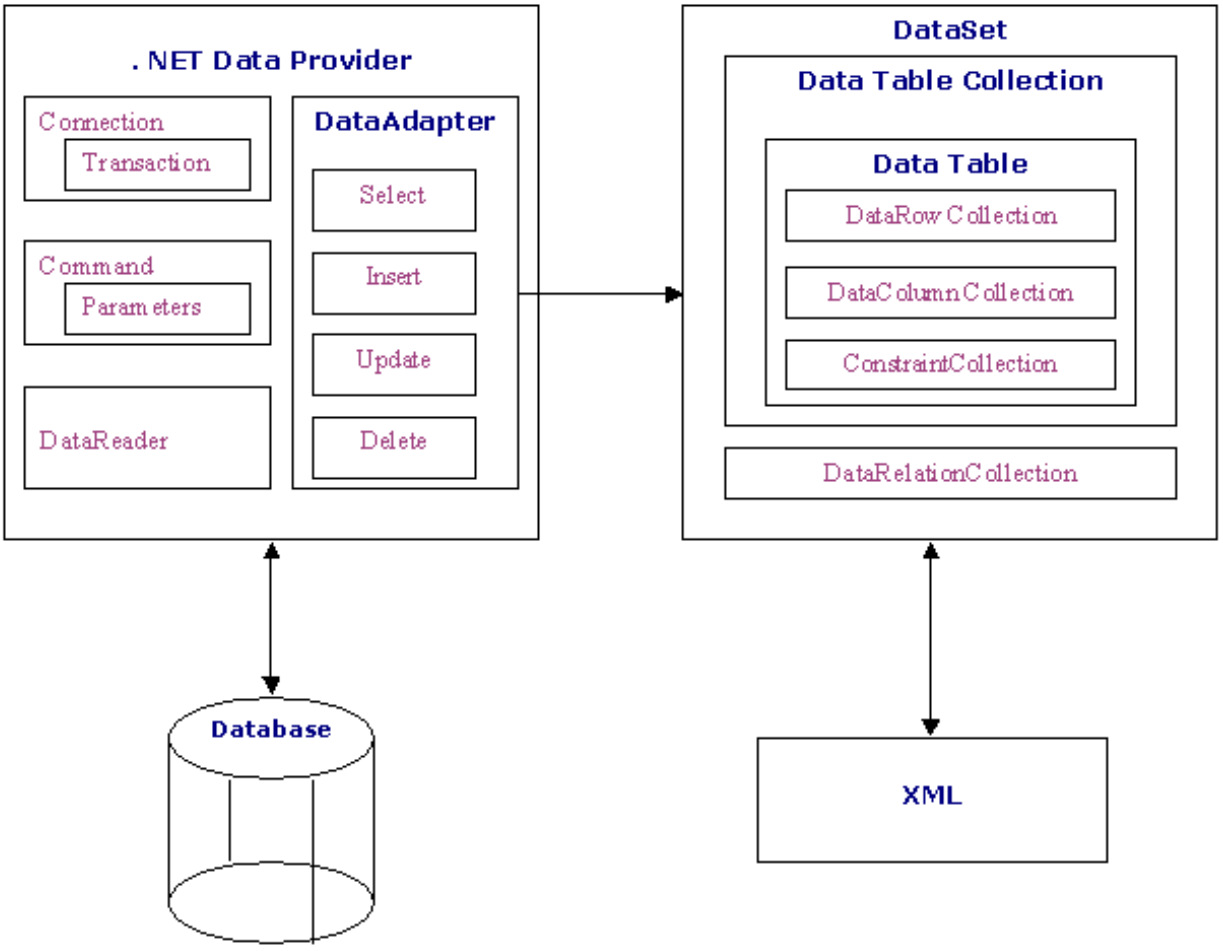
ADO .NET

Arhitectura ADO.NET



ADO .NET Data Architecture

ADO.NET

ADO.NET este compus dintr-o multime de clase ce expun servicii de acces la date sub platforma .NET.

Obiecte «conectate »

Connection
Transaction
DataAdapter
Command
Parameter
DataReader

Obiecte “deconectate”

DataSet
DataTable DataView
DataRow
DataColumn
Constraint
DataRelation

Furnizori de date .NET (.NET Data Providers) :

Ole DB .NET Data Provider ;
ADO.NET Data Provider ;
Firebird ADO.NET Data Provider ; etc.

Obiectul Connection

Un obiect **Connection** reprezinta o conexiune la sursa de date. Putem specifica tipul sursei de date, locul unde se afla sursa de date, user, password, obiecte de securitate, etc.

Un obiect de conexiune este folosit de obiectele **DataAdapter**, **Command**.

```
// Open and close a connection using the
// OLE DB .NET Data Provider.

OleDbConnection cnOleDb = new OleDbConnection();
cnOleDb.ConnectionString = "Provider=Provider=SQLOLEDB;
"Data Source=(local);InitialCatalog=Northwind;...";
cnOleDb.Open();
...
cnOleDb.Close();

// Open and close a connection using the
// SQL Client .NET Data Provider.

SqlConnection cnSql = new SqlConnection();
cnSql.ConnectionString = "Data Source=(local);" +"Initial Catalog =
Northwind;...";
cnSql.Open();
...
cnSql.Close();

// conectare la serverul Firebird
public static FbConnection Connect(DBProxy conexiune)
{
  FbConnectionStringBuilder cs = new FbConnectionStringBuilder();
// cache parametri pentru a crea o noua conexiune la bd
      cs.DataSource = conexiune.Db_source;
      cs.Port = Convert.ToInt16(conexiune.Port);
      cs.Database = conexiune.UserDatabase;
      cs.UserID = conexiune.User;
      cs.Password = conexiune.Parola;
      cs.Dialect = 3;

      FbConnection connection = new FbConnection(cs.ToString());
                  try
                  {
                        connection.Open();
                        return connection;
                  }
                  catch(Exception ex)
                  {
                        // jurnalizare esec ...
                        return null;
                  }
}
```

Clasa **SqlConnection – pentru SQL Server**

Pentru SQL Server clasa pentru conexiune la sursa de date este **`SqlConnection`**, namespace **`System.Data.SqlClient`**.

Constructori

`SqlConnection`	Overloaded. Initializes a new instance of the SqlConnection class.

Proprietati

`ConnectionString`	Gets or sets the string used to open a SQL Server database.
`ConnectionTimeout`	Gets the time to wait while trying to establish a connection before terminating the attempt and generating an error.
`Database`	Gets the name of the current database or the database to be used after a connection is opened.
`DataSource`	Gets the name of the instance of SQL Server to which to connect.
`PacketSize`	Gets the size (in bytes) of network packets used to communicate with an instance of SQL Server.

Metode

`BeginTransaction`	Overloaded. Begins a database transaction.
`ChangeDatabase`	Changes the current database for an open **SqlConnection**.
`Close`	Closes the connection to the database. This is the preferred method of closing any open connection.
`CreateCommand`	Creates and returns a SqlCommand object associated with the **SqlConnection**.
`CreateObjRef (inherited from MarshalByRefObject)`	Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
`Dispose (inherited from Component)`	Overloaded. Releases the resources used by the Component.
`Open`	Opens a database connection with the property settings specified by the ConnectionString.

Obiectul Command

Observatie
Pentru SQL Server clasa se numeste **SqlCommand**.

Aceste obiecte pot reprezenta o cerere la baza de date, un apel de procedura stocata, sau o cerere directa pentru a returna continutul unei anumite tabele din baza de date.

Cererile pot fi de tip **`Select, Insert, Update, Delete`** sau de creare / modificare / stergere tabele, view-uri, proceduri stocate, triggere, constrangeri.

Obiectul **`Command`** prezinta diferite modalitati de a executa o cerere:

Item	Description
`BeginExecuteNonQuery`	Initiates the asynchronous execution of the Transact-SQL statement or stored procedure that is described by this SqlCommand, generally executing commands such as INSERT, DELETE, UPDATE, and SET statements. Each call to BeginExecuteNonQuery must be paired with a call to EndExecuteNonQuery which finishes the operation, typically on a separate thread.
`BeginExecuteReader`	Initiates the asynchronous execution of the Transact-SQL statement or stored procedure that is described by this SqlCommand and retrieves one or more results sets from the server. Each call to BeginExecuteReader must be paired with a call to EndExecuteReader which finishes the operation, typically on a separate thread.
`BeginExecuteXmlReader`	Initiates the asynchronous execution of the Transact-SQL statement or stored procedure that is described by this SqlCommand. Each call to BeginExecuteXmlReader must be paired with a call to EndExecuteXmlReader, which finishes the operation, typically on a separate thread, and returns an XmlReader object.
`ExecuteReader`	Executes commands that return rows. For increased performance, ExecuteReader invokes commands using the Transact-SQL sp_executesql system stored procedure. Therefore, ExecuteReader might not have the effect that you want if used to execute commands such as Transact-SQL SET statements.
`ExecuteNonQuery`	Executes commands such as Transact-SQL INSERT, DELETE, UPDATE, and SET statements.
`ExecuteScalar`	Retrieves a single value (for example, an aggregate value) from a database.
`ExecuteXmlReader`	Sends the CommandText to the Connection and builds an XmlReader object.

`Proprietati`

CommandText	Gets or sets the Transact-SQL statement or stored procedure to execute at the data source.
CommandTimeout	Gets or sets the wait time before terminating the attempt to execute a command and generating an error.
CommandType	Gets or sets a value indicating how the CommandText property is to be interpreted.
Connection	Gets or sets the SqlConnection used by this instance of the **SqlCommand**.
Parameters	Gets the SqlParameterCollection.
Transaction	Gets or sets the SqlTransaction within which the **SqlCommand** executes.
UpdatedRowSource	Gets or sets how command results are applied to the DataRow when used by the Update method of the DbDataAdapter.

Observatie
Obiectul **Command** poate fi reutilizat, in timp ce obiectul **DataReader** trebuie inchis inainte de a executa o noua comanda.

Exemplu:

```
// sterg articole din combo box
// conexiune mentine informatii pentru a deschide
// o noua conexiune la baza de date
this.combo_Amplasare.Items.Clear();
string str = "Select id_locmunca, denumire From LocMunca" +
     " order by Denumire";
if (!this.conexiune.IsConnected())
     this.conexiune.Connect();
FbCommand cmd = new FbCommand(str, this.conexiune.Conn);
FbDataReader dr = cmd.ExecuteReader();
try
{
  while (dr.Read())
  {

      this.combo_Amplasare.Items.Add(dr.GetInt16(0).ToString() +
           " | " + dr.GetString(1).Trim());
  }
}
catch(Exception ex)
{
     // cod...
}
finally
{
     dr.Close(); // inchidere DataReader
     cmd.Dispose();
     this.conexiune.Close(); // inchidere conexiune
}
```

*Exemplu cu **ExecuteScalar***

```
// Select trebuie sa fie de tip singleton, adica sa returneze cel mult o inregistrare
string stmt = "Select count(*) From Amo " +
      " where trim(nrinv) = '12'";
if (!this.conexiune.IsConnected())
     this.conexiune.Connect();

FbCommand cmd = new FbCommand(stmt, this.conexiune.Conn);
object obj = cmd.ExecuteScalar();
// utilizare obj ...
cmd.Dispose();
this.conexiune.Close();
```

*Exemplu cu **ExecuteNonQuery***

```
string stmt = "Insert Into Amo(nrinv,denumire) values('inv1'" +
 ",'Laptop')";
if (!this.conexiune.IsConnected())
     this.conexiune.Connect();
// Incep o noua tranzactie
FbTransaction tx = conexiune.Conn.BeginTransaction();
FbCommand cmd = new FbCommand(stmt, this.conexiune.Conn);
try
{
      cmd.ExecuteNonQuery();
      // Comanda executata cu succes.
      // Acceptam modificarile in baza de date.
      tx.Commit();
}
catch(Exception ex)
{
      // ...
      // Nu acceptam modificarile in baza de date.
      tx.RollBack();
}
finally
{
      cmd.Dispose();
      this.conexiune.Close();
}
```

*Exemplu cu proprietatea **Parameters (MSDN)***

```
private static void UpdateDemographics(Int32 customerID,
    string demoXml, string connectionString)
{
    // Update the demographics for a store, which is stored
    // in an xml column.
    string commandText = "UPDATE Sales.Store SET " +
       " Demographics = @demographics "
        + "WHERE CustomerID = @ID;";

    using (SqlConnection conn = new SqlConnection(connString))
```

```
{
    SqlCommand command = new SqlCommand(commandText, conn);
    command.Parameters.Add("@ID", SqlDbType.Int);
    command.Parameters["@ID"].Value = customerID;

    // Use AddWithValue to assign Demographics.
    // SQL Server will implicitly convert strings into XML.
    command.Parameters.AddWithValue("@demographics", demoXml);
    try
    {
        conn.Open();
        Int32 rowsAffected = command.ExecuteNonQuery();
        Console.WriteLine("RowsAffected: {0}", rowsAffected);
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

Obiectul `DataReader`

Este proiectat pentru a regasi si examina inregistrarile returnate de cererea facuta serverului bazei de date.

DataReader creaza un « cursor » ce poate fi parcurs (o singura data) secvential de la inceput catre sfarsit.
DataReader nu suporta update.
Valorile returnate de **DataReader** sunt *read only*.

Pentru a crea un **`DataReader`**, trebuie sa apelam metoda **`ExecuteReader`** pe obiectul **`Command`**.

Observatie
Obiectul **DataReader** trebuie inchis cat mai repede posibil. Metoda folosita pentru inchidere DataReader este **Close**().

Exemplu – reluat de la Command

```
// sterg articole din combo box
// conexiune mentine informatii pentru a deschide
// o noua conexiune la baza de date
this.combo_Amplasare.Items.Clear();
string str = "Select id_locmunca, denumire From LocMunca” +
      “ order by Denumire";
if (!this.conexiune.IsConnected())
     this.conexiune.Connect();
FbCommand cmd = new FbCommand(str, this.conexiune.Conn);
FbDataReader dr = cmd.ExecuteReader();
try
{
  while (dr.Read())
  {

      this.combo_Amplasare.Items.Add(dr.GetInt16(0).ToString() +
            " | " + dr.GetString(1).Trim());
  }
}
catch(Exception ex)
{
      // cod...
}
finally
{
     dr.Close(); // inchidere DataReader
     cmd.Dispose();
     this.conexiune.Close(); // inchidere conexiune
}
```

Metode

Name	Description
Close	Closes the SqlDataReader object. (Overrides DbDataReader .Close ().)
Dispose ()	Releases all resources used by the current instance of the DbDataReader class. (Inherited from DbDataReader.)
Dispose(Boolean)	Releases the managed resources used by the DbDataReader and optionally releases the unmanaged resources. (Inherited from DbDataReader.)
GetBoolean	Gets the value of the specified column as a Boolean. (Overrides DbDataReader .GetBoolean(Int32) .)
GetByte	Gets the value of the specified column as a byte. (Overrides DbDataReader .GetByte(Int32) .)
GetBytes	Reads a stream of bytes from the specified column offset into the buffer an array starting at the given buffer offset. (Overrides DbDataReader .GetBytes(Int32, Int64, Byte [], Int32, Int32) .)
GetChar	Gets the value of the specified column as a single character. (Overrides DbDataReader .GetChar(Int32) .)
GetDataTypeName	Gets a string representing the data type of the specified column. (Overrides DbDataReader .GetDataTypeName(Int32) .)
GetDateTime	Gets the value of the specified column as a DateTime object. (Overrides DbDataReader .GetDateTime(Int32) .)
GetDecimal	Gets the value of the specified column as a Decimal object. (Overrides DbDataReader .GetDecimal(Int32) .)
GetDouble	Gets the value of the specified column as a double-precision floating point number. (Overrides DbDataReader .GetDouble(Int32) .)
GetEnumerator	Returns an IEnumerator that iterates through the SqlDataReader. (Overrides DbDataReader .GetEnumerator ().)
GetFieldType	Gets the Type that is the data type of the object. (Overrides DbDataReader .GetFieldType(Int32) .)
GetFloat	Gets the value of the specified column as a single-precision floating point number. (Overrides DbDataReader .GetFloat(Int32) .)
GetGuid	Gets the value of the specified column as a globally unique identifier (GUID). (Overrides DbDataReader .GetGuid(Int32) .)
GetInt16	Gets the value of the specified column as a 16-bit signed integer. (Overrides DbDataReader .GetInt16(Int32) .)
GetInt32	Gets the value of the specified column as a 32-bit signed integer. (Overrides DbDataReader .GetInt32(Int32) .)
GetInt64	Gets the value of the specified column as a 64-bit signed integer. (Overrides DbDataReader .GetInt64(Int32) .)
GetName	Gets the name of the specified column. (Overrides DbDataReader .GetName(Int32) .)
GetOrdinal	Gets the column ordinal, given the name of the column. (Overrides DbDataReader .GetOrdinal(String) .)

GetSchemaTable	Returns a DataTable that describes the column metadata of the SqlDataReader. (Overrides DbDataReader .GetSchemaTable ().)
GetSqlBinary	Gets the value of the specified column as a SqlBinary.
GetSqlBoolean	Gets the value of the specified column as a SqlBoolean.
GetSqlByte	Gets the value of the specified column as a SqlByte.
GetSqlBytes	Gets the value of the specified column as SqlBytes.
GetSqlChars	Gets the value of the specified column as SqlChars.
GetSqlDateTime	Gets the value of the specified column as a SqlDateTime.
GetSqlDecimal	Gets the value of the specified column as a SqlDecimal.
GetSqlDouble	Gets the value of the specified column as a SqlDouble.
GetSqlGuid	Gets the value of the specified column as a SqlGuid.
GetSqlInt16	Gets the value of the specified column as a SqlInt16.
GetSqlInt32	Gets the value of the specified column as a SqlInt32.
GetSqlInt64	Gets the value of the specified column as a SqlInt64.
GetSqlMoney	Gets the value of the specified column as a SqlMoney.
GetSqlSingle	Gets the value of the specified column as a SqlSingle.
GetSqlString	Gets the value of the specified column as a SqlString.
IsDBNull	Gets a value that indicates whether the column contains non-existent or missing values. (Overrides DbDataReader .IsDBNull(Int32) .)
Read	Advances the SqlDataReader to the next record. (Overrides DbDataReader .Read ().)

Observatie
Pentru citirea fiecarei coloane ca rezultat al unei cereri Sql se folosesc metodele de tip **Get**...
Metodele ce contin Sql in denumirea lor sunt specifice SQL Server.

Obiectul `Transaction`

Aplicatia creaza un obiect **`Transaction`** (**`SqlTransaction pentru SQL Server`**) apeland metoda **`BeginTransaction`** pe obiectul **`Connection`**.

O atentie deosebita trebuie acordata proprietatii **IsolationLevel**. Nivelul de izolare este specific fiecarui furnizor ADO .NET. Cititi cu atentie documentatia aferenta serverului de baze de date precum si furnizorului ADO .NET.

Ce este tranzactia intr-o baza de date ? ...

Metode des folosite:

Name	Description
`Commit()`	Commits the database transaction. (Overrides DbTransaction .Commit ().)
`Dispose ()`	Releases the unmanaged resources used by the DbTransaction. (Inherited from DbTransaction.)
`Dispose(Boolean)`	Releases the unmanaged resources used by the DbTransaction and optionally releases the managed resources. (Inherited from DbTransaction.)
`Rollback ()`	Rolls back a transaction from a pending state. (Overrides DbTransaction .Rollback ().)
`Rollback(String)`	Rolls back a transaction from a pending state, and specifies the transaction or savepoint name.
`Save()`	Creates a savepoint in the transaction that can be used to roll back a part of the transaction, and specifies the savepoint name.

Proprietati

Name	Description
`Connection`	Gets the SqlConnection object associated with the transaction, or null if the transaction is no longer valid.
`DbConnection`	Specifies the DbConnection object associated with the transaction. (Inherited from DbTransaction.)
`IsolationLevel`	Specifies the IsolationLevel for this transaction. (Overrides DbTransaction .IsolationLevel .)

Pentru **IsolationLevel** (SQL Server) exista urmatoarele posibilitati :

Member name	Description
`Unspecified`	A different isolation level than the one specified is being used, but the level cannot be determined. When using OdbcTransaction, if you do not set IsolationLevel or you set IsolationLevel to Unspecified, the transaction executes according to the

	isolation level that is determined by the driver that is being used.
Chaos	The pending changes from more highly isolated transactions cannot be overwritten.
ReadUncommitted	A dirty read is possible, meaning that no shared locks are issued and no exclusive locks are honored.
ReadCommitted	Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data.
RepeatableRead	Locks are placed on all data that is used in a query, preventing other users from updating the data. Prevents non-repeatable reads but phantom rows are still possible.
Serializable	A range lock is placed on the DataSet, preventing other users from updating or inserting rows into the dataset until the transaction is complete.
Snapshot	Reduces blocking by storing a version of data that one application can read while another is modifying the same data. Indicates that from one transaction you cannot see changes made in other transactions, even if you requery.

Obiectul `DataAdapter`

Obiectul **`DataAdapter`** reprezinta o multime de comenzi si o conexiune la baza de date ce sunt folosite pentru a completa un **`DataSet`** si a actualiza baza de date.

Obiectele **`DataAdapter`** actioneaza ca un "pod" intre baza de date si obiectele deconectate.
Metoda **`Fill.DataAdapter`** va plasa rezultatele cererii intr-un **`DataSet`** sau **`DataTable`**, pasata ca parametru in aceasta metoda.
Modificarile facute intr-un **`DataSet`** pot fi transferate in baza de date prin diverse mecanisme puse la dispozitie de **`DataAdapter`**.

Observatie
DataAdapter are nevoie de conexiune la baza de date in timp ce **DataSet** reprezinta o baza de date in memorie si deci nu are nevoie de conexiune la baza de date.

Vezi colectiile **`TableMappings`**, **`ColumnMappings`**.

Update-ul este facut la nivel de inregistrare. Pentru fiecare inregistrare inserata, modificata sau stearsa, metoda **`Update`** determina tipul de schimbare ce a fost facut pe aceasta (Insert, Update sau Delete).

DataAdapter include proprietatile:

- **`SelectCommand`**,
- **`InsertCommand`**,
- **`DeleteCommand`**,
- **`UpdateCommand`**,
- **`TableMappings`**,

pentru a facilita incarcarea si actualizarea datelor.

`InsertCommand`, **`DeleteCommand`**, si **`UpdateCommand`** sunt template-uri generice ce sunt completate in mod automat cu valori individuale din fiecare rand modificat, folosind mecanismul furnizat de Parameter.

Exemplu completare DataSet

```
private static DataSet SelectRows(DataSet dataset,
    string connectionString,string queryString)
{
    using (SqlConnection connection =
        new SqlConnection(connectionString))
    {
        SqlDataAdapter adapter = new SqlDataAdapter();
        // Comanda ce se executa in cadrul DataAdapter
        // Are nevoie de conexiune la baza de date
        adapter.SelectCommand = new SqlCommand(
            queryString, connection);
        // Completare DataSet
        adapter.Fill(dataset);
        return dataset;
```

```
    }
}
```

In urmatorul exemplu se creaza un **SqlDataAdapter** si se seteaza proprietatile **SelectCommand**, **InsertCommand**, **UpdateCommand** si **DeleteCommand** (MSDN).

```
public static SqlDataAdapter CreateCustomerAdapter(
    SqlConnection connection)
{
    SqlDataAdapter adapter = new SqlDataAdapter();

    // Creare SelectCommand.
    SqlCommand command = new SqlCommand("SELECT * FROM Customers " +
        "WHERE Country = @Country AND City = @City", connection);

    // Parametri pentru SelectCommand.
    command.Parameters.Add("@Country", SqlDbType.NVarChar, 15);
    command.Parameters.Add("@City", SqlDbType.NVarChar, 15);

    // Setare proprietate SelectCommand
    adapter.SelectCommand = command;

    // Creare InsertCommand.
    command = new SqlCommand(
        "INSERT INTO Customers (CustomerID, CompanyName) " +
        "VALUES (@CustomerID, @CompanyName)", connection);

    // Parametri pentru InsertCommand.
    command.Parameters.Add("@CustomerID", SqlDbType.NChar, 5,
                    "CustomerID");
    command.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 40,
                    "CompanyName");

    // Setare proprietate InsertCommand
    adapter.InsertCommand = command;

    // Creare UpdateCommand.
    command = new SqlCommand(
        "UPDATE Customers SET CustomerID = @CustomerID, " +
              "CompanyName = @CompanyName " +
              "WHERE CustomerID = @oldCustomerID", connection);

    // Parametri pentru UpdateCommand.
    command.Parameters.Add("@CustomerID", SqlDbType.NChar, 5,
               "CustomerID");
    command.Parameters.Add("@CompanyName", SqlDbType.NVarChar, 40,
               "CompanyName");
    SqlParameter parameter = command.Parameters.Add(
        "@oldCustomerID", SqlDbType.NChar, 5, "CustomerID");
    parameter.SourceVersion = DataRowVersion.Original;

    // Setare proprietate UpdateCommand
    adapter.UpdateCommand = command;
```

```
    // Creare DeleteCommand.
    command = new SqlCommand(
        "DELETE FROM Customers WHERE CustomerID = @CustomerID",
              connection);

    // Parametri pentru DeleteCommand.
    parameter = command.Parameters.Add(
        "@CustomerID", SqlDbType.NChar, 5, "CustomerID");
    parameter.SourceVersion = DataRowVersion.Original;

    // Setare proprietate DeleteCommand
    adapter.DeleteCommand = command;

    return adapter;
}
```

Constructori SqlDataAdapter

Name	Description
`SqlDataAdapter()`	Initializes a new instance of the SqlDataAdapter class.
`SqlDataAdapter(SqlCommand)`	Initializes a new instance of the SqlDataAdapter class with the specified SqlCommand as the SelectCommand property.
`SqlDataAdapter(String, SqlConnection)`	Initializes a new instance of the SqlDataAdapter class with a SelectCommand and a SqlConnection object.
`SqlDataAdapter(String, String)`	Initializes a new instance of the SqlDataAdapter class with a SelectCommand and a connection string.

Metode DataAdapter

Name	Description
`Fill(DataSet)`	Adds or refreshes rows in the DataSet. (Inherited from DbDataAdapter.)
`Fill(DataTable)`	Adds or refreshes rows in a specified range in the DataSet to match those in the data source using the DataTable name. (Inherited from DbDataAdapter.)
`Fill(DataSet, String)`	Adds or refreshes rows in the DataSet to match those in the data source using the DataSet and DataTable names. (Inherited from DbDataAdapter.)
`Fill(DataTable, IDataReader)`	Adds or refreshes rows in the DataTable to match those in the data source using the DataTable name and the specified IDataReader. (Inherited from DataAdapter.)
`Fill(DataTable, IDbCommand, CommandBehavior)`	Adds or refreshes rows in a DataTable to match those in the data source using the specified DataTable, IDbCommand and CommandBehavior. (Inherited from DbDataAdapter.)
`Update( DataRow [])`	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of DataRow objects. (Inherited from

	DbDataAdapter.)
`Update(DataSet)`	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet. (Inherited from DbDataAdapter.)
`Update(DataTable)`	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataTable. (Inherited from DbDataAdapter.)
`Update( DataRow [], DataTableMapping)`	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified array of DataRow objects. (Inherited from DbDataAdapter.)
`Update(DataSet, String)`	Calls the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the DataSet with the specified DataTable name. (Inherited from DbDataAdapter.)

Obiecte deconectate

Obiectul `DataTable`

Reprezinta o tabela in memorie.

Obiectele ce folosesc **`DataTable`** sunt **`DataSet`** si **`DataView`**.

In situatia cand cream **`DataTable`** in mod programatic, trebuie sa definim schema acesteia adaugand obiecte **`DataColumn`** la colectia **`DataColumnCollection`** (accesate prin proprietatea **`Columns`**).
Pentru a adauga inregistrari (randuri) la **`DataTable`** folosim metoda **`NewRow`** ce returneaza un obiect **`DataRow`**.

`DataTable` contine o colectie de obiecte pentru constrangeri – **`Constraint`** – folosite pentru a asigura integritatea datelor.

Putem determina daca au fost efectuate modificari la o tabela prin folosirea evenimentelor :

- **`RowChanged`**
- **`RowChanging`**
- **`RowDeleted`**
- **`RowDeleting`**

Observatie
Obiectele **`DataSet`** si **`DataTable`** sunt mostenite din **`MarshalByValueComponent`** si suporta interfata **`ISerialize`** pentru remoting. Acestea sunt singurele obiecte ADO .NET ce pot fi folosite in remoting.

Exemplu (MSDN)

```
// Put the next line into the Declarations section.
 private System.Data.DataSet myDataSet;

 private void MakeDataTables()
{
    // Run all of the functions.
    MakeParentTable();
    MakeChildTable();
    MakeDataRelation();
    BindToDataGrid();
 }

 private void MakeParentTable(){
    // Create a new DataTable.
    System.Data.DataTable myDataTable = new DataTable("ParentTable");
    // Declare variables for DataColumn and DataRow objects.
    DataColumn myDataColumn;
    DataRow myDataRow;

    // Create new DataColumn, set DataType, ColumnName
    // and add to DataTable.
    myDataColumn = new DataColumn();
    myDataColumn.DataType = System.Type.GetType("System.Int32");
    myDataColumn.ColumnName = "id";
    myDataColumn.ReadOnly = true;
    myDataColumn.Unique = true;
    // Add the Column to the DataColumnCollection.
    myDataTable.Columns.Add(myDataColumn);

    // Create second column.
    myDataColumn = new DataColumn();
    myDataColumn.DataType = System.Type.GetType("System.String");
    myDataColumn.ColumnName = "ParentItem";
    myDataColumn.AutoIncrement = false;
    myDataColumn.Caption = "ParentItem";
    myDataColumn.ReadOnly = false;
    myDataColumn.Unique = false;
    // Add the column to the table.
    myDataTable.Columns.Add(myDataColumn);

    // Make the ID column the primary key column.
    DataColumn[] PrimaryKeyColumns = new DataColumn[1];
    PrimaryKeyColumns[0] = myDataTable.Columns["id"];
    myDataTable.PrimaryKey = PrimaryKeyColumns;

    // Instantiate the DataSet variable.
    myDataSet = new DataSet();
    // Add the new DataTable to the DataSet.
    myDataSet.Tables.Add(myDataTable);

    // Create three new DataRow objects and add them to the DataTable
    for (int i = 0; i<= 2; i++){
       myDataRow = myDataTable.NewRow();
```

```
        myDataRow["id"] = i;
        myDataRow["ParentItem"] = "ParentItem " + i;
        myDataTable.Rows.Add(myDataRow);
    }
}

private void MakeChildTable(){
    // Create a new DataTable.
    DataTable myDataTable = new DataTable("childTable");
    DataColumn myDataColumn;
    DataRow myDataRow;

    // Create first column and add to the DataTable.
    myDataColumn = new DataColumn();
    myDataColumn.DataType= System.Type.GetType("System.Int32");
    myDataColumn.ColumnName = "ChildID";
    myDataColumn.AutoIncrement = true;
    myDataColumn.Caption = "ID";
    myDataColumn.ReadOnly = true;
    myDataColumn.Unique = true;
    // Add the column to the DataColumnCollection.
    myDataTable.Columns.Add(myDataColumn);

    // Create second column.
    myDataColumn = new DataColumn();
    myDataColumn.DataType= System.Type.GetType("System.String");
    myDataColumn.ColumnName = "ChildItem";
    myDataColumn.AutoIncrement = false;
    myDataColumn.Caption = "ChildItem";
    myDataColumn.ReadOnly = false;
    myDataColumn.Unique = false;
    myDataTable.Columns.Add(myDataColumn);

    // Create third column.
    myDataColumn = new DataColumn();
    myDataColumn.DataType= System.Type.GetType("System.Int32");
    myDataColumn.ColumnName = "ParentID";
    myDataColumn.AutoIncrement = false;
    myDataColumn.Caption = "ParentID";
    myDataColumn.ReadOnly = false;
    myDataColumn.Unique = false;
    myDataTable.Columns.Add(myDataColumn);

    myDataSet.Tables.Add(myDataTable);
    // Create three sets of DataRow objects, five rows each,
    // and add to DataTable.
    for(int i = 0; i <= 4; i ++){
        myDataRow = myDataTable.NewRow();
        myDataRow["childID"] = i;
        myDataRow["ChildItem"] = "Item " + i;
        myDataRow["ParentID"] = 0 ;
        myDataTable.Rows.Add(myDataRow);
    }
    for(int i = 0; i <= 4; i ++){
        myDataRow = myDataTable.NewRow();
        myDataRow["childID"] = i + 5;
```

```
        myDataRow["ChildItem"] = "Item " + i;
        myDataRow["ParentID"] = 1 ;
        myDataTable.Rows.Add(myDataRow);
    }
    for(int i = 0; i <= 4; i ++){
        myDataRow = myDataTable.NewRow();
        myDataRow["childID"] = i + 10;
        myDataRow["ChildItem"] = "Item " + i;
        myDataRow["ParentID"] = 2 ;
        myDataTable.Rows.Add(myDataRow);
    }
 }

 private void MakeDataRelation(){
    // DataRelation requires two DataColumn (parent and child)
    // and a name.
    DataRelation myDataRelation;
    DataColumn parentColumn;
    DataColumn childColumn;
    parentColumn = myDataSet.Tables["ParentTable"].Columns["id"];
    childColumn = myDataSet.Tables["ChildTable"].Columns["ParentID"];
    myDataRelation = new DataRelation("parent2Child", parentColumn,
childColumn);

myDataSet.Tables["ChildTable"].ParentRelations.Add(myDataRelation);
 }

 private void BindToDataGrid(){
    // Instruct the DataGrid to bind to the DataSet, with the
    // ParentTable as the topmost DataTable.
    dataGrid1.SetDataBinding(myDataSet,"ParentTable");
 }

string strSQL = "SELECT CustomerID, CompanyName FROM Customers";
string strConn = "Provider=SQLOLEDB;Data Source=(local);..."
OleDbDataAdapter daCustomers = new OleDbDataAdapter(strSQL, strConn);
DataTable tblCustomers = new DataTable();
daCustomers.Fill(tblCustomers);
```

Observatie
Fiecare **DataTable** are o colectie de **Columns** ce contine obiecte **DataColumn**. Un obiect **DataColumn** reprezinta o coloana in obiectul **DataTable**, dar nu contine date ci numai structura coloanei. **DataColumn** expune proprietatea **Expression**, ce ne permite sa definim campuri calculate.

Consultati MSDN pentru metodele din aceasta clasa.

Obiectul `DataRow`

Pentru a accesa valorile memorate intr-un obiect **`DataTable`** va trebui sa folosim obiecte **`Row`** ce contin o colectie de obiecte **`DataRow`**.

Examinarea datei dintr-o coloana specifica a unei inregistrari se face folosind proprietatea **`Item`** a obiectului **`DataRow`**.

```
string strSQL, strConn;
//...
OleDbDataAdapter da = new OleDbDataAdapter(strSQL, strConn);
DataTable tbl = new DataTable();
da.Fill(tbl);
foreach (DataRow row in tbl.Rows)
      Console.WriteLine(row[0]);
```

`DataSet`

Reprezinta un cache in memorie al datelor incarcate dintr-o sursa de date.

Este sigur la operatiile de citire. Operatiile de scriere trebuiesc sincronizate.

Observatie
`DataSet` consta dintr-o colectie de obiecte **`DataTable`** pe care le putem pune in relatie folosind obiecte **`DataRelation`**.

Putem folosi de asemenea obiecte **`UniqueConstraint`** sau **`ForeignKeyConstraint`**.

Colectia **`DataRelationCollection`** ne permite sa navigam prin ierarhia de tabele. Tabelele sunt continute in **`DataTableCollection`** accesata prin proprietatea **`Tables`**.

Un **`DataSet`** poate citi si scrie date si schema de inregistrare (structura) ca documente XML. Datele si schema pot fi transportate prin retea folosind HTTP si apoi folosite de orice aplicatie ce poate lucra cu XML.
Schema poate fi salvata cu metoda **`WriteXmlSchema`**, iar cu ajutorul metodei **`WriteXml`** putem salva si date si schema.
Pentru a citi un document XML ce include si schema si date vom folosi metoda **`ReadXml`**.

Etapele pentru crearea, reimprospatarea si actualizarea unui **`DataSet`** sunt:

1. Construim si completam fiecare **`DataTable`** din **`DataSet`** cu date dintr-o sursa de date folosind **`DataAdapter`**.
2. Schimbam datele din obiectele individuale **`DataTable`** prin adaugarea, actualizarea sau stergerea de obiecte **`DataRow`**.
3. Invocam metoda **`GetChanges`** pentru a crea un al doilea **`DataSet`** ce pastreaza numai datele modificate.
4. Apelam metoda **`Update`** din **`DataAdapter`**, pasind ca parametru cel de-al doilea **`DataSet`**.
5. Apelam metoda **`Merge`** pentru a salava modificarile din cel de-al doilea DataSet in primul **`DataSet`**.

6. Apelam metoda **`AcceptChanges`** pe primul DataSet sau daca vrem sa renuntam apelam metoda **`RejectChanges`**.

Observatie
Etapa 5 poate sa nu fie facuta. Actualizarea DataSet-ului se face si la apelul metodei **`AcceptChanges`**.

Exemplu (Firebird)

```
// set1 este un DataSet ce constituie sursa de date pentru un datagrid
// Codul urmator actualizeaza o tabela numita Documente cu
// modificarile facute in datagrid.
// set1 este completat folosind un DataAdapter si metoda Fill.
// Etapele 1 si 2 se presupune ca au fost efectuate

// Etapa 3.

DataSet set2 = this.set1.GetChanges();
int modifiedRows = 0;
// obtin conexiune la baza de date
if (!this.conexiune.IsConnected())
    this.conexiune.Connect();

if (set2 != null)
{
    tranzactie = conexiune.Conn.BeginTransaction();
    adapter.UpdateCommand = new FbCommand("UPDATE documente
      SET DENUMIRE=@DENUMIRE," + "TIP=@TIP, NR_COLOANE=@NRCOL,
      ID_FORMULA=@IDF WHERE ID_DOCUMENT=@IDDOC", conexiune.Conn, tranzactie);
    adapter.UpdateCommand.Parameters.Add("@IDDOC", FbDbType.Numeric, 4,
       "ID_DOCUMENT");
    adapter.UpdateCommand.Parameters.Add("@DENUMIRE", FbDbType.Char, 25,
       "DENUMIRE");
    adapter.UpdateCommand.Parameters.Add("@TIP", FbDbType.Char, 1, "TIP");
    adapter.UpdateCommand.Parameters.Add("@NRCOL", FbDbType.Numeric, 2,
       "NR_COLOANE");
    adapter.UpdateCommand.Parameters.Add("@IDF", FbDbType.Char, 6,
       "ID_FORMULA");
    try
    {
      // Etapa 4

      modifiedRows = this.adapter.Update(set2, "documente");
      tranzactie.Commit();

      // Etapa  5. Etapa 6
      this.set1.AcceptChanges();
    }
    catch (FbException w)
    {
      tranzactie.Rollback();
      // jurnalizare eroare
    }
   }
// cod ..
```

Metode de accesare a datelor din DataReader

Metoda inceata

```
string strConn, strSQL;
strConn = "Provider=SQLOLEDB;Data Source=(local)\\NetSDK;" +
"Initial Catalog=Northwind;Trusted_Connection=Yes;";
OleDbConnection cn = new OleDbConnection(strConn);
cn.Open();
strSQL = "SELECT CustomerID, CompanyName FROM Customers";
OleDbCommand cmd = new OleDbCommand(strSQL, cn);
OleDbDataReader rdr = cmd.ExecuteReader();
while (rdr.Read())
     Console.WriteLine(rdr["CustomerID"] + " - " +
     rdr["CompanyName"]);
rdr.Close();
```

Metoda mai rapida

```
...
OleDbDataReader rdr = cmd.ExecuteReader();
int intCustomerIDOrdinal = rdr.GetOrdinal("CustomerID");
int intCompanyNameOrdinal = rdr.GetOrdinal("CompanyName");
while (rdr.Read())
Console.WriteLine(rdr[intCustomerIDOrdinal] + " - " +
rdr[intCompanyNameOrdinal]);
rdr.Close();
```

Metoda si mai rapida

```
...
OleDbDataReader rdr = cmd.ExecuteReader();
int intCustomerIDOrdinal = rdr.GetOrdinal("CustomerID");
int intCompanyNameOrdinal = rdr.GetOrdinal("CompanyName");
while (rdr.Read())
Console.WriteLine(rdr.GetString(intCustomerIDOrdinal) + " - " +
rdr.GetString(intCompanyNameOrdinal));
rdr.Close();
```

DataReader trebuie inchis cat mai repede posibil.

```
string strConn = "Provider=SQLOLEDB;Data Source=(local)\\NetSDK;" +
"Initial Catalog=Northwind;Trusted_Connection=Yes;";
OleDbConnection cn = new OleDbConnection(strConn);
cn.Open();
OleDbCommand cmd = cn.CreateCommand();
cmd.CommandText = "SELECT COUNT(*) FROM Customers";
int intCustomers = Convert.ToInt32(cmd.ExecuteScalar());
cmd.CommandText = "SELECT CompanyName FROM Customers " +
"WHERE CustomerID = 'ALFKI'";
string strCompanyName = Convert.ToString(cmd.ExecuteScalar());
```

Cerere parametrizata

```
string strConn, strSQL;
strConn = "Provider=SQLOLEDB;Data Source=(local)\\NetSDK;" +
"Initial Catalog=Northwind;Trusted_Connection=Yes;";
OleDbConnection cn = new OleDbConnection(strConn);
cn.Open();
strSQL = "SELECT OrderID, CustomerID, EmployeeID, OrderDate " +
"FROM Orders WHERE CustomerID = ?";
OleDbCommand cmd = new OleDbCommand(strSQL, cn);
cmd.Parameters.Add("@CustomerID", OleDbType.WChar, 5);
cmd.Parameters[0].Value = "ALFKI";
OleDbDataReader rdr = cmd.ExecuteReader();
...
OleDbConnection cn = new OleDbConnection(strConn);
cn.Open();
OleDbCommand cmd = cn.CreateCommand();
cmd.CommandText = "GetCustomer";
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add("@CustomerID", OleDbType.WChar, 5);
cmd.Parameters[0].Value = "ALFKI";
OleDbDataReader rdr = cmd.ExecuteReader();
if (rdr.Read())
      Console.WriteLine(rdr["CompanyName"]);
else
      Console.WriteLine("No customer found");
rdr.Close();
cn.Close();
```

Folosirea tranzactiilor

```
cn.Open();
OleDbTransaction txn = cn.BeginTransaction();
string strSQL = "INSERT INTO Customers (...) VALUES (...)";
OleDbCommand cmd = new OleDbCommand(strSQL, cn, txn);
int intRecordsAffected = cmd.ExecuteNonQuery();
if (intRecordsAffected == 1)
{
   Console.WriteLine("Update succeeded");
   txn.Commit();
}
else
{
  //Assume intRecordsAffected = 0
  Console.WriteLine("Update failed");
  txn.Rollback();
}
```

DataAdapter

```
OleDbDataAdapter da;
//Initialize DataAdapter.
DataTableMapping TblMap;
DataColumnMapping ColMap;
TblMap = da.TableMappings.Add("Table", "Employees");
```

```
ColMap = TblMap.ColumnMappings.Add("EmpID", "EmployeeID");
ColMap = TblMap.ColumnMappings.Add("LName", "LastName");
ColMap = TblMap.ColumnMappings.Add("FName", "FirstName");
```