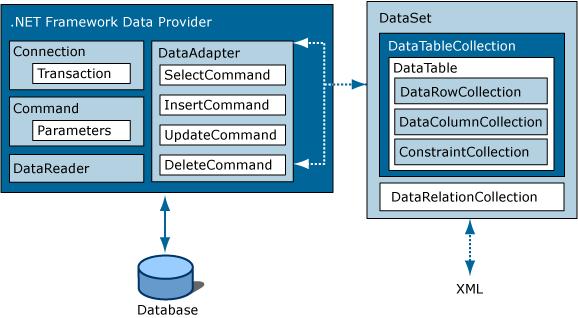
# ADO.NET

ADO.NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.

ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, placed in an ADO.NET DataSet object in order to be exposed to the user in an ad hoc manner, combined with data from multiple sources, or passed between tiers.

## Architecture

The Connection object provides connectivity to a data source. The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information. The DataReader provides a high-performance stream of data from the data source. Finally, the DataAdapter provides the bridge between the DataSet object and the data source. The DataAdapter uses Command objects to execute SQL commands at the data source to both load the DataSet with data and reconcile changes that were made to the data in the DataSet back to the data source.



DataSet can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application.

### DataSet Vs DataReader

Use a DataSet to do the following:

* Cache data locally in your application so that you can manipulate it. If you only need to read the results of a query, the DataReader is the better choice.
* Remote data between tiers or from an XML Web service.
* Interact with data dynamically such as binding to a Windows Forms control or combining and relating data from multiple sources.
* Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.

If you do not require the functionality provided by the DataSet, you can improve the performance of your application by using the DataReader to return your data in a forward-only, read-only manner

### DataSet and XML

DataSet can be populated with data from an XML source, whether it is a file or an XML stream. The DataSet can be written as World-Wide Web Consortium (W3C) compliant XML that includes its schema as XML schema definition language (XSD) schema, regardless of the source of the data in the DataSet. Because of the native serialization format of the DataSet is XML, it is an excellent medium for moving data between tiers, making the DataSet an optimal choice for remoting data and schema context to and from an XML Web service.

### LINQ to SQL

LINQ to SQL supports queries against an object model that is mapped to the data structures of a relational database without using an intermediate conceptual model.

## .NET Framework Data Providers

The following table lists the data providers that are included in the .NET Framework.

|  |  |
| --- | --- |
| .NET Framework data provider | Description |
| .NET Framework Data Provider for SQL Server | Provides data access for Microsoft SQL Server version 7.0 or later versions. Uses the System.Data.SqlClient namespace. |
| .NET Framework Data Provider for OLE DB | For data sources exposed by using OLE DB. Uses the System.Data.OleDb namespace. |
| .NET Framework Data Provider for ODBC | For data sources exposed by using ODBC. Uses the System.Data.Odbc namespace. |
| .NET Framework Data Provider for Oracle | For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the System.Data.OracleClient namespace. |

### Core Objects of .NET Framework Data Providers

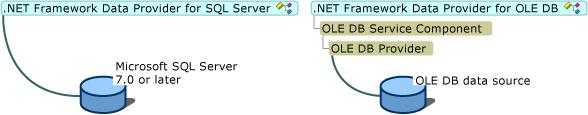
The following table outlines the four core objects that make up a .NET Framework data provider.

|  |  |
| --- | --- |
| Object | Description |
| Connection | Establishes a connection to a specific data source. The base class for all Connection objects is the DbConnection class. |
| Command | Executes a command against a data source. Exposes Parameters and can execute in the scope of a Transaction from a Connection. The base class for all Command objects is the DbCommand class. |
| DataReader | Reads a forward-only, read-only stream of data from a data source. The base class for all DataReader objects is the DbDataReader class. |
| DataAdapter | Populates a DataSet and resolves updates with the data source. The base class for all DataAdapter objects is the DbDataAdapter class. |

In addition to the core classes listed in the table earlier in this document, a .NET Framework data provider also contains the classes listed in the following table.

|  |  |
| --- | --- |
| Object | Description |
| Transaction | Enlists commands in transactions at the data source. The base class for all Transaction objects is the DbTransaction class. ADO.NET also provides support for transactions using classes in the System.Transactions namespace. |
| CommandBuilder | A helper object that automatically generates command properties of a DataAdapter or derives parameter information from a stored procedure and populates the Parameters collection of a Command object. The base class for all CommandBuilder objects is the DbCommandBuilder class. |
| ConnectionStringBuilder | A helper object that provides a simple way to create and manage the contents of connection strings used by the Connection objects. The base class for all ConnectionStringBuilder objects is the DbConnectionStringBuilder class. |
| Parameter | Defines input, output, and return value parameters for commands and stored procedures. The base class for all Parameter objects is the DbParameter class. |
| Exception | Returned when an error is encountered at the data source. For an error encountered at the client, .NET Framework data providers throw a .NET Framework exception. The base class for all Exception objects is the DbException class. |
| Error | Exposes the information from a warning or error returned by a data source. |
| ClientPermission | Provided for .NET Framework data provider code access security attributes. The base class for all ClientPermission objects is the DBDataPermission class. |

ADO.NET is optimized for SQLServer7.0 or later with direct access.



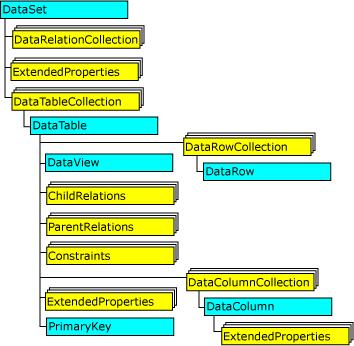
### Choosing a .NET Framework Data Provider

Depending on the design and data source for your application, your choice of .NET Framework data provider can improve the performance, capability, and integrity of your application. The following table discusses the advantages and limitations of each .NET Framework data provider.

|  |  |
| --- | --- |
| Provider | Notes |
| .NET Framework Data Provider for SQL Server | Recommended for middle-tier applications that use Microsoft SQL Server 7.0 or a later version.  Recommended for single-tier applications that use Microsoft Database Engine (MSDE) or SQL Server 7.0 or a later version.  Recommended over use of the OLE DB provider for SQL Server (SQLOLEDB) with the .NET Framework Data Provider for OLE DB.  For SQL Server 6.5 and earlier, you must use the OLE DB provider for SQL Server with the .NET Framework Data Provider for OLE DB. |
| .NET Framework Data Provider for OLE DB | Recommended for middle-tier applications that use SQL Server 6.5 or earlier.  For SQL Server 7.0 or a later version, the .NET Framework Data Provider for SQL Server is recommended.  Also recommended for single-tier applications that use Microsoft Access databases. Use of an Access database for a middle-tier application is not recommended. |
| .NET Framework Data Provider for ODBC | Recommended for middle and single-tier applications that use ODBC data sources. |
| .NET Framework Data Provider for Oracle | Recommended for middle and single-tier applications that use Oracle data sources. |

## DataSets

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.



## Side-by-Side Execution

## Working with DataSets

There are several ways of working with a DataSet:

* Programmatically create a DataTable, DataRelation, and Constraint within a DataSet and populate the tables with data.
* Populate the DataSet with tables of data from an existing relational data source using a DataAdapter.
* Load and persist the DataSet contents using XML

Adding a **DataRelation** to a DataSet adds, by default, a UniqueConstraint to the parent table and a ForeignKeyConstraint to the child table.

DataRow.GetParenetRow and DataRow.GetChildRows for navigating data.

DataSet.Merge for merging data sets.

When copying a **DataSet**, you can:

* Create an exact copy of the **DataSet**, including the schema, data, row state information, and row versions.
* Create a **DataSet** that contains the schema of an existing **DataSet**, but only rows that have been modified. You can return all rows that have been modified, or specify a specific **DataRowState**
* Copy the schema, or relational structure, of the **DataSet** only, without copying any rows. Rows can be imported into an existing DataTable using ImportRow .

DataSet.Copy, DataSet.GetChanges, Table.ImportRow

**MergeFailed** event, **Initialized** event and **Dispose** event.

The DataSet and DataTable objects are the only ADO.NET objects that can be remoted.

The DataSet provides access to data through a strongly typed metaphor. Tables and columns that are part of the **DataSet** can be accessed using user-friendly names and strongly typed variables.

A typed **DataSet** is a class that derives from a **DataSet.** Typed **DataSet** provides strongly typed methods, events, and properties.

xsd.exe /d /l:CS XSDSchemaFileName.xsd /eld /n:XSDSchema.Namespace. We can generate strong typed dataset from xml schema directly.

Using annotations, you can customize the names of objects in your typed **DataSet** with more meaningful names, making code more readable and your typed **DataSet** easier for clients to use, while leaving underlying schema intact (that is important for data changes).

### Working with Data Table

A **DataTable** represents one table of in-memory relational data; the data is local to the .NET-based application in which it resides, but can be populated from a data source such as Microsoft SQL Server using a **DataAdapter**.

You define the schema of a **DataTable** using DataColumn objects as well as ForeignKeyConstraint and UniqueConstraint objects. The columns in a table can map to columns in a data source, **contain calculated values from expressions, automatically increment their values**, or contain primary key values.

You can define an **expression** for a column, enabling it to contain a value calculated from other column values in the same row or from the column values of multiple rows in the table, which will be quite **useful an powerful**

|  |  |
| --- | --- |
| Expression type | Example |
| Comparison | "Total >= 500" |
| Computation | "UnitPrice \* Quantity" |
| Aggregation | Sum(Price) |

Set the AutoIncrement property of the column to **true** to enable a column's auto  incrementation.

You can use constraints to enforce restrictions on the data in a DataTable, in order to maintain the integrity of the data. A constraint is an automatic rule, applied to a column or related columns.Constraints are enforced when the EnforceConstraints property of the DataSet is **true.**

There are two kinds of constraints in ADO.NET: the ForeignKeyConstraint and the UniqueConstraint.

Settings for **DeleteRule** and **UpdateRule** properties of the **ForeignKeyConstraint**.

|  |  |
| --- | --- |
| Rule setting | Description |
| **Cascade** | Delete or update related rows. |
| **SetNull** | Set values in related rows to **DBNull**. |
| **SetDefault** | Set values in related rows to the default value. |
| **None** | Take no action on related rows. This is the default |

Settings for the **AcceptRejectRule**.of the **ForeignKeyConstraint**.

|  |  |
| --- | --- |
| Rule setting | Description |
| **Cascade** | Accept or reject changes to child rows. |
| **None** | Take no action on child rows. This is the default. |

The **UniqueConstraint** object ensures that all data in the specified column or columns is unique per row.

The **Select** method of the **DataTable** object returns a set of DataRow objects that match the specified criteria and determines which version of the rows to view or manipulate based on a DataViewRowState.

|  |  |
| --- | --- |
| DataViewRowState value | Description |
| **CurrentRows** | Current rows including unchanged, added, and modified rows. |
| **Deleted** | A deleted row. |
| **ModifiedCurrent** | A current version, which is a modified version of original data. (See **ModifiedOriginal**.) |
| **ModifiedOriginal** | The original version of all modified rows. The current version is available using **ModifiedCurrent**. |
| **Added** | A new row. |
| **None** | None. |
| **OriginalRows** | Original rows, including unchanged and deleted rows. |
| **Unchanged** | An unchanged row. |

You can use the Load method to load a DataTable with rows from a data source.

|  |  |
| --- | --- |
| LoadOption value | Description |
| **OverwriteRow** | If incoming rows have the same **PrimaryKey** value as a row already in the **DataTable**, the **Original** and **Current** values of each column are replaced with the values in the incoming row, and the **RowState** property is set to **Unchanged**. |
| **PreserveCurrentValues (default)** | If incoming rows have the same **PrimaryKey** value as a row already in the **DataTable**, the **Original** value is set to the contents of the incoming row, and the **Current** value is not changed. |
| **UpdateCurrentValues** | If incoming rows have the same **PrimaryKey** value as the row already in the **DataTable**, the **Current** value is copied to the **Original** value, and the **Current** value is then set to the contents of the incoming row. |

When you modify column values in a **DataRow** directly, the **DataRow** manages the column values using the **Current**, **Default**, and **Original** row versions. In addition to these row versions, the **BeginEdit**, **EndEdit**, and **CancelEdit** methods use a fourth row version: **Proposed**.

During the edit operation, you can apply validation logic to individual columns by evaluating the **ProposedValue** in the **ColumnChanged** event of the **DataTable**

ADO.NET manages rows in tables using **row states** and **versions**. A row state indicates the status of a row; row versions maintain the values stored in a row as it is modified, including current, original, and default values.

|  |  |
| --- | --- |
| RowState value | Description |
| Unchanged | No changes have been made since the last call to AcceptChanges or since the row was created by DataAdapter.Fill. |
| Added | The row has been added to the table, but AcceptChanges has not been called. |
| Modified | Some element of the row has been changed. |
| Deleted | The row has been deleted from a table, and AcceptChanges has not been called. |
| Detached | The row is not part of any DataRowCollection. |

|  |  |
| --- | --- |
| DataRowVersion value | Description |
| Current | The current values for the row. This row version does not exist for rows with a RowState of Deleted. |
| Default | The default row version for a particular row. The default row version for an Added, Modified, or Unchanged row is Current. The default row version for a Deleted row is Original. The default row version for a Detached row is Proposed. |
| Original | The original values for the row. This row version does not exist for rows with a RowState of Added. |
| Proposed | The proposed values for the row. This row version exists during an edit operation on a row, or for a row that is not part of a DataRowCollection. |

Table.Rows.Remove() and TableRow.Delete()

The DataRow object provides a RowError property on each row for this purpose. Adding data to the **RowError** property of a **DataRow** sets the HasErrors property of the **DataRow** to **true.**

After verifying the accuracy of changes made to data in a DataTable, you can accept the changes using the AcceptChanges method of the DataRow, DataTable, or DataSet, which will set the **Current** row values to be the **Original** values and will set the **RowState** property to **Unchanged**.

The DataTable object provides a series of events that can be processed by an application.

When you create a **DataTableReader** from a **DataTable**, the resulting **DataTableReader** object contains one result set with the same data as the **DataTable** from which it was created, except for any rows that have been marked as deleted. The columns appear in the same order as in the original **DataTable**

The DataTable and DataSet classes have a CreateDataReader method.

A DataTableReader may contain multiple result sets. When there is more than one result set, the NextResult method advances the cursor to the next result set.

### Work with Data View

A DataView enables you to create different views of the data stored in a DataTable.

A **DataView** provides a dynamic view of data in the underlying **DataTable**: the content, ordering, and membership reflect changes as they occur. This behavior differs from the **Select** method of the **DataTable**, which returns a DataRow array from a table based on a particular filter and/or sort order: this content reflects changes to the underlying table, but its membership and ordering remain static.

a **DataView** cannot be treated as a table and cannot provide a view of joined tables. You also cannot exclude columns that exist in the source table, nor can you append columns, such as computational columns, that do not exist in the source table.

The **DataViewManager** provides you with a convenient way to manage default view settings for each table. When binding a control to more than one table of a **DataSet**, binding to a **DataViewManager** is the ideal choice.

You can use the **DataView** constructor, or you can create a reference to the DefaultView property of the DataTable. You can also set **Sort**, **RowFilter**, or **RowStateFilter** properties later but will result in re-indexing.

If you want to return the results of a particular query on the data, as opposed to providing a dynamic view of a subset of the data, use the Find or FindRows methods of the **DataView** to achieve best performance rather than setting the **RowFilter** property.

You can use the RowStateFilter property to specify which row versions to view.

You can search for rows by using the Find and FindRows methods of the DataView. The **Find** and **FindRows** methods take an array of values as input whose length matches the number of columns in the sort order.

You can create a DataView containing rows from the related child table by using the CreateChildView method of the DataRowView for the rows in the parent table.

The ability to use the **DataView** to modify data in the underlying table is controlled by setting one of three Boolean properties of the **DataView**. These properties are AllowNew, AllowEdit, and AllowDelete. They are set to **true** by default.

You can use the ListChanged event of the DataView to determine if a view has been updated.

You can use a DataViewManager to manage view settings for all the tables in a DataSet. If you have a control that you want to bind to multiple tables, such as a grid that navigates relationships, a **DataViewManager** is ideal.

### Using XML in a DataSet (ADO.NET)

You can use the XML stream or document to supply to the DataSet either data, schema information, or both.

ADO.NET also allows you to create an XML representation of a DataSet, with or without its schema, in order to transport the DataSet across HTTP for use by another application or XML-enabled platform.

A DiffGram is an XML format that identifies current and original versions of data elements. When a DataSet is written as a DiffGram, it populates the DiffGram with all the necessary information to accurately recreate the contents, though not the schema, of the DataSet, including column values from both the **Original** and **Current** row versions, row error information, and row order.

DiffGram can also be used for loading/saving DataSet, as one of the possible formats.

The **ReadXml** method reads the contents of the XML stream or document and loads the DataSet with data. It will also create the relational schema of the DataSet depending on the **XmlReadMode** specified and whether or not a relational schema already exists.

ADO.NET you can write an XML representation of a DataSet, with or without its schema. To obtain the XML representation of the DataSet as a string, use the **GetXml** method

To load **DataSet** schema information from an XML document, you can use either the **ReadXmlSchema** or the **InferXmlSchema** method of the **DataSet**. **ReadXmlSchema** allows you to load or infer **DataSet** schema information from the document containing XML Schema definition language (XSD) schema, or an XML document with inline XML Schema.

You can write the schema of a DataSet as XML Schema definition language (XSD) schema, so that you can transport it, with or without related data, in an XML document.

the .NET Framework enables real-time, synchronous access to both the relational and hierarchical representations of data through the **DataSet** object and the XmlDataDocument object, respectively.When a **DataSet** is synchronized with an **XmlDataDocument**, both objects are working with a single set of data.

There are several ways that you can synchronize a **DataSet** with an **XmlDataDocument**. You can:

* Populate a **DataSet** with schema (that is, a relational structure) and data and then synchronize it with a new **XmlDataDocument**
* Populate a **DataSet** with schema only (such as a strongly typed **DataSet**), synchronize it with an **XmlDataDocument**, and then load the **XmlDataDocument** from an XML document.
* Create a new **XmlDataDocument** and load it from an XML document, and then access the relational view of the data using the **DataSet** property of the **XmlDataDocument**

Synchronizing an **XmlDataDocument** with a **DataSet** allows the formatting and hierarchical element structure of the original XML document to be maintained in the **XmlDataDocument**, while the **DataSet** contains only data and schema information appropriate to the **DataSet**.

the **DataRelation** exposes a **Nested** property. Setting the **Nested** property of a **DataRelation** to **true** causes the child rows of the relation to be nested within the parent column when written as XML data or synchronized with an **XmlDataDocument.**

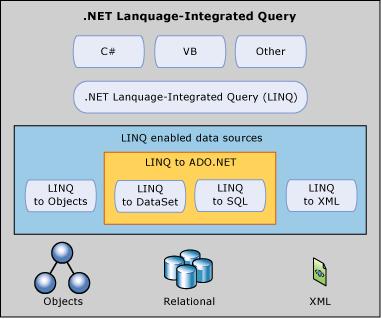
This section provides an overview of how the relational schema of a DataSet is built from an XML Schema definition language (XSD) schema document. In general, for each **complexType** child ]element of a schema element, a table is generated in the **DataSet**.

If no XML Schema or XDR schema is available in the XML, the schema of the DataSet can be inferred from the structure of the XML elements and attributes.

In WebService, the **DataSet** is implicitly converted to an XML stream using the DiffGram format, sent over the network, and then reconstructed from the XML stream as a **DataSet** on the ]receiving end.

## LINQ to ADO.NET

LINQ query operators enable you to project, filter, and traverse in-memory collections or tables in a database. The LINQ queries are expressed in the programming language itself, and not as string literals embedded in the application code. It simplifies querying by eliminating the need to use a separate query language. And if you use the Visual Studio 2008 IDE, LINQ also lets you take advantage of compile-time checking, static typing, and IntelliSense.



You can write LINQ queries against various enumerable data sources (that is, a data source that implements the IEnumerable interface), such as in-memory data structures, XML documents, SQL databases, and DataSet objects.ntegrating queries into the programming language also enables Visual Studio programmers to be more productive by providing compile-time type and syntax checking, and IntelliSense.

LINQ to ADO.NET consists of two separate technologies: LINQ to DataSet and LINQ to SQL. LINQ to DataSet provides richer, optimized querying over the DataSet and LINQ to SQL enables you to directly query SQL Server database schemas.

### LINQ to DataSet

The LINQ to DataSet functionality is exposed primarily through the extension methods in the DataRowExtensions and DataTableExtensions classes. LINQ to DataSet builds on and uses the existing ADO.NET 2.0 architecture, and is not meant to replace ADO.NET 2.0 in application code

Formulating queries using LINQ to DataSet is similar to using Language-Integrated Query (LINQ) against other LINQ-enabled data sources. LINQ queries can be performed against single tables in a DataSet or against more than one table by using the Join and GroupJoin standard query operators.

LINQ to DataSet adds several DataSet-specific extensions that make it easier to query over a set of DataRow objects. These DataSet-specific extensions include operators for comparing sequences of rows, as well as methods that provide access to the column values of a DataRow.

#### Queries

LINQ to DataSet queries can be formulated in two different syntaxes: query expression syntax and method-based query syntax.

**Query Expression Syntax**

DataTable products = ds.Tables["Product"];

IEnumerable<DataRow> query = from product in products.AsEnumerable() select product; Console.WriteLine("Product Names:");

foreach (DataRow p in query)

{ Console.WriteLine(p.Field<string>("Name")); }

**Method-Based Query Syntax**

DataTable products = ds.Tables["Product"];

var query = products.AsEnumerable().

    Select(product => new

                {

                    ProductName = product.Field<string>("Name"),

                    ProductNumber = product.Field<string>("ProductNumber"),

                    Price = product.Field<decimal>("ListPrice")

                });

Console.WriteLine("Product Info:");

foreach (var productInfo in query)

{

    Console.WriteLine("Product name: {0} Product number: {1} List price: ${2} ",

        productInfo.ProductName, productInfo.ProductNumber, productInfo.Price);

}

**Composing Queries**

IEnumerable<DataRow> productsQuery =

    from product in products.AsEnumerable()

    select product;

IEnumerable<DataRow> largeProducts =

    productsQuery.Where(p => p.Field<string>("Size") == "L");

LINQ is deplayed executed. After a query has been executed, no additional queries can be composed, and all subsequent queries will use the in-memory LINQ operators.

when you use LINQ queries over a DataSet object you are querying an enumeration of DataRow objects, instead of an enumeration of a custom type. This means that you can use any of the members of the DataRow class in your LINQ queries.

**-Single-Table Queries**

var query =

    from order in orders.AsEnumerable()

    where order.Field<bool>("OnlineOrderFlag") == true

    select new

           {

               SalesOrderID = order.Field<int>("SalesOrderID"),

               OrderDate = order.Field<DateTime>("OrderDate"),

               SalesOrderNumber = order.Field<string>("SalesOrderNumber")

           };

The Field method provides access to the column values of a DataRow and the SetField (not shown in the previous example) sets column values in a DataRow

**Cross-Table Queries**

In addition to querying a single table, you can also perform cross-table queries in LINQ to DataSet. This is done by using a join.

The Language-Integrated Query (LINQ) framework provides two join operators, Join and GroupJoin.

var query =

    from order in orders.AsEnumerable()

    join detail in details.AsEnumerable()

    on order.Field<int>("SalesOrderID") equals

        detail.Field<int>("SalesOrderID")

    where order.Field<bool>("OnlineOrderFlag") == true

    && order.Field<DateTime>("OrderDate").Month == 8

    select new

           {

               SalesOrderID =

                   order.Field<int>("SalesOrderID"),

               SalesOrderDetailID =

                   detail.Field<int>("SalesOrderDetailID"),

               OrderDate =

                   order.Field<DateTime>("OrderDate"),

               ProductID ======---

                   detail.Field<int>("ProductID")

           };

GroupJoin Sample:

 var q1 = from c in customers

             select new { CustomerName = c.Name,

                          Orders = from o in orders

                                   where c.Key == o.Key

                                   select o };

**Querying Typed DataSets**

With a typed DataSet, you do not have to use the generic Field method or SetField method to access column data.

var query = from o in orders

            where o.OnlineOrderFlag == true

            select new { o.SalesOrderID,

                         o.OrderDate,

                         o.SalesOrderNumber };

**Comparing DataRows**

==========

Language-Integrated Query (LINQ) defines various set operators to compare source elements to see if they are equal. LINQ provides the following set operators:

* Distinct, Returns distinct elements from a sequence.
* Union, Produces the set union of two sequences
* Intersect， Produces the set intersection of two sequences
* Except, Produces the set difference of two sequences

// Find the intersection of the two tables. var contacts = contacts1.AsEnumerable().Intersect(contacts2.AsEnumerable(), DataRowComparer.Default);

The CopyToDataTable method takes the results of a query and copies the data into a DataTable, which can then be used for data binding. -

// Bind the System.Windows.Forms.DataGridView object

// to the System.Windows.Forms.BindingSource object.

dataGridView.DataSource = bindingSource;

......

IEnumerable<DataRow> query =

    from order in orders.AsEnumerable()

    where order.Field<DateTime>("OrderDate") > new DateTime(2001, 8, 1)

    select order;

// Create a table from the query.

DataTable boundTable = query.CopyToDataTable<->();

// Bind the table to a System.Windows.Forms.BindingSource object,

// which acts as a proxy for a System.Windows.Forms.DataGridView object.

bindingSource.DataSource = boundTable;

**Creating a DataView Object**

Note the difference between CopyToDataTable and AsDataView.

EnumerableRowCollection<DataRow> query =

    from order in orders.AsEnumerable()

    where order.Field<bool>("O======nlineOrderFlag") == true

    orderby order.Field<decimal>("TotalDue")

    select order;

DataView view = query.AsDataView();

bindingSource1.DataSource = view;

This DataView uses the filtering and sorting specified in the query, but is better suited for data binding.

DataView constructs an index. An index contains keys built from one or more columns in the table or view. These keys are stored in a structure that enables the DataView to find the row or rows associated with the key values quickly and efficiently.

When debugging, Edit and Continue does not support changes to LINQ to DataSet queries.

LINQ to DataSet query can be formulated in one point of a program and executed in a different one. At the point where the query is formulated, the query can reference any element that is visible at that point; At execution time, the query will effectively be able to access those members that were referenced by the query at formulation, even if the calling code does not have visibility into them.

In general, LINQ to DataSet queries should not be passed to untrusted components, unless the query has been carefully constructed so that it does not expose information that should be kept private.

 LINQ to DataSet queries are not composed by using string manipulation or concatenation, as they are in Transact-SQL, and are not susceptible to SQL injection attacks in the traditional sense.

### LINQ to SQL

In LINQ to SQL, the data model of a relational database is mapped to an object model expressed in the programming language of the developer. When the application runs, LINQ to SQL translates into SQL the language-integrated queries in the object model and sends them to the database for execution. When the database returns the results, LINQ to SQL translates them back to objects that you can work with in your own programming language.

By using LINQ to SQL, you can use the LINQ technology to access SQL databases just as you would access an in-memory collection.

// Northwnd inherits from System.Data.Linq.DataContext.

Northwnd nw = new Northwnd(@"northwnd.mdf");

var companyNameQuery =

    from cust in nw.Customers

    where cust.City == "London"

    select cust.CompanyName;

Two tools are available for automatically generating a Visual Basic or C# model from the metadata of an existing database.

* Object Relational Designer
* The SQLMetal command line

sqlmetal /code:northwind.cs /language:csharp "c:\northwnd.mdf" /sprocs /functions /pluralize

you can generate your object model as an external XML file by using the SQLMetal command-line tool.

The DataContext is the main conduit by which you connect to a database, retrieve objects from it, and submit changes back to it. The purpose of the DataContext is to translate your requests for objects into SQL queries to be made against the database, and then to assemble objects out of the results.

// DataContext takes a connection string.

DataContext db = new DataContext(@"c:\Northwnd.mdf");

// Get a typed table to run queries.

Table<Customer> Customers = db.GetTable<Customer>();

Best practice is to declare a strongly typed DataContext instead of relying on the basic DataContext class and the GetTable method.

Northwnd db = new Northwnd(@"c:\Northwnd.mdf");

var query =

    from cust in db.Customers

    where cust.City == "London"

    select cust;

Assuming a DataContext connection, you can use ExecuteCommand to execute SQL commands that do not return objects.

db.ExecuteCommand("UPDATE Products SET UnitPrice = UnitPrice + 1.00");

When you do not intend to change the data, you can increase the performance of queries by seeking read-only results.

db.ObjectTrackingEnabled = false;

IOrderedQueryable<Employee> hireQuery =

    from emp in db.Employees

    orderby emp.HireDate

    select emp;

Use the LoadWith method to specify which data related to your main target should be retrieved at the same time. For example, if you know you will need information about customers' orders, you can use LoadWith to make sure that the order information is retrieved at the same time as the customer information

You can retrieve many objects in one query by using LoadWith.

DataLoadOptions dlo = new DataLoadOptions();

dlo.LoadWith<Customer>(c => c.Orders);

db.LoadOptions = dlo;

Use the AssociateWith method to specify sub-queries to limit the amount of retrieved data.

DataLoadOptions dlo = new DataLoadOptions();

dlo.AssociateWith<Customer>(c => c.Orders.Where(p => p.ShippedDate != DateTime.Today));

db.LoadOptions = dlo;

LINQ to SQL translates the queries you write into parameterized SQL queries (in text form) and sends them to the SQL server for processing.

When you have an application that executes structurally similar queries many times, you can often increase performance by compiling the query one time and executing it several times with different parameters.

public static Func<Northwnd, string, IQueryable<Customer>>

    CustomersByCity =

        CompiledQuery.Compile((Northwnd db, string city) =>

            from c in db.Customers where c.City == city select c);

Insert rows to database.

// Add the new object to the Orders collection.

db.Orders.InsertOnSubmit(ord);

// Submit the change to the database.

db.SubmitChanges();

Modify rows.

foreach (Order ord in query)

{

    ord.ShipName = "Mariner";

    ord.ShipVia = 2;

}

db.SubmitChanges();

Others like delete, submit, transaction are all supported by LINQ to SQL.

## Manipulating Data (ADO.NET)

#### **Connection**

The .NET Framework Data Provider for OLE DB includes an OleDbConnection object, the .NET Framework Data Provider for SQL Server includes a SqlConnection object, the .NET Framework Data Provider for ODBC includes an OdbcConnection object, and the .NET Framework Data Provider for Oracle includes an OracleConnection object.

Do not call Close or Dispose on a **Connection**, a **DataReader**, or any other managed object in the Finalize method of your class. In a finalizer, only release unmanaged resources that your class owns directly. If your class does not own any unmanaged resources, do not include a Finalize method in your class definition.

Two events of Connection; InfoMessage and StateChange

When using ConnectionString, remember to protect the private information, through ways like Encrypting.

#### Connection Pooling