioni con le seguenti caratteristiche:
      - transazioni brevi;
      - pochi accessi paralleli;
      - necessità di dati sempre aggiornati.
  - **Connectionless**
    - non utilizza una connessione permanente al db;
    - i **dati** sono caricati in **memoria centrale**;
    - le modifiche apportate ai dati in memoria centrale non sono immediatamente memorizzate nel db;
    - utile per applicazioni che richiedono **numerosi accessi contemporanei** al database e che eseguono transazioni di lunga durata (es. applicazioni web).


ADO.NET  6





## ADO.NET namespace

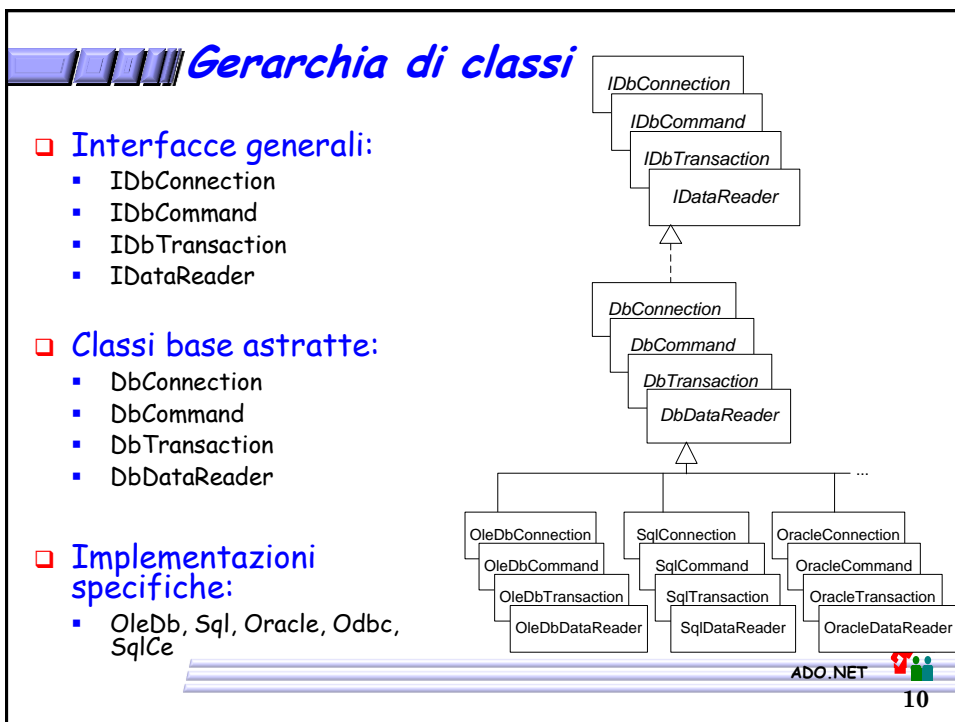
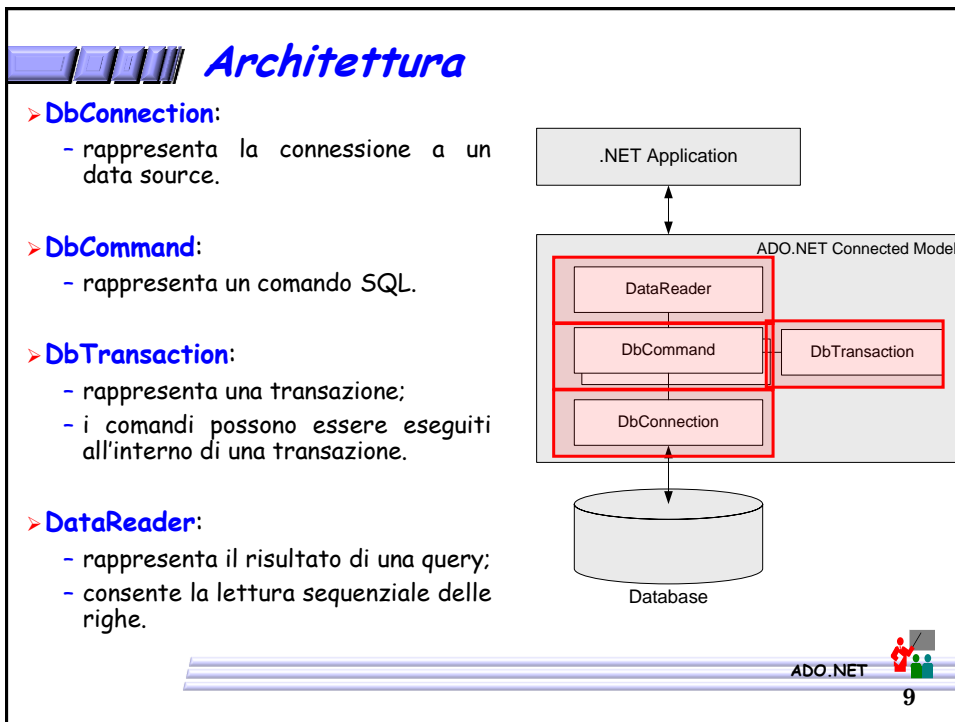
System.Data	Fornisce l'accesso alle classi che rappresentano l'architettura di ADO.NET
System.Data.Common	Classi condivise dai diversi DataProvider
System.Data.OleDb	Classi che permettono la connessione a sorgenti dati OLE DB compliant
System.Data.SqlClient	Classi ottimizzate per la connessione a Microsoft®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

ADO.NET  
7



## ACCESSO CONNECTION-ORIENTED

ADO.NET  
8



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

Dichiarazione della connessione  

```
try {
```

Richiesta connessione al database

Esecuzione comandi SQL

Elaborazione risultati

```
        Rilascio risorse  

    } catch ( Exception ) { Gestione eccezioni }  

finally  

{  

    try {  

        Chiusura della connessione  

    } catch (Exception) { Gestione eccezioni }  

}
```

ADO.NET

11

## Interfaccia IDbConnection

La *ConnectionString* contiene i dati necessari per stabilire la connessione con il database.

```
string ConnectionString {get; set;}
```

Apertura e chiusura della connessione

```
void Open();  
void Close();
```

Proprietà dell'oggetto connessione

```
string Database {get;}  
int ConnectionTimeout {get;}  
ConnectionState State {get;}
```

Creazione dell'oggetto *Command*

```
IDbCommand CreateCommand();
```

Creazione dell'oggetto *Transaction*

```
IDbTransaction BeginTransaction();  
IDbTransaction  
BeginTransaction(IsolationLevel lvl);
```

**<<interface>>**  
**IDbConnection**

```
//----- Properties  
string ConnectionString  
    {get; set;}  
string Database {get;}  
int ConnectionTimeout {get;}  
ConnectionState State {get;}  
  
//----- Methods  
IDbTransaction  
BeginTransaction();  
IDbTransaction BeginTransaction  
    (IsolationLevel lvl);  
  
IDbCommand CreateCommand();  
  
void Close();  
void Open();  
  
...
```

↓

```
public enum ConnectionState {  
    Broken, Closed,  
    Connecting, Executing,  
    Fetching, Open  
}
```

ADO.NET

12

## *IDbConnection: connection string*

- ❑ Sintassi: coppie chiave-valore separate da punto e virgola (;)
- ❑ Configurazione della connessione:
  - nome del provider;
  - identificazione del data source;
  - credenziali utente;
  - altre impostazioni specifiche del DBMS.

### Es. OleDb

```
"provider=Microsoft.Jet.OLEDB.4.0;data source=c:\bin\LocalAccess40.mdb;"  
"provider=MSDAORA; data source=ORACLE8i7; user id=OLEDB; password=OLEDB;"
```

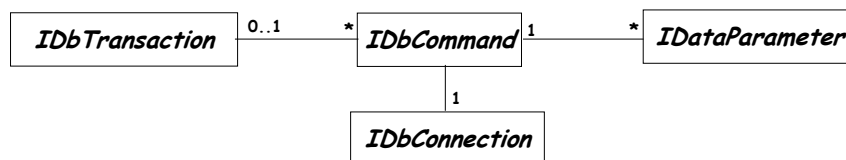
### Es. MS SQL-Server

```
"data source=(local)\NetSDK; initial catalog=Northwind; user id=sa;  
pooling=false; Integrated Security=SSPI; connection timeout=20;"
```

ADO.NET 

13

## *Oggetto Command*



L'oggetto **Command** definisce **comandi SQL** o stored procedure. Ha le seguenti caratteristiche:

- è eseguito su una connessione;
- può avere parametri;
- può far parte di una transazione.

ADO.NET 

14

## Interfaccia IDbCommand

Il *CommandText* specifica il comando SQL o la stored procedure.

```
string CommandText {get; set;}
```

Oggetto *Connection*

```
IDbConnection Connection {get; set;}
```

Proprietà dell'oggetto *Command*

```
CommandType CommandType {get; set;}
```

```
int CommandTimeout {get; set;}
```

Creazione e accesso ai parametri

```
IDbDataParameter CreateParameter();
```

```
IDataParameterCollection Parameters {get;}
```

Esecuzione del comando

```
IDataReader ExecuteReader();
```

```
IDataReader ExecuteReader(CommandBehavior b);
```

```
int ExecuteNonQuery();
```

```
object ExecuteScalar();
```

```
<<interface>>
IDbCommand

//----- Properties
string CommandText {get; set;}
CommandType CommandType {get; set;}
int CommandTimeout {get; set;}
IDbConnection Connection {get; set;}
IDataParameterCollection Parameters {get;}
IDbTransaction Transaction {get; set;}
...

//----- Methods
IDbDataParameter CreateParameter();
IDataReader ExecuteReader();
IDataReader ExecuteReader(CommandBehavior b);
object ExecuteScalar();
int ExecuteNonQuery();
...
```

```
public enum CommandType {
    Text,
    StoredProcedure,
    TableDirect
}
```

ADO.NET

15

## Il metodo ExecuteReader

```
IDataReader ExecuteReader()
```

```
IDataReader ExecuteReader( CommandBehavior behavior );
```

```
public enum CommandBehavior {
    CloseConnection, Default, KeyInfo, SchemaOnly,
    SequentialAccess, SingleResult, SingleRow
}
```

- ❑ Esegue la query specificata in *CommandText*.
- ❑ Restituisce il risultato in un oggetto di tipo *IDataReader*.
- ❑ Esempio:

```
cmd.CommandText =
"SELECT EmployeeID, LastName, FirstName FROM Employees ";
IDataReader reader = cmd.ExecuteReader();
```

ADO.NET

16



## Il metodo *ExecuteNonQuery*

```
int ExecuteNonQuery();
```

- ❑ Esegue le operazioni specificate in *CommandText* che non sono interrogazioni:
  - ❑ *CreateTable*;
  - ❑ *Insert*;
  - ❑ *Delete*;
  - ❑ *Update*;
  - ❑ ...
- ❑ Il risultato restituito è il numero di record interessati dall'operazione.
- ❑ Esempio:

```
cmd.CommandText="UPDATE Empls SET City = 'Seattle' WHERE iD=8";  
int affectedRows = cmd.ExecuteNonQuery();
```

ADO.NET



17

## Il metodo *ExecuteScalar*

```
object ExecuteScalar();
```

- ❑ Il risultato è il valore della prima colonna della prima riga restituita dalla query.
- ❑ *CommandText* contiene tipicamente una funzione aggregata.
- ❑ Esempio:

```
cmd.CommandText = " SELECT count(*) FROM Employees ";  
int count = (int) cmd.ExecuteScalar();
```

ADO.NET



18

## Parametri

❑ Gli oggetti *Command* possono utilizzare parametri di input e output.

```
IDataParameterCollection Parameters {get;}
```

❑ Gli oggetti *Parameter* specificano:

- **Name** - nome del parametro;
- **Value** - valore del parametro;
- **DbType** - tipo di dato;
- **Direction** - direzione del parametro:
  - » Input;
  - » Output;
  - » InputOutput;
  - » ReturnValue.

```

<<interface>>
IDbCommand
...
IDataParameterCollection
Parameters {get;}
...

Parameters *
<<interface>>
IDataParameter
//----- Properties
DbType DbType {get; set;}
ParameterDirection Direction {get; set;}
string ParameterName {get; set;}
object Value {get; set;}
...

<<interface>>
IDbDataParameter
//----- Properties
int Size {get; set;}
...
      
```

19

## Utilizzare i parametri

- Definire il comando SQL con "place holder":
 

OLEDB: Identificazione dei parametri tramite la posizione (notazione: "?")

```
OleDbCommand cmd = new OleDbCommand();
cmd.CommandText = "DELETE FROM Empls WHERE EmployeeID = ?";
```

SQL Server: Identificazione dei parametri tramite il nome (notazione: @name)

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "DELETE FROM Empls WHERE EmployeeID = @ID";
```
- Creare e aggiungere un parametro:
 

```
cmd.Parameters.Add( new OleDbParameter("@ID",
OleDbType.BigInt));
```
- Assegnare un valore al parametro ed eseguire il comando:
 

```
cmd.Parameters["@ID"].Value = 1234;
cmd.ExecuteNonQuery();
```

20

## Interfaccia IDataReader

Lettura della riga successiva (lettura sequenziale):

```
bool Read();
```

Accesso ai valori delle colonne tramite Indexer:

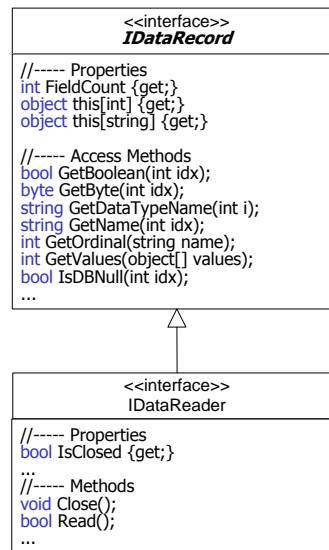
```
object this[int] {get;}
object this[string] {get;};
```

Accesso tipizzato ai valori delle colonne tramite specifici metodi di accesso:

```
bool GetBoolean(int idx);
byte GetByte(int idx);
...
```

Recupero di meta-informazioni:

```
string GetDataTypeName(int i);
string GetName(int idx);
int GetOrdinal(string name);
...
```



ADO.NET

21

## Utilizzare un IDataReader

- ❑ Creazione dell'oggetto IDataReader e lettura delle righe

```
IDataReader reader = cmd.ExecuteReader();
while (reader.Read()) {
```

- ❑ Lettura dei valori delle colonne e memorizzazione in un array

```
object[] dataRow = new object[reader.FieldCount];
int cols = reader.GetValues(dataRow);
```

- ❑ Lettura dei valori delle colonne tramite indexer

```
object val0 = reader[0];
object nameVal = reader["LastName"];
```

- ❑ Lettura del valore di una colonna con accesso tipizzato

```
string firstName = reader.GetString(2);
```

- ❑ Chiusura dell'oggetto IDataReader

```
reader.Close();
```

ADO.NET

22

## Un esempio completo (1)

Visualizzare i prodotti che hanno un prezzo unitario superiore a un valore prefissato (parametro) in ordine decrescente di prezzo unitario.

ConnectionString  
per db Access

```
string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
+ "c:\\db\\Northwind.mdb;User Id=admin;Password=";

// Provide the query string with a parameter placeholder.
string queryString =
    "SELECT ProductID, UnitPrice, ProductName from products "
    + "WHERE UnitPrice > ?"
    + "ORDER BY UnitPrice DESC;";

// Specify the parameter value.
int paramValue = 5;
```

QueryString con  
parametro

ADO.NET

23

## Un esempio completo (2)

Creazione  
oggetti  
Command e  
Parameter

Inizializzazione  
e riempimento  
DataGridView

```
using (OleDbConnection connection =
    new OleDbConnection(connectionString))
{
    // Create the Command and Parameter objects.
    OleDbCommand command = new OleDbCommand(queryString, connection);
    command.Parameters.AddWithValue("@pricePoint", paramValue);

    try
    {
        // Open the connection in a try/catch block.
        connection.Open();

        // Create and execute the DataReader, visualizing the result in
        // a DataGridView.
        OleDbDataReader reader = command.ExecuteReader();
        // Initializes the DataGridView
        dataGridViewResult.ColumnCount = reader.FieldCount;
        for (int i = 0; i < dataGridViewResult.ColumnCount; i++)
        {
            dataGridViewResult.Columns[i].Name = reader.GetName(i);
        }
        Object[] row = new Object[reader.FieldCount];
        while (reader.Read())
        {
            reader.GetValues(row);
            dataGridViewResult.Rows.Add(row);
        }
        reader.Close();
    }
    catch (Exception ex)
    { }
}
```

Creazione  
connessione

Apertura  
connessione

ProductID	UnitPrice	ProductName
38	263.5	Côte de Blaye
29	123.79	Thüringer Rostbr...
9	97	Mishi Kobe Niku
20	81	Sir Rodney's Mar

ADO.NET

24



# ACCESSO CONNECTIONLESS

ADO.NET



25



## Motivazioni

### □ Motivazioni:

- in alcuni casi sono necessari **numerosi accessi paralleli** al database e **operazioni di lunga durata**;
- l'accesso di tipo connection-oriented risulta troppo **oneroso** e inadatto a queste situazioni.

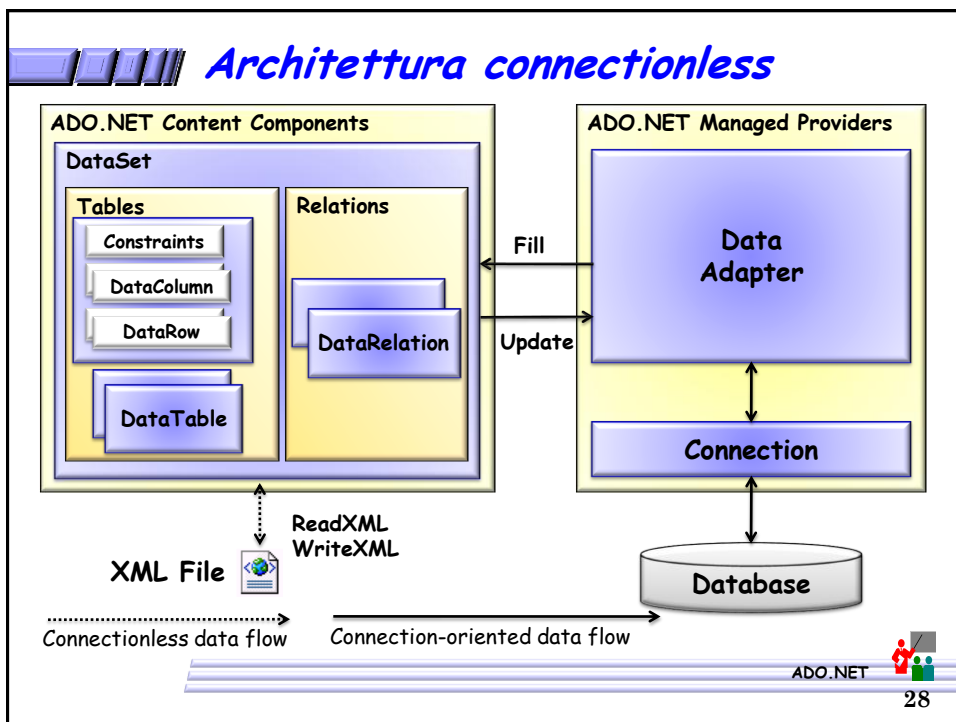
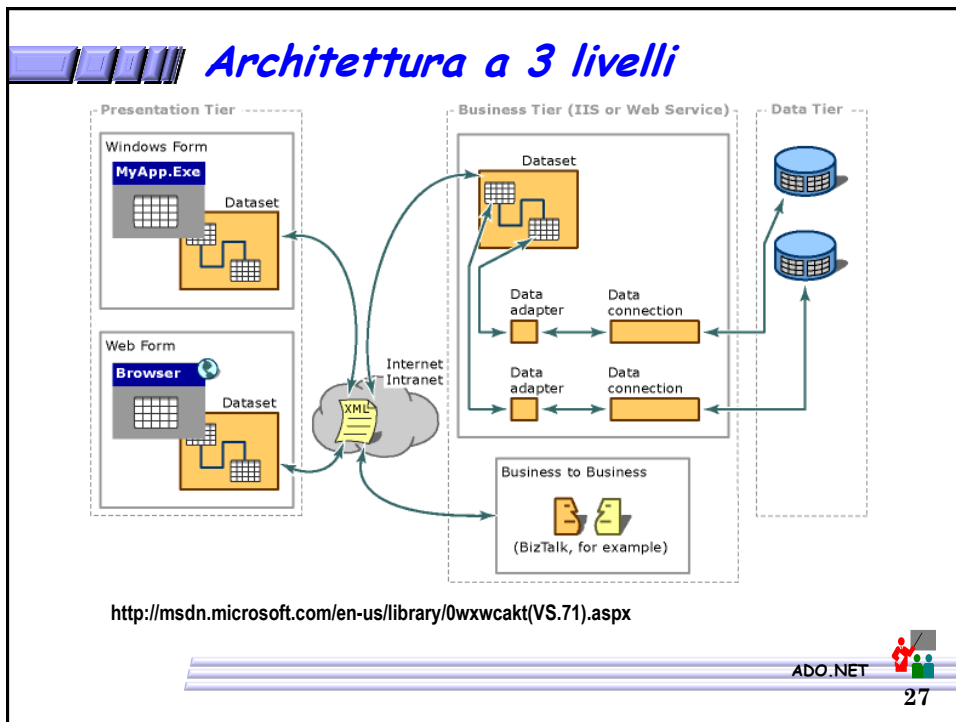
### □ Idea:

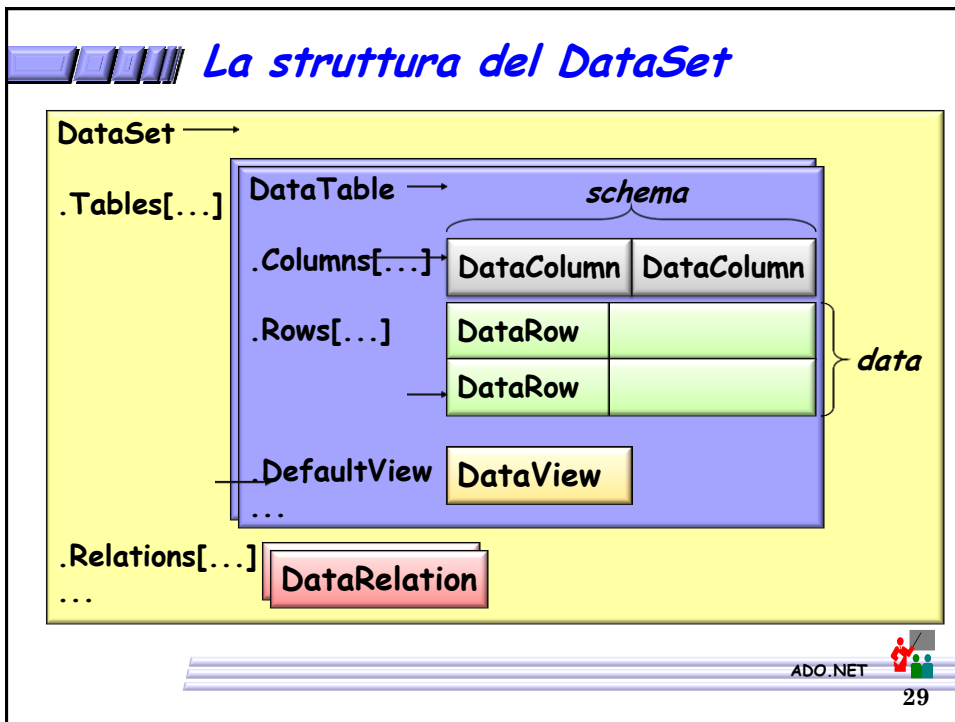
- caricare i dati in **memoria centrale**;
  - ✚ Main Memory Data Base
- effettuare solo **brevi connessioni** al db per la **lettura** e l'**aggiornamento** dei dati;
  - ✚ DataAdapter
- il database in memoria centrale è indipendente dal data source;
  - ✚ possono nascere **conflitti** in fase di aggiornamento dati.


ADO.NET

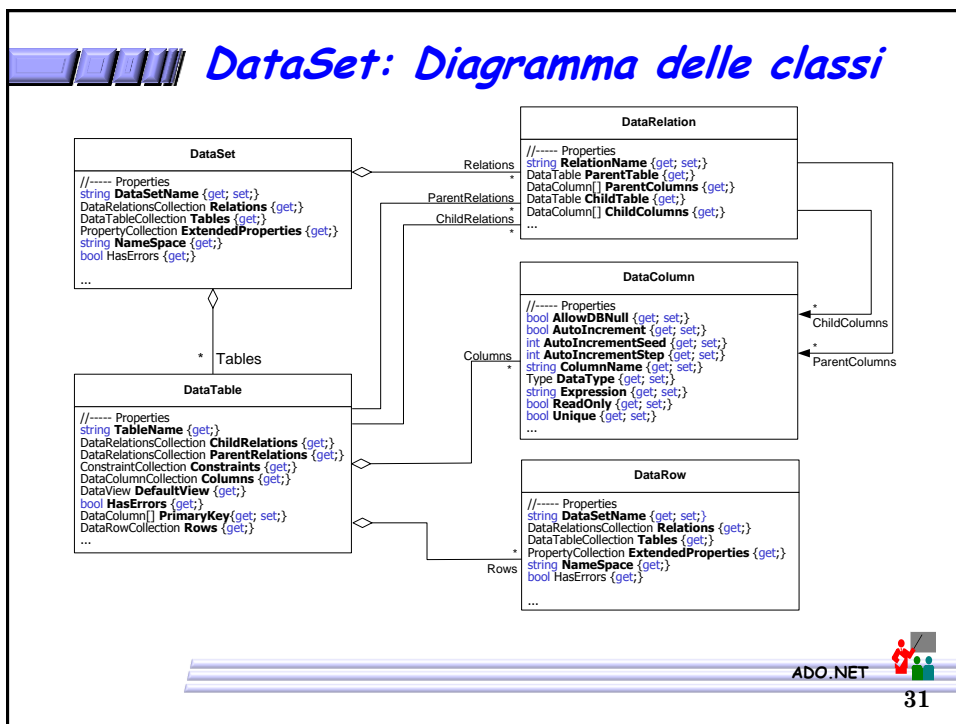


26






- ## Il DataSet
- ❑ Rappresenta un database in **memoria centrale**:
    - struttura **relazionale**;
    - interfaccia **object-oriented**.
  - ❑ Un **DataSet** è costituito da:
    - una collection di *DataTable*;
    - Una collection di *DataRelation*.
  - ❑ Una **DataTable** contiene:
    - una collezione di *DataTableColumn* (= schema della relazione);
    - una collezione di *DataTableRows* (= istanza della relazione);
    - defaultView (*DataTableView*).
  - ❑ Una **DataRelation** ha le seguenti caratteristiche:
    - associa due oggetti di tipo *DataTable*;
    - definisce una *ParentTable* e *ParentColumns*, e una *ChildTable* e *ChildColumns*.
- ADO.NET  30



## DataSet: classe DataTable

- ❑ Oggetto che rappresenta una tabella in memoria centrale. È costituito da:
  - colonne;
  - righe.
- ❑ Lo schema della tabella è definito dalla collection *Columns*.
- ❑ I vincoli di integrità dei dati sono espressi attraverso gli oggetti *Constraint*.
- ❑ Eventi pubblici:
  - modifica/cancellazione di righe;
  - modifica di colonne.

ADO.NET  32



## DataSet: classe DataColumn

- ❑ Elemento costitutivo dello schema di una DataTable (contenuti nella collection *Columns*).
- ❑ Definisce quale tipo di dato può essere inserito tramite la proprietà *DataType*.
- ❑ Altre importanti proprietà sono *AllowNull*, *Unique*, *ReadOnly*.
- ❑ Possono contenere vincoli, ovvero una collection di *Constraints*.
- ❑ Possono contenere relazioni (*Relations*).

ADO.NET



33

## DataSet: classe DataRow

- ❑ Rappresentano i dati in una DataTable (contenuti nella collection *Rows*).
- ❑ Ciascuna riga deve essere conforme allo schema definito dalle *DataColumns*.
- ❑ La classe dispone di proprietà che permettono di determinare lo stato delle righe (es. *new*, *changed*, *deleted*, ...).
- ❑ Tutte le operazioni di inserimento/modifica devono essere "committed" con il metodo *AcceptChanges* della DataTable.

ADO.NET



34

## *DataSet: classe DataRelation*

- ❑ Collega due DataTable attraverso DataColumnns.
- ❑ Le due colonne collegate devono avere lo **stesso DataType**.
- ❑ Gli aggiornamenti apportati a una tabella possono ripercuotersi a catena (politica *cascade*) alla tabella "figlia".
- ❑ Non sono permesse modifiche al database che invalidino la relazione.

ADO.NET



35

## *DataSet: interfaccia IDataAdapter*

- ❑ Interfaccia che popola o invia aggiornamenti a un DataSet.
- ❑ L'interfaccia ha implementazioni specifiche per OleDb (*OleDbDataAdapter*) e Sql (*SqlDataAdapter*).
- ❑ Non è basata su connessione (connectionless); utilizza un approccio **asincrono**.
- ❑ È un "superset" dell'oggetto Command.
- ❑ Contiene quattro oggetti Command di default: *Select*, *Insert*, *Update* e *Delete*.

ADO.NET



36

## Esempi DataSet: caricamento e visualizzazione dati

```

string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
+ "c:\\db\\Northwind.mdb;User Id=admin;Password=";

OleDbConnection connection = new OleDbConnection(connectionString);
string queryString = "SELECT CustomerID, CompanyName FROM Customers "
+ "ORDER BY CompanyName";

try
{
    // Open connection
    connection.Open();
    // Create DataAdapter with a specific query on the connection
    OleDbDataAdapter dataAdapter = new OleDbDataAdapter();
    dataAdapter.SelectCommand = new OleDbCommand(queryString, connection);
    // Fill a DataSet with the result
    DataSet ds = new DataSet();
    dataAdapter.Fill(ds, "Customers");
    //Close connection
    connection.Close();
    // Visualize results
    dataGridViewResult.DataSource = ds;
    dataGridViewResult.DataMember = "Customers";
}
catch (Exception)
{
    throw;
}

```

**Creazione di un DataAdapter con una query specifica sulla connessione attiva**

**Riempimento dataset**

**Visualizzazione del risultato in una DataGridView**

CustomerID	CompanyName
ALFKI	Alfreds Futterkiste
ANATR	Ana Trujillo Empa...
ANTON	Antonio Moreno ...
AROUT	Around the Hom

ADO.NET

37

## Esempi DataSet: aggiornamento

```

// Create a new Connection and OleDbDataAdapter
OleDbConnection myConnection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
+ "c:\\tmp\\Northwind.mdb;User Id=admin; Password=");
OleDbDataAdapter myOleDbDataAdapter = new OleDbDataAdapter("SELECT CustomerID, CompanyName, "
+ "ContactName from Customers", myConnection);
DataSet myDataSet = new DataSet();

// Create command builder. This line automatically generates the update commands for you,
// so you don't have to provide or create your own.
OleDbCommandBuilder myOleDbCommandBuilder = new OleDbCommandBuilder(myOleDbDataAdapter);

// Set the MissingSchemaAction property to AddWithKey because Fill will not cause primary
// key & unique key information to be retrieved unless AddWithKey is specified.
myOleDbDataAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
myOleDbDataAdapter.Fill(myDataSet, "Customers");

DataRow myDataRow = myDataSet.Tables["Customers"].NewRow();
myDataRow["CustomerId"] = "NewID";
myDataRow["ContactName"] = "New Name";
myDataRow["CompanyName"] = "New Company Name";
myDataSet.Tables["Customers"].Rows.Add(myDataRow);

DataRow myDataRow1 = myDataSet.Tables["Customers"].Rows.Find("ALFKI");
myDataRow1["ContactName"] = "Peach";

myOleDbDataAdapter.Update(myDataSet, "Customers");

```

**Creazione automatica UPDATE command**

**Aggiunta nuovo record**

**Aggiornamento record esistente**

**Aggiornamento database**

ADO.NET

38

## Esempi DataSet: creazione da codice

```

DataSet dataset = new DataSet();
dataset.DataSetName = "BookAuthors";

DataTable authors = new DataTable("Author"); DataTable books = new DataTable("Book");

DataColumn id = authors.Columns.Add("ID", typeof(Int32));
id.AutoIncrement = true;
authors.PrimaryKey = new DataColumn[] {id};
DataColumn name = authors.Columns.Add("Name", typeof(String));
DataColumn isbn = books.Columns.Add("ISBN", typeof(String));
books.PrimaryKey = new DataColumn[] {isbn};
DataColumn title = books.Columns.Add("Title", typeof(String));
DataColumn authid = books.Columns.Add("AuthID", typeof(Int32));
DataColumn[] foreignkey = new DataColumn[] {authid};

dataset.Tables.Add(authors); dataset.Tables.Add(books);

DataRelation bookauth = new DataRelation("BookAuthors", authors.PrimaryKey, foreignkey);
dataset.Relations.Add(bookauth);

DataRow shkspr = authors.NewRow();
shkspr["Name"] = "William Shakespeare";
authors.Rows.Add(shkspr);

DataRow row = books.NewRow();
row["AuthID"] = shkspr["ID"]; row["ISBN"] = "1000-XYZ"; row["Title"] = "MacBeth";
books.Rows.Add(row);

dataset.AcceptChanges();

```

**Creazione tabelle**

**Definizione dello schema**

**Aggiunta delle tabelle al dataset**

**Definizione foreign key**

**Inserimento record**

**Salvataggio modifiche nel dataset**

ADO.NET

39

## DataSet: accesso ai dati e gestione delle modifiche

➤ Le *DataRelation* permettono di reperire i record collegati in tabelle diverse.

```

DataRow[] shakespeare = authors.Select("Name = 'William Shakespeare'");
DataRow[] shakespeareBooks = shakespeare[0].GetChildRows(bookauth);

```

➤ Per la gestione delle modifiche il DataSet tiene traccia di tutti gli **aggiornamenti** effettuati. Gli aggiornamenti possono essere:

- accettati con *AcceptChanges*;
- rifiutati con *RejectChanges*.

```

if (dataset.HasErrors)
    dataset.RejectChanges();
else
    dataset.AcceptChanges();

```

ADO.NET

40

```
dataset.WriteXml(@"c:\provaDataSet.xml", XmlWriteMode.WriteSchema);
```

IgnoreSchema	Scrive il contenuto corrente del dataset in formato XML, ma non memorizza lo schema dei dati. Questo è il default.
WriteSchema	Scrive il contenuto corrente del dataset e lo schema delle relazioni in formato XML.
DiffGram	Scrive schema e istanza come un <i>DiffGram</i> , includendo i valori originali e quelli correnti.

```
<xs:element name="Author">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Const" type="xs:string" base="xs:string" minOccurs="1" maxOccurs="1" />
      <xs:element name="BookAuthors" type="xs:string" base="xs:string" minOccurs="1" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```



41