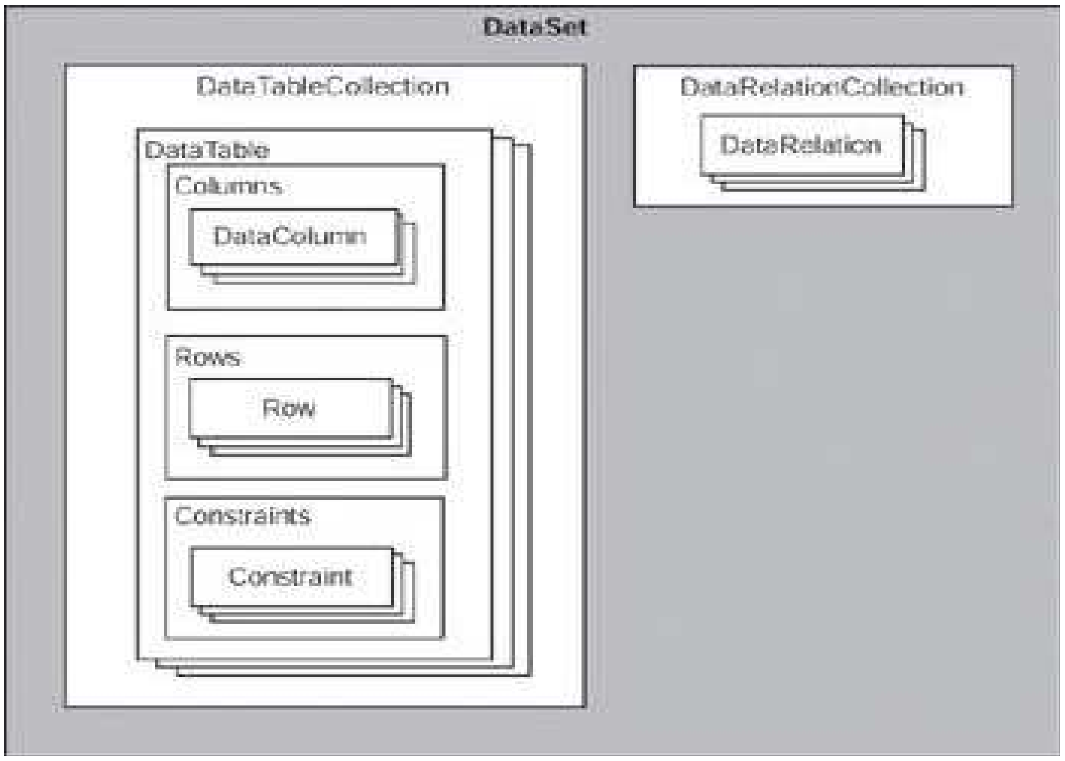
ADO.NET DataSet

The DataSet object is central to supporting disconnected, distributed data scenarios with ADO.NET. The **DataSet** is a memory-resident representation of data that provides a consistent relational programming model regardless of the data source. It can be used with multiple and differing data sources, with XML data, or to manage data local to the application. The **DataSet** represents a complete set of data, including related tables, constraints, and relationships among the tables. The following illustration shows the **DataSet** object model.

- ***Understanding DataSets***

The structure of the DataSet is shown in the following figure.



The DataTableCollection

An ADO.NET **DataSet** contains a collection of zero or more tables represented by DataTable objects. The DataTableCollection contains all the **DataTable** objects in a **DataSet**.

A **DataTable** is defined in the System.Data namespace and represents a single table of memory-resident data. It contains a collection of columns represented by a DataColumnCollection, and constraints represented by a ConstraintCollection, which together define the schema of the table. A **DataTable** also contains a collection of rows represented by the DataRowCollection, which contains the data in the table. Along with its current state, a DataRow retains both its current and original versions to identify changes to the values stored in the row.

The DataRelationCollection

A **DataSet** contains relationships in its DataRelationCollection object. A relationship, represented by the DataRelation object, associates rows in one **DataTable** with rows in another **DataTable**. A relationship is analogous to a join path that might exist between primary and foreign key columns in a relational database. A **DataRelation** identifies matching columns in two tables of a **DataSet**.

Relationships enable navigation from one table to another within a **DataSet**. The essential elements of a **DataRelation** are the name of the relationship, the name of the tables being related, and the related columns in each table. Relationships can be built with more than one column per table by specifying an array of DataColumn objects as the key columns. When you add a relationship to the **DataRelationCollection**, you can optionally add a **UniqueKeyConstraint** and a **ForeignKeyConstraint** to enforce integrity constraints when changes are made to related column values.

- ***DataSet Properties***

The properties exposed by the DataSet object are listed below:

Properties	**Description**
CaseSensitive	Determines whether comparisons are case sensitive
DataSetName	The name used to reference the DataSet in code
DefaultViewManager	Defines the default filtering and sorting order of the DataSet
EnforceConstraints	Determines whether constraint rules are followed during changes
ExtendedProperties	Custom user information
HasErrors	Indicates whether any of the DataRows in the

	DataSet contain errors
Locale	The locale information to be used when comparing strings
Namespace	The namespace used when reading or writing an XML document
Prefix	An XML prefix used as an alias for the namespace
Relations	A collection of DataRelati on objects that define the relationship of the DataTables within the DataSet
Tables	The collection of DataTables contained in the DataSet

- ***DataSet Methods***

Table 6-3: Primary DataSet Methods

Method	**Description**
AcceptChanges	Commits all pending changes to the DataSet
Clear	Empties all the tables in the DataSet
Clone	Copies the structure of a DataSet
Copy	Copies the structure and contents of a DataSet
GetChanges	Returns a DataSet containing only the changed rows in each of its tables
GetXml	Returns an XML representati on of the DataSet
GetXmlSchema	Returns an XSD representati on of the DataSet’s schema
HasChanges	Returns a Boolean value indicating whether the DataSet has pending Changes
InferXmlSchema	Infers a schema from an XML TextReader or file
Merge	Combines two DataSets
ReadXml	Reads an XML schema and data into the DataSet
ReadXmlSchema	Reads an XML schema into the DataSet
RejectChanges	Rolls back all changes pending in the DataSet
Reset	Returns the DataSet to its original state
WriteXml	Writes an XML schema and data from the DataSet
WriteXmlSchema	Writes the DataSet structure as an XML schema

- ***Read Data using DataSet:***

```
using System.Text;
using System.Windows.Forms;

namespace ReadData_Using_DataSet
{
    public partial class Form1 : Form
    {
        System.Data.OleDb.OleDbConnection cn;
        System.Data.OleDb.OleDbDataAdapter da;
        System.Data.DataSet ds;
        System.Data.DataTable dt;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
          cn = new
          System.Data.OleDb.OleDbConnection("Provider=Micro
          soft.Jet.OLEDB.4.0;Data Source=D:\\Student.mdb");
            cn.Open();
          da = new
          System.Data.OleDb.OleDbDataAdapter("select * from
          stu", cn);
            ds = new DataSet();
            da.Fill(ds, "stu");
            dt = ds.Tables["stu"];
            for (int i = 0; i < dt.Rows.Count-1; i++)
            {

                    comboBox1.Items.Add(dt.Rows[i]["name"].
                    ToString());
            }
            cn.Close();
        }
    }
}
```

- ***Manipulate Data using DataSet:***

```
using System;
using System.Windows.Forms;

namespace DataManipulation_Using_DataSet1
{
    public partial class Form1 : Form
    {
        System.Data.OleDb.OleDbConnection cn;
        System.Data.OleDb.OleDbDataAdapter da;
        System.Data.DataSet ds;
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
          cn = new
          System.Data.OleDb.OleDbConnection("Provider=Micro
          soft.Jet.OLEDB.4.0;Data Source=D:\\Student.mdb");
        }

        private void btnInsert_Click(object sender,
        EventArgs e)
        {
               cn.Open();
               da = new
               System.Data.OleDb.OleDbDataAdapter("select *
               from stu", cn);
               ds = new DataSet();
               da.Fill(ds, "stu");
               da.InsertCommand = new
               System.Data.OleDb.OleDbCommand("insert into
               stu values(" + txtRoll.Text + ",'" +
               txtName.Text + "')", cn);
               da.InsertCommand.ExecuteNonQuery();
               MessageBox.Show("Inserted");
               txtName.Text = "";
               txtRoll.Text = "";
               cn.Close();
        }

        private void btnUpdate_Click(object sender,
        EventArgs e)
        {
            cn.Open();
               da = new
               System.Data.OleDb.OleDbDataAdapter("select *
               from stu", cn);
            ds = new DataSet();
            da.Fill(ds, "stu");
               da.UpdateCommand  = new
               System.Data.OleDb.OleDbCommand("update stu
               set name='"+ txtName.Text + "' where roll="
               + txtRoll.Text + "", cn);
            da.UpdateCommand.ExecuteNonQuery();
            MessageBox.Show("Records have been update!");
            txtName.Text = "";
            txtRoll.Text = "";
            cn.Close();

        }

        private void btnDelete_Click(object sender,
        EventArgs e)
        {
            cn.Open();
               da = new
               System.Data.OleDb.OleDbDataAdapter("select *
               from stu", cn);
            ds = new DataSet();
            da.Fill(ds, "stu");
               da.DeleteCommand = new
               System.Data.OleDb.OleDbCommand("delete from
               stu where roll=" + txtRoll.Text + "", cn);
            da.DeleteCommand.ExecuteNonQuery();
            MessageBox.Show("Deleted");
            txtName.Text = "";
            txtRoll.Text = "";
            cn.Close();
        }


    }
}
```