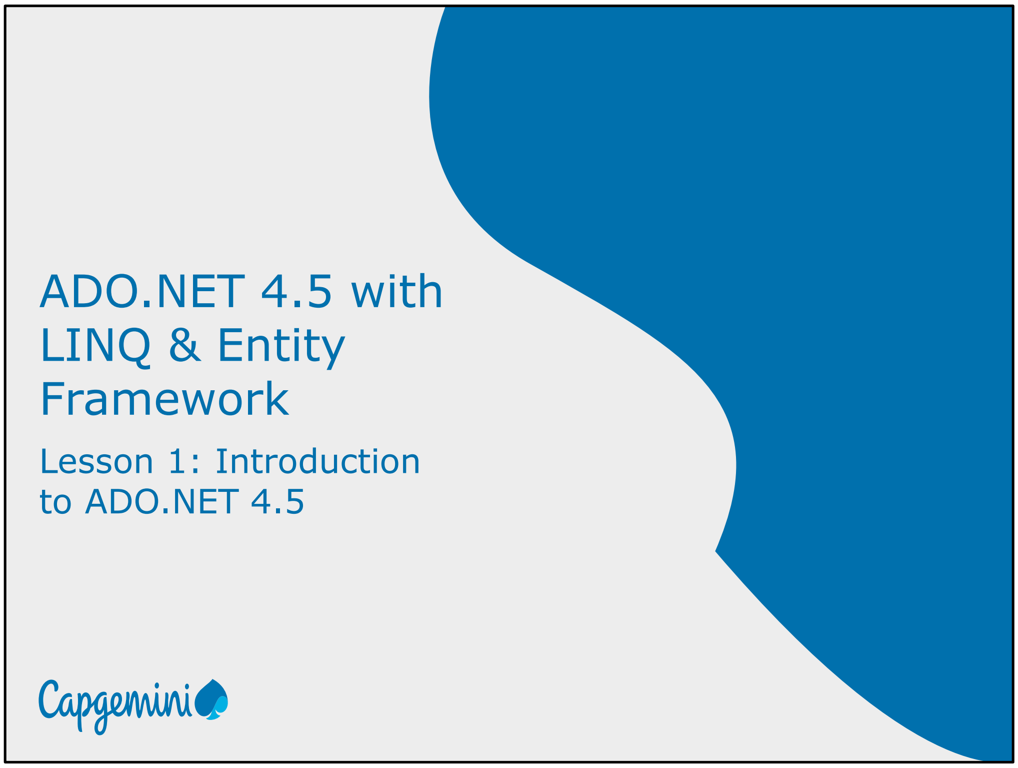


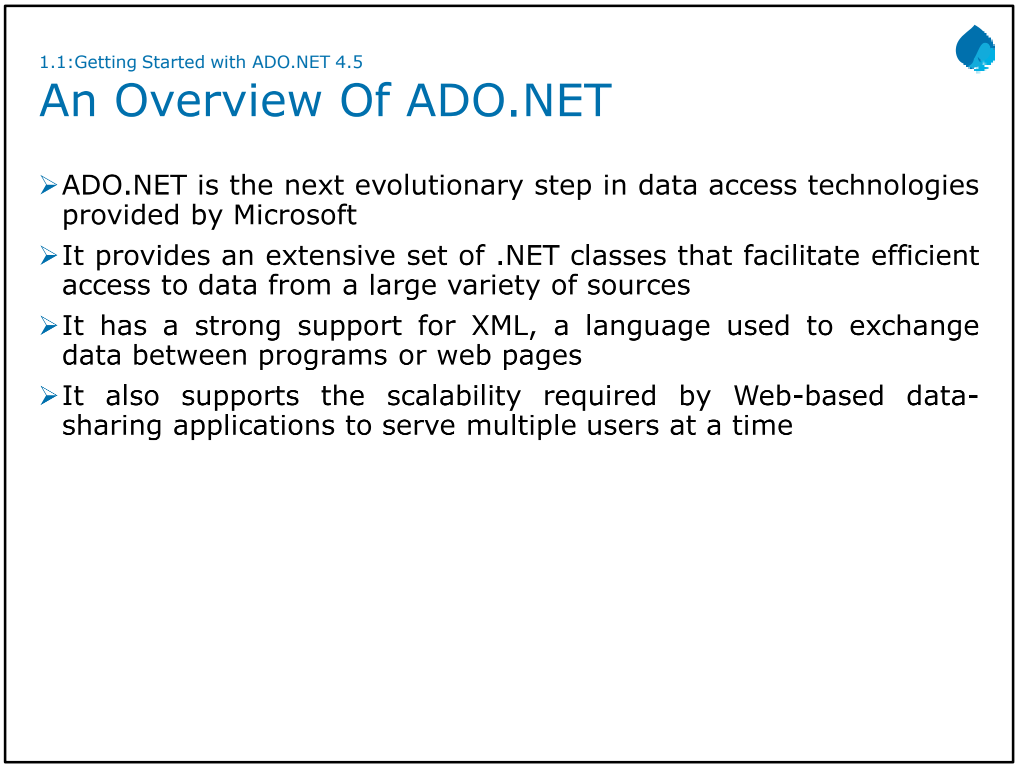
ADO.NET 4.5 with LINQ & Entity Framework

Lesson 1: Introduction to ADO.NET 4.5

Capgemini

Lesson Objectives

- In this lesson, you will learn about:
  - Getting started with ADO.NET 4.5
  - Overview of .NET Data Providers
  - ADO.NET Connected and Disconnected Architecture
  - ADO.NET Generic Classes
  - Best Practices for Managing Connections



1.1:Getting Started with ADO.NET 4.5

An Overview Of ADO.NET

- ADO.NET is the next evolutionary step in data access technologies provided by Microsoft
- It provides an extensive set of .NET classes that facilitate efficient access to data from a large variety of sources
- It has a strong support for XML, a language used to exchange data between programs or web pages
- It also supports the scalability required by Web-based data-sharing applications to serve multiple users at a time

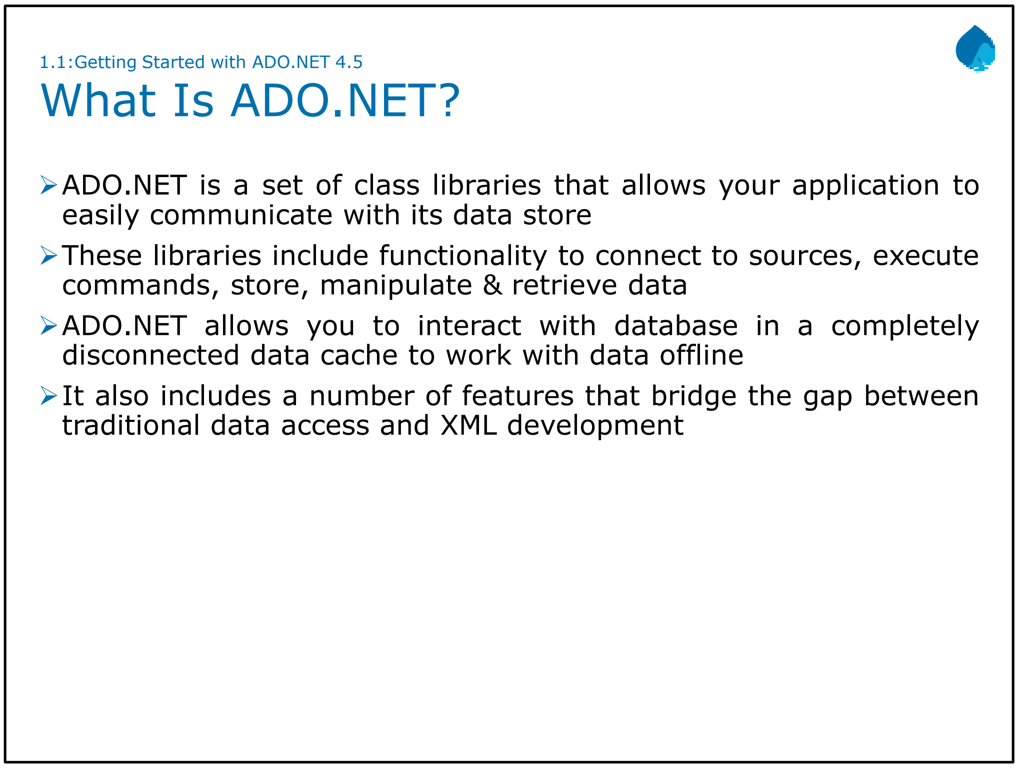
An overview of ADO.NET

ADO.NET is Microsoft's latest data access technology. As an integral part of the .NET Framework it is far more than simply an upgrade of previous incarnations of ActiveX Data Objects ADO. ADO.NET provides an extensive set of .NET classes that facilitate efficient access to data from a large variety of sources also enabling sophisticated manipulation and sorting of data. ADO.NET is essentially a collection of classes that expose methods and attributes used to manage communications between an application and a data store.

ADO.NET is essentially a collection of classes that expose methods used to manage communications between an application and a data store. Being an integral part of the .NET Framework, ADO.NET simplifies integration of data sharing in distributed ASP.NET applications.

Native support for XML is another principal feature for ADO.NET. ADO.NET enables you to create an XML representation of a dataset, with or without its schema. In an XML representation the data is represented in the XML format, and the data metadata is written by using the XML Schema definition language (XSD). XML and XML Schema provide a convenient format for transferring the contents of a actual data to and from remote clients.

ADO.NET also supports the scalability required by Web-based data-sharing applications. Web applications must serve tens, hundreds, or even thousands of users. ADO.NET does not retain lengthy database locks or active connections that monopolize limited resources. This allows the number of users to grow with only minimal increase in the demands on the resources of a system.



1.1:Getting Started with ADO.NET 4.5

What Is ADO.NET?

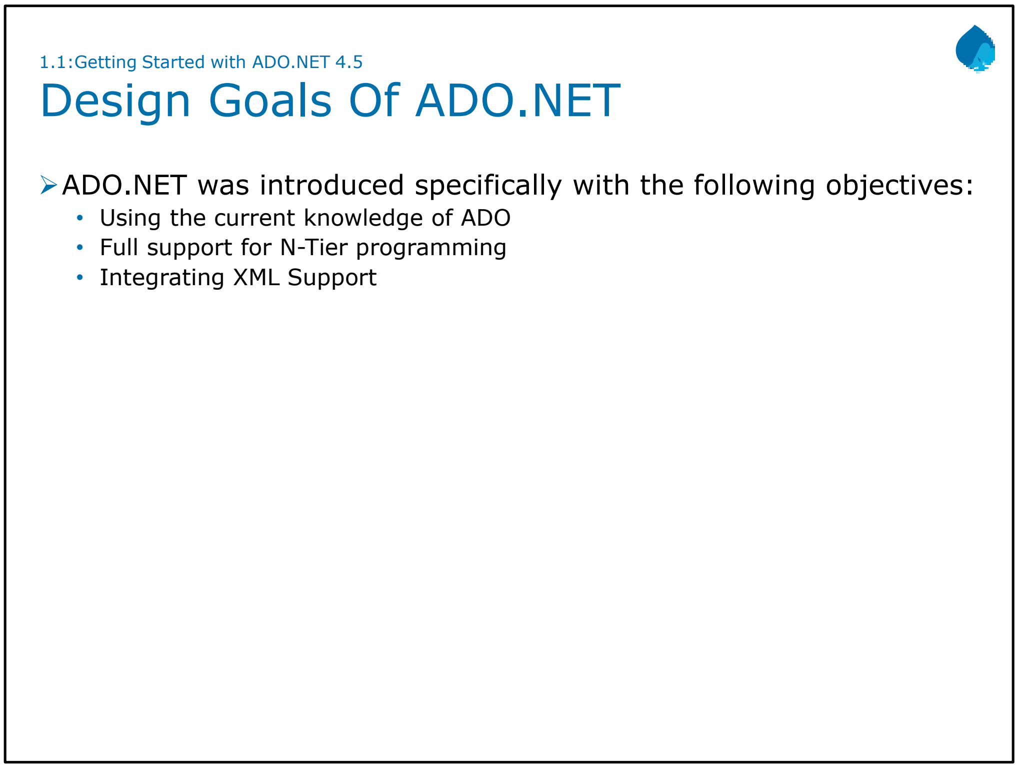
- ADO.NET is a set of class libraries that allows your application to easily communicate with its data store
- These libraries include functionality to connect to sources, execute commands, store, manipulate & retrieve data
- ADO.NET allows you to interact with database in a completely disconnected data cache to work with data offline
- It also includes a number of features that bridge the gap between traditional data access and XML development

What is ADO.NET?

Nearly all business applications need to store large volumes of data and storage of data is accomplished by a database management system. Accessing data has become a major programming task for modern software programming, both for standalone applications and for web applications. A mechanism is needed to work with this data without having to concern low level details such as how data is stored in a database.

Microsoft’s ADO.NET technology offers a solution to many of the problems associated with data access. ADO.NET is a .NET library including a set of classes which expose data access services to any .NET programmer. With ADO.NET you get a rich set of components for creating distributed, data-sharing applications. It is an integral part of the .NET Framework, providing access to relational, XML, and application data. ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers. ADO.NET provides consistent access to data sources such as Microsoft SQL Server and XML, and also 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, manipulate, and update the data that they contain. ADO.NET separates data access from data manipulation into discrete components that can be used separately or in sequence. ADO.NET includes .NET Framework data providers which can be used for connecting to a database, executing commands, and retrieving results. Microsoft's ADO.NET announced the introduction of a disconnected mode of data access, enabling data exchange even across process boundaries efficiently. This is in sharp contrast to the earlier data access technologies with only connected data access mode of operation. It should be noted that ADO.NET supports both connected and disconnected mode of data access. We will be discussing more on the disconnected feature as we go along the course.



1.1:Getting Started with ADO.NET 4.5

Design Goals Of ADO.NET

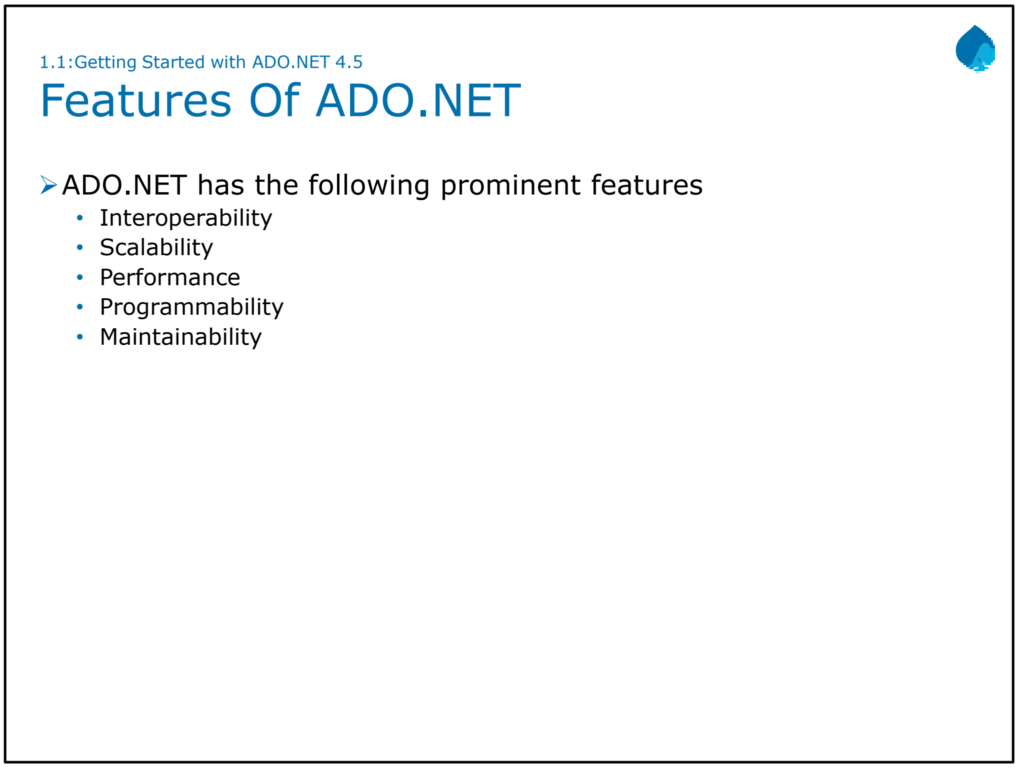
- ADO.NET was introduced specifically with the following objectives:
  - Using the current knowledge of ADO
  - Full support for N-Tier programming
  - Integrating XML Support

Design Goals of ADO.NET
ADO.NET was designed to meet certain needs :

Using the current knowledge of ADO: The design for ADO.NET addresses many of the today's application development model requirements. However, at the same time, the programming model stays as similar as possible to ADO, so that the current ADO developers do not have to start from the scratch. ADO.NET is an intrinsic part of the .NET Framework yet retains familiarity to the ADO programmer.
ADO.NET also coexists with ADO. Although most new .NET-based applications will be written using ADO.NET, ADO remains available to
the .NET programmer through .NET COM interoperability services.

Supporting the N-Tier Programming Model: The concept of working with a disconnected set of data has become a focal point in the programming model. ADO.NET provides first-class support for the disconnected, n-tier programming environment for which many new applications are written.

Integrating XML Support: XML and data access are closely tied. XML is about encoding data, and data access is increasingly becoming about XML. XML support is built into ADO.NET at a very fundamental level. The XML classes in the .NET Framework and ADO.NET are part of the same architecture; they integrate at many different levels. You therefore no longer have to choose between the data access set of services and their XML counterparts; the ability to cross over from one to the other is inherent in the design of both.



1.1:Getting Started with ADO.NET 4.5

Features Of ADO.NET

- ADO.NET has the following prominent features
  - Interoperability
  - Scalability
  - Performance
  - Programmability
  - Maintainability

Features of ADO.NET:
ADO.NET offers several features which are advantageous over the previous data access methods. They can be broadly categorized as:

Interoperability: All data in ADO.NET is transported in XML format. In other words ADO.NET applications takes full advantage of the flexibility and broad acceptance of XML. An ADO.NET application can transmit data in XML format and at the receiving end any component which can be a Visual Studio application or any other application can receive the data. The only requirement is that the receiving component should be able to read XML. As an industry standard, XML was designed keeping this kind of interoperability in mind.

Scalability: Today's implementation model of applications can have huge data requirements, and hence, scalability is necessary. The client/server model is out which had limited amount of requirements. An application that consumes resources such as database locks and database connections may serve hundreds of users well, but as the number increases, it will not continue to do so. ADO.NET promotes the use of disconnected datasets, with automatic connection pooling bundled as part of the package which accommodates scalability.

Performance: Since ADO.NET is mainly about disconnected datasets, the database server is no longer a bottleneck, and hence, applications should incur a performance boost. While transmitting a disconnected recordset among tiers, a significant processing cost can result from converting the values in the recordset to data types. In ADO.NET, such data type conversion is not necessary.

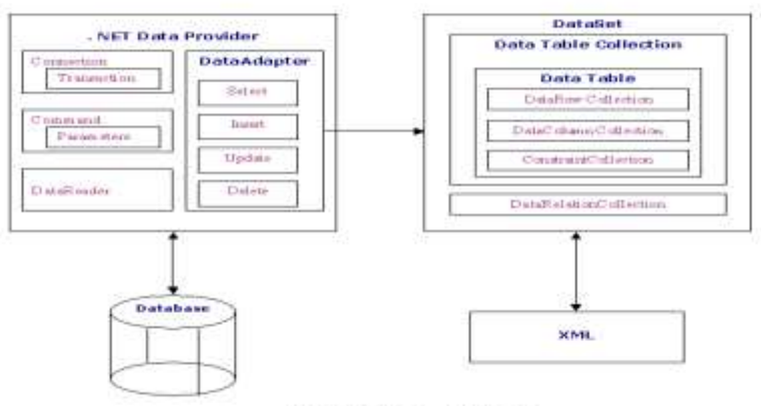
Features of ADO.NET:

Programmability - ADO.NET data components in Visual Studio encapsulate data access functionality in various ways that help you program more quickly and with fewer mistakes. For example, data commands abstract the task of building and executing SQL statements or stored procedures.

Maintainability: In the life of a deployed system, modest changes are possible, but substantial, architectural changes are rarely attempted because they are so difficult. That is unfortunate, because in a natural course of events, such substantial changes can become necessary. For example, as a deployed application becomes popular with users, the increased performance load might require architectural changes. As the performance load on a deployed application server grows, system resources can become scarce and response time or throughput can suffer. Faced with this problem, software architects can choose to divide the server's business-logic processing and user-interface processing onto separate tiers on separate machines. In effect, the application server tier is replaced with two tiers, alleviating the shortage of system resources. The problem is not about designing a three-tiered application. Rather, it is about increasing the number of tiers after an application is deployed. If the original application is implemented in ADO.NET using datasets, this transformation is made easier. Remember, when you replace a single tier with two tiers, you arrange for those two tiers to trade information. Because the tiers can transmit data through XML-formatted datasets, the communication is relatively easy.

1.1:Getting Started with ADO.NET 4.5

ADO.NET Architecture



ADO .NET Data Architecture

The ADO.NET Architecture :

Like any other architecture, there are certain important parts that make up ADO.NET. In this section, you'll look at the various objects that make up ADO.NET. As you probably know, .NET classes can be grouped under namespaces. All ADO.NET-related functionality appears under the System.Data namespace. This doesn't mean that some other software developer cannot write libraries that don't belong to that namespace, but as the Microsoft .NET Framework ships, all ADO.NET-related functionality sits inside the System.Data namespace. Let's further examine the various details of ADO.NET Architecture.

Data Access in ADO.NET relies on two components - .NET Data Providers and DataSet

1.1:Getting Started with ADO.NET 4.5

.NET Data Providers

- A .NET data provider is used for connecting to a database, executing commands, and retrieving results
- Those results are either processed directly, or placed in an ADO.NET DataSets
- They are designed for fast, efficient means of data retrieval and reconciliation
- .NET Data Provider is a set of related components that work together to provide data in an efficient and performance driven manner

The .NET Data Provider is responsible for providing and maintaining the connection to the database. It is a set of related components that work together to provide data in an efficient and performance driven manner.

1.1:Getting Started with ADO.NET 4.5

Types Of .NET Data Providers

- The various providers are:
  - .NET Framework Data Provider for SQL Server
  - .NET Framework Data Provider for Oracle
  - .NET Framework Data Provider for OLE DB
  - .NET Framework Data Provider for ODBC

Data Provider for SQL Server: The .NET Framework Data Provider for SQL Server uses its own protocol to communicate with SQL Server. It is lightweight and performs well because it is optimized to access a SQL Server directly without adding an OLE DB or Open Database Connectivity (ODBC) layer. It provides data access for Microsoft SQL Server version 7.0 or later. The classes for this provider are located in the System.Data.SqlClient namespace. The .NET Framework Data Provider for SQL Server supports both local and distributed transactions. To include this namespace in your application:

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

Data Provider for OLE DB: OleDbDataProvider allows us to connect to other types of databases like Access and Oracle. To use this data provider, include the following namespace in your application:

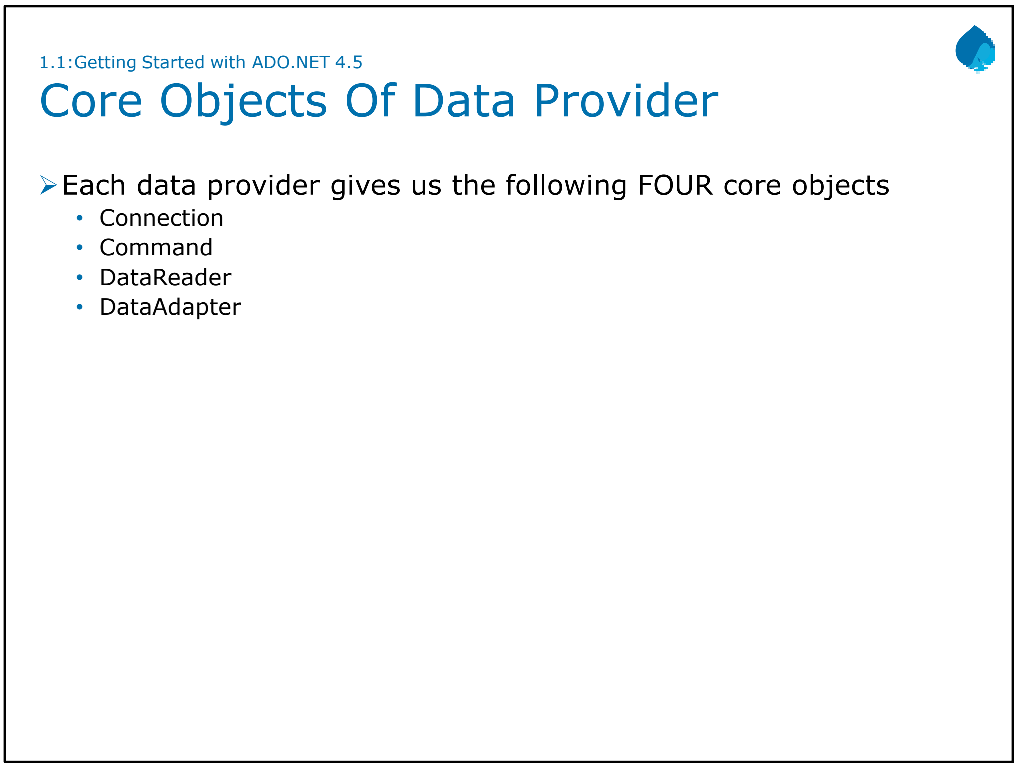
```
using System.Data.OleDb;
```

Data Provider for Oracle: You can use this provider for connecting to Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later. To use this data provider, include the following namespace in your application:

```
using System.Data.OracleClient;
```

Data Provider for ODBC: You can use this provider for connecting to databases like IBM DB2, Informix etc. To use this data provider, include the following namespace in your application:

```
using System.Data.Odbc;
```



1.1:Getting Started with ADO.NET 4.5

Core Objects Of Data Provider

- Each data provider gives us the following FOUR core objects
  - Connection
  - Command
  - DataReader
  - DataAdapter

Core Objects of Data Provider:

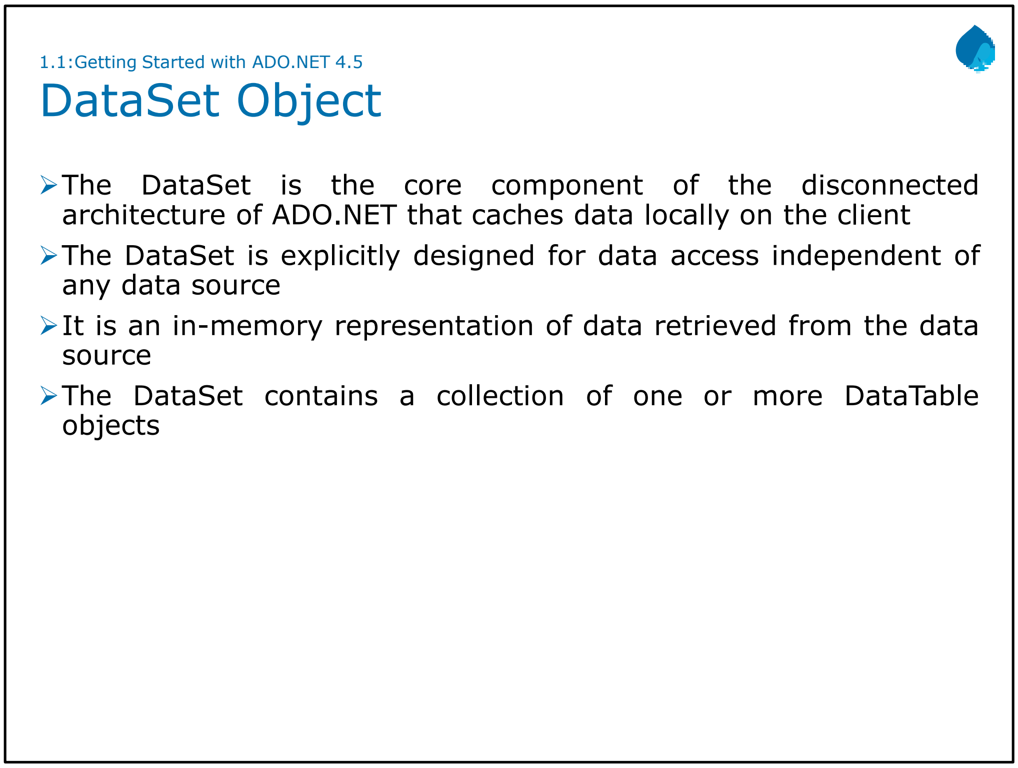
Connection: This is the object that allows you to establish a connection with the data source. Depending on the actual .NET data provider involved, connection objects automatically pool physical database connections for you. It’s important to realize that they don’t pool connection object instances, but they try and recycle physical database connections. Examples of connection objects are OleDbConnection, SqlConnection, OracleConnection, and so on.

• Command: This object represents an executable command on the underlying data source. This command may or may not return any results. These commands can be used to manipulate existing data, query existing data, and update or even delete existing data. In addition, these commands can be used to manipulate underlying table structures. Examples of command objects are SqlCommand, OracleCommand, and so on. A command needs to be able to accept parameters. The Parameter object of ADO.NET allows commands to be more flexible and accept input values and act accordingly.

Core Objects of Data Provider:

DataReader – This object is designed to help you retrieve and examine the rows returned by your query as quickly as possible. You can use the DataReader object to examine the results of a query one row at a time. When you move forward to the next row, the contents of the previous row are discarded. The DataReader doesn't support updating. The data returned by the DataReader is read-only. Because the DataReader object supports such a minimal set of features, it's extremely fast and lightweight. The disadvantage of using a DataReader object is that it requires an open database connection and increases network activity.

DataAdapter: This object acts as a gateway between the disconnected and connected flavors of ADO.NET. The architecture of ADO.NET, in which connection must be opened to access the data retrieved from database is called as connected architecture where as in a disconnected architecture data retrieved from database can be accessed by holding it in a memory with the help of DataSet object even when connection to database was closed (We will be discussing more about both of these architecture and the objects involved in it in detail on the coming lessons). It establishes the connection for you or, given an established connection, it has enough information specified to itself to enable it to understand a disconnected object's data and act upon the database in a prespecified manner. Examples of DataAdapters are SqlDataAdapter, OracleDataAdapter, and so on. It has commands like select, insert, update and delete. Select command is used to retrieve data from database and insert, update and delete commands are used to send changes to the data in dataset to database. It needs a connection to transfer the data.



1.1:Getting Started with ADO.NET 4.5

DataSet Object

- The DataSet is the core component of the disconnected architecture of ADO.NET that caches data locally on the client
- The DataSet is explicitly designed for data access independent of any data source
- It is an in-memory representation of data retrieved from the data source
- The DataSet contains a collection of one or more DataTable objects

DataSet is one of the very important component of ADO.NET Architecture which basically an independent object is explicitly designed for data access independent of any data source.

DataSet Object – Its an in-memory cache of data retrieved from the data source. The DataSet exhibits similar properties to an in-memory relational database - for example, data is organized into multiple tables using DataTable objects, tables can be related using DataRelation objects, and data integrity can be enforced using the constraint objects UniqueConstraint and ForeignKeyConstraint.

Another feature of the DataSet is that it tracks changes that are made to the data it holds before updating the source data. DataSet are also fully XML-featured. They contain methods such as GetXml and WriteXml that respectively produce and consume XML data easily. In an XML scenario where there is no database, these methods enable use of ADO.NET without the Data Provider being involved.

The DataSet object provides a consistent programming model that works with all current models of data storage: flat, relational, and hierarchical. It represents the data that it holds as collections and data types. The data within a DataSet is manipulated via the set of standard APIs exposed through the DataSet and its child objects regardless of its data source.

1.1:Getting Started with ADO.NET 4.5

DataSet Object

- These DataTable objects are made up of rows and columns
- The DataTable objects are the containers which holds actual data in the DataSet
- Along with data it also stores primary key, foreign key, constraint and relation information about DataTable Objects

The DataSet Object in detail

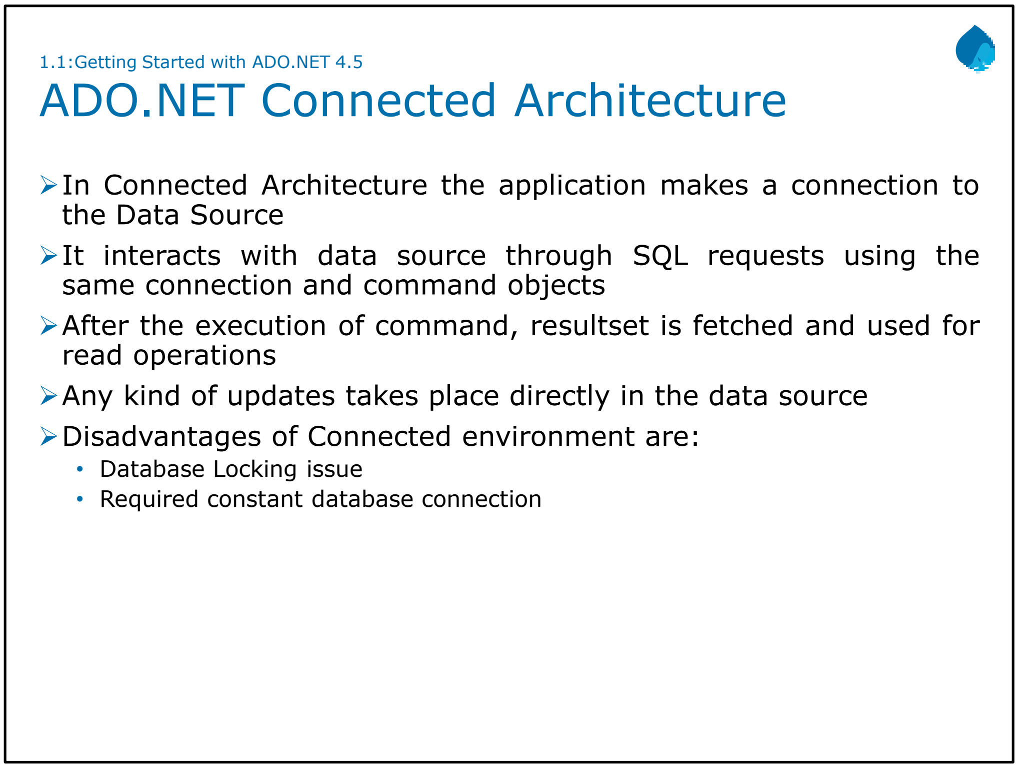
DataTable: A DataSet object is made up of a collection of tables, relationships, and constraints. In ADO.NET, DataTable objects are used to represent the tables in a DataSet object.

DataColumn: Each DataTable object has a Columns collection, which is a container for DataColumn objects. A DataColumn object corresponds to a column in a table.

DataRow: To access the actual values stored in a DataTable object, use the object's Rows collection, which contains a series of DataRow objects.

DataView: Once we've retrieved the results of a query into a DataTable object, we can use a DataView object to view the data in different ways. If we want to sort the contents of a DataTable object based on a column, simply set the DataView object's Sort property to the name of that column. We can also use the Filter property of the DataView so that only the rows that match certain criteria are visible. We can use multiple DataView objects to examine the same DataTable at the same time.

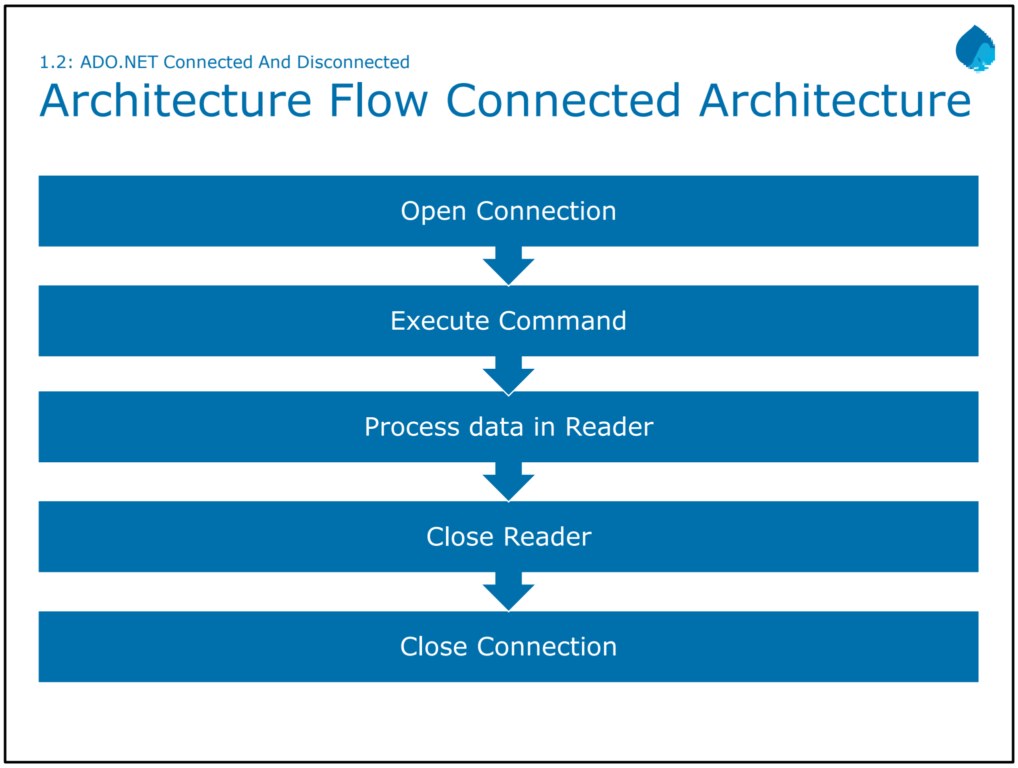
DataRelation: A DataSet, like a database, might contain various interrelated tables. A DataRelation object lets you specify relations between various tables that allow you to both validate data across tables and browse parent and child rows in various DataTables.



1.1:Getting Started with ADO.NET 4.5

ADO.NET Connected Architecture

- In Connected Architecture the application makes a connection to the Data Source
- It interacts with data source through SQL requests using the same connection and command objects
- After the execution of command, resultset is fetched and used for read operations
- Any kind of updates takes place directly in the data source
- Disadvantages of Connected environment are:
  - Database Locking issue
  - Required constant database connection



1.2: ADO.NET Connected And Disconnected

Architecture Flow Connected Architecture

Open Connection

Execute Command

Process data in Reader

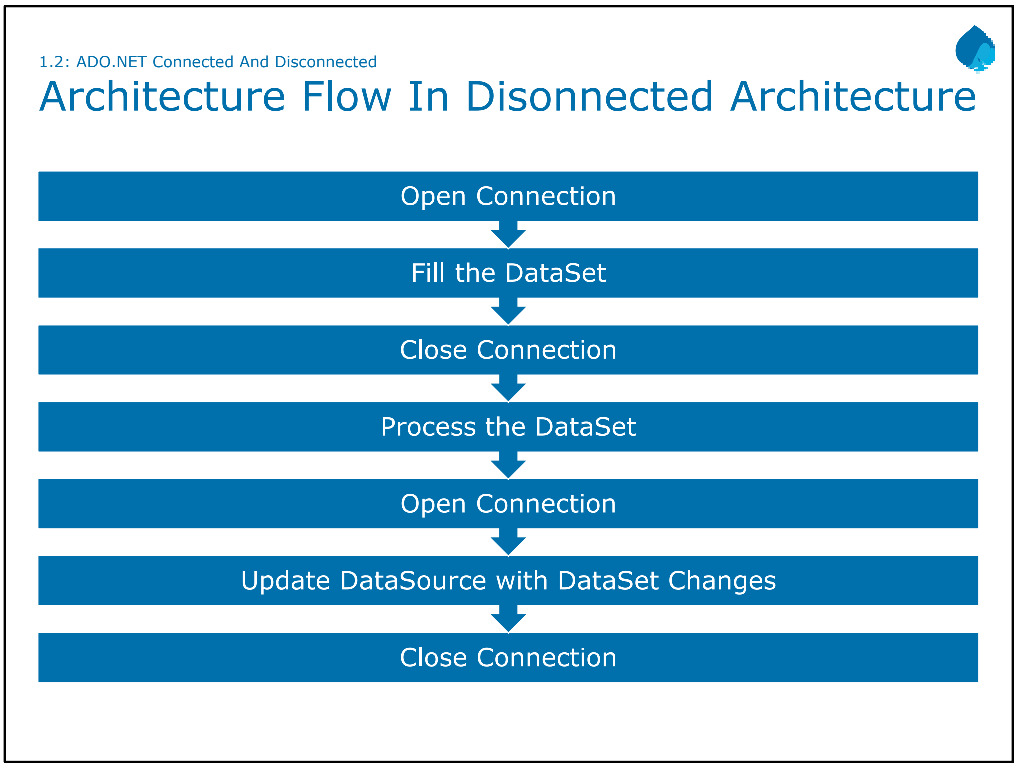
Close Reader

Close Connection

1.2: ADO.NET Connected And Disconnected

Architecture ADO.NET Disconnected Architecture

- In this environment, a subset of data from a central data store can be copied and stored in the DataSet object
- The data is modified independently & changes are merged back into the central data store
- A disconnected environment improves the scalability and performance of an applications
- Disadvantages of disconnected environment are
  - Data is not always up to date
  - Change conflicts can occur and must be resolved



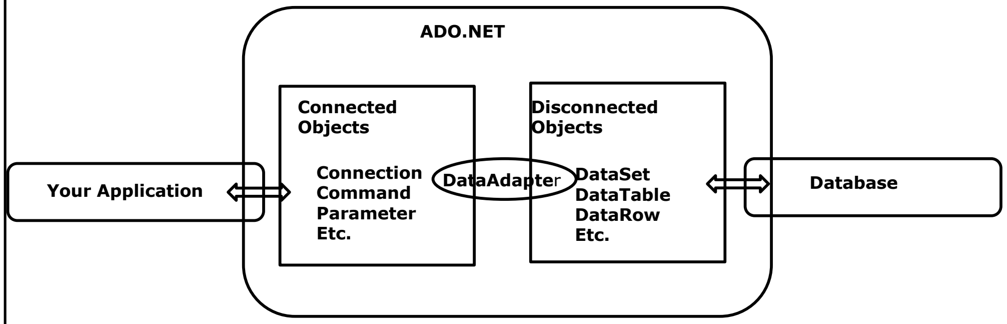
1.2: ADO.NET Connected And Disconnected

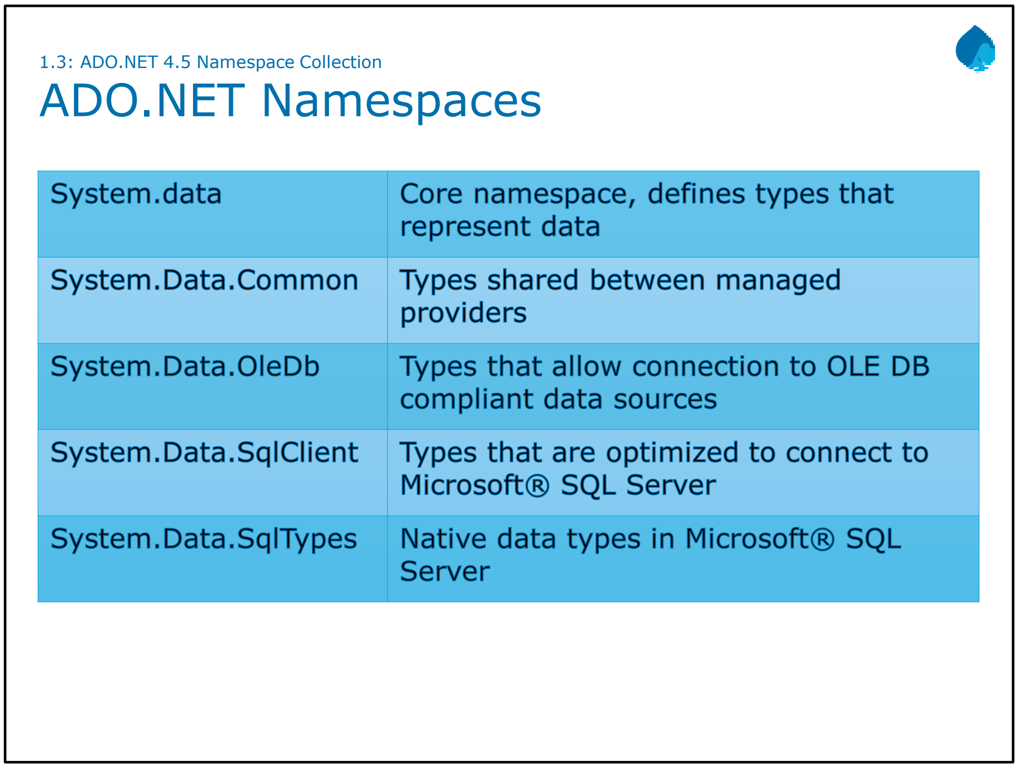
Architecture Flow In Disonnected Architecture

$$\begin{tikzcd}
\text{Open Connection} \arrow[d] \\
\text{Fill the DataSet} \arrow[d] \\
\text{Close Connection} \arrow[d] \\
\text{Process the DataSet} \arrow[d] \\
\text{Open Connection} \arrow[d] \\
\text{Update DataSource with DataSet Changes} \arrow[d] \\
\text{Close Connection}
\end{tikzcd}$$

1.2: ADO.NET Connected And Disconnected

Architecture ADO.NET Connected And Disconnected Object

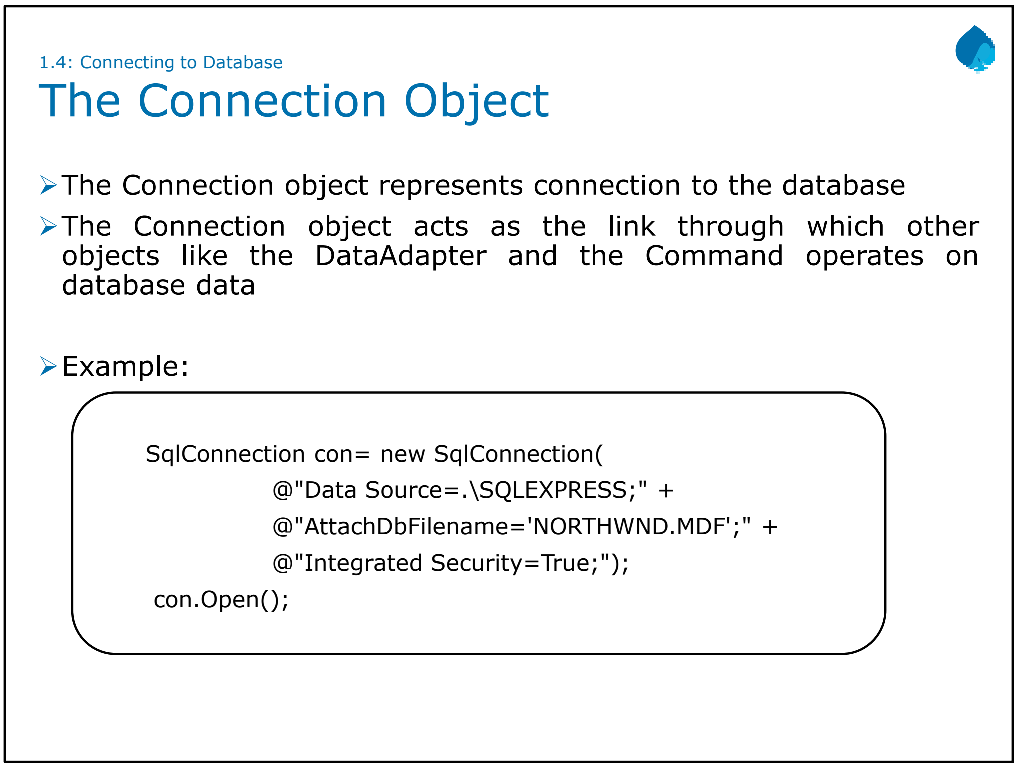




1.3: ADO.NET 4.5 Namespace Collection

ADO.NET Namespaces

System.data	Core namespace, defines types that represent data
System.Data.Common	Types shared between managed providers
System.Data.OleDb	Types that allow connection to OLE DB compliant data sources
System.Data.SqlClient	Types that are optimized to connect to Microsoft® SQL Server
System.Data.SqlTypes	Native data types in Microsoft® SQL Server



1.4: Connecting to Database

The Connection Object

- The Connection object represents connection to the database
- The Connection object acts as the link through which other objects like the DataAdapter and the Command operates on database data

- Example:

```
SqlConnection con= new SqlConnection(
              @"Data Source=.\SQLEXPRESS;" +
              @"AttachDbFilename='NORTHWND.MDF';" +
              @"Integrated Security=True;");
 con.Open();
```

The Connection Object:

Database connections are a critical and limited resource. Connections must be managed to ensure that an application performs well and is scalable.

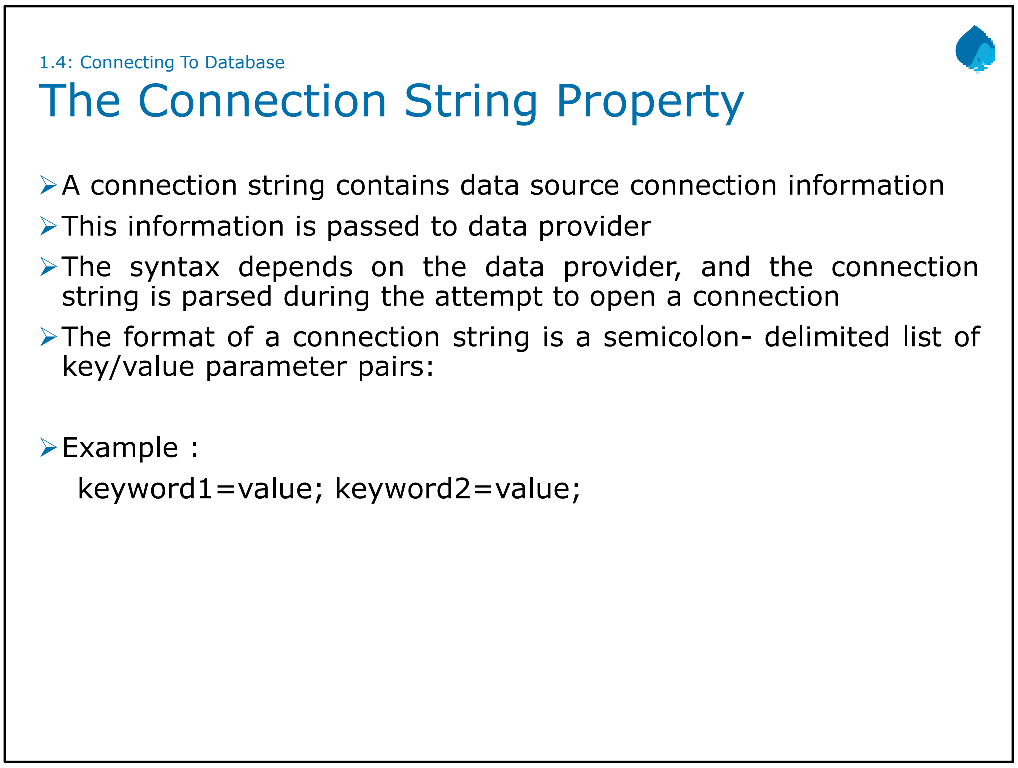
A connection object is used to connect to a specific data source. A SqlConnection object represents a unique session to a SQL Server data source. Similarly an OracleConnection or OLEDBConnection represents unique session to an Oracle data source or OLEDB data source respectively. SQL Server and Oracle data providers provide connection pooling, while the OLE DB and ODBC providers use the pooling provided by OLE DB or ODBC, respectively.

The SqlConnection is used together with SqlDataAdapter and SqlCommand to increase performance when connecting to a Microsoft SQL Server database. For all third-party SQL server products, and other OLE DB-supported data sources, use OleDbConnection.

When you create an instance of SqlConnection, all properties are set to their initial values. The following code examples demonstrate how to create and open connections:

For SQL Server:

```
Using System.Data.SQLClient;
SQLConnection con;
Con=new SQL Connection();
con.open();
```



1.4: Connecting To Database

The Connection String Property

- A connection string contains data source connection information
- This information is passed to data provider
- The syntax depends on the data provider, and the connection string is parsed during the attempt to open a connection
- The format of a connection string is a semicolon- delimited list of key/value parameter pairs:

- Example :

  keyword1=value; keyword2=value;

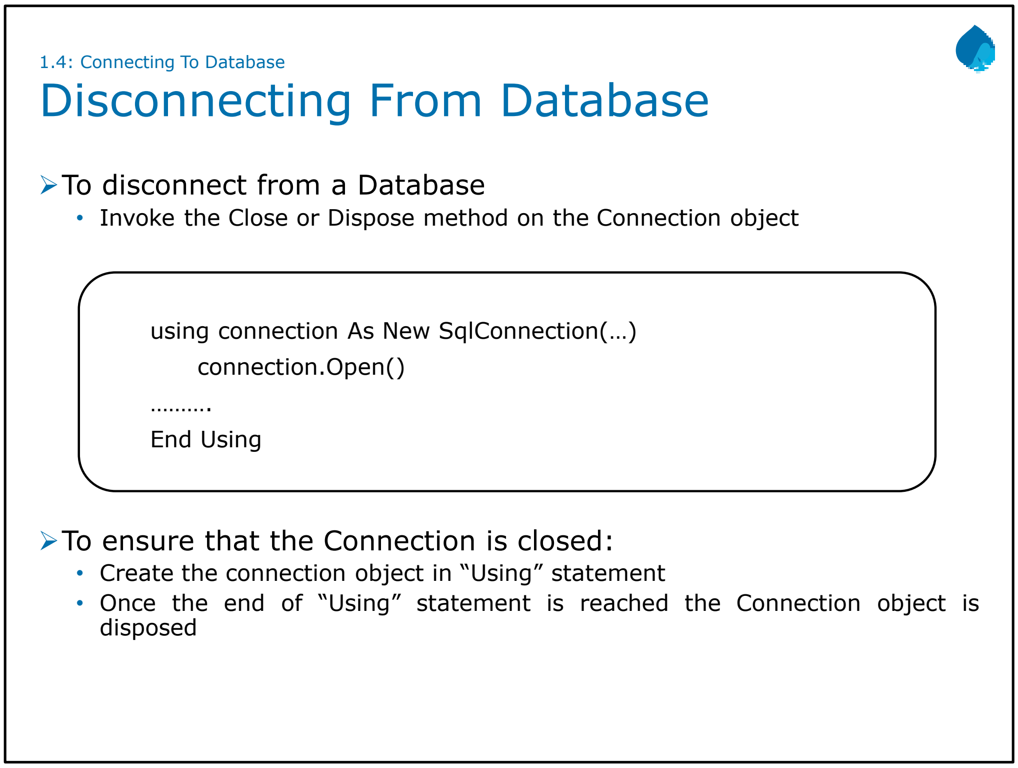
The ConnectionString Property : A connection string contains initialization information that is passed as a parameter from a data provider to a data source. The syntax depends on the data provider, and the connection string is parsed during the attempt to open a connection. Syntax errors generate a run-time exception, but other errors occur only after the data source receives connection information. Once validated, the data source applies the options specified in the connection string and opens the connection.
Keywords are not case sensitive, and spaces between key/value pairs are ignored. However, values may be case sensitive, depending on the data source. Any values containing a semicolon, single quotation marks, or double quotation marks must be enclosed in double quotation marks.

E.g.- "Data Source=SSM0107;Initial Catalog=EMS;User ID=sa;Password=sa"

Connection string parameters

Data Source - The name or network address of the instance of SQL Server to which to connect.
Initial Catalog – The name of the database.
Integrated Security – When false, User ID and Password are specified in the connection. When true, the current Windows account credentials are used for authentication.
User ID/Password – Valid User credentials to get an access to data source.
Persist Security Info - When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state.

Resetting the connection string resets all connection string values including the password. Recognized values are true, false, yes, and no.



1.4: Connecting To Database

Disconnecting From Database

- To disconnect from a Database
  - Invoke the Close or Dispose method on the Connection object

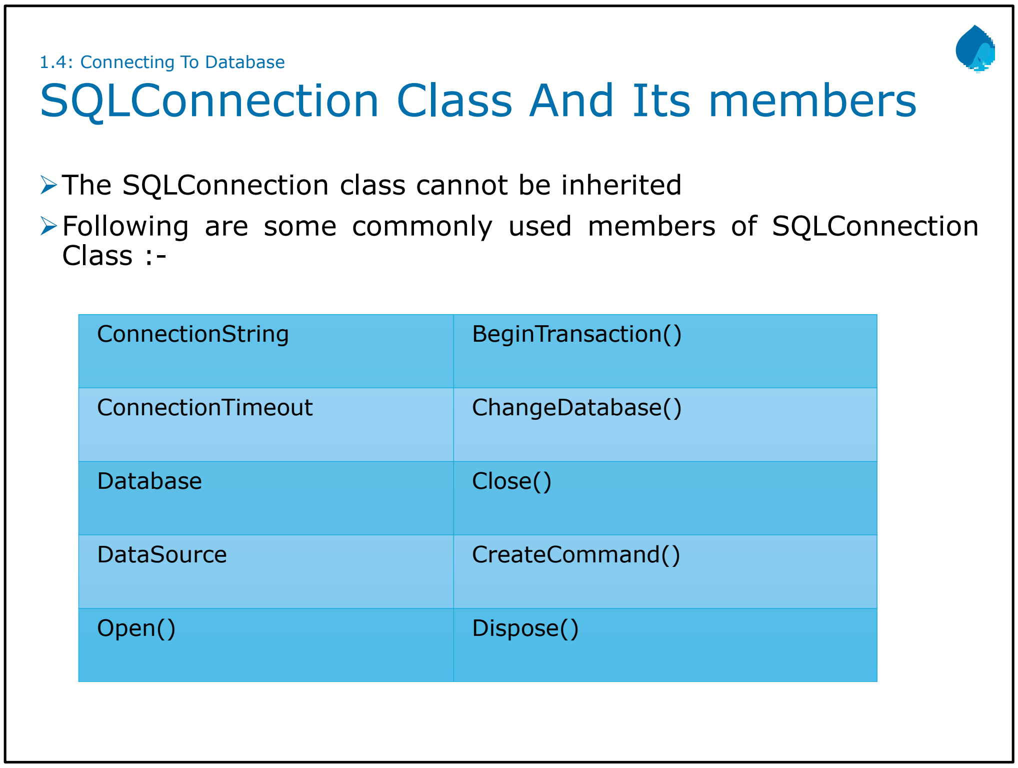
```
using connection As New SqlConnection(…)
    connection.Open()
……….
End Using
```

- To ensure that the Connection is closed:
  - Create the connection object in "Using" statement
  - Once the end of "Using" statement is reached the Connection object is disposed

Disconnecting from Database:

Connections should be opened as late as possible and closed as soon as possible using the Close() method. Alternatively, you can create the connection in a using block to ensure that the system disposes of the connection when the code exits the block. The connection should be used as briefly as possible, meaning that connections should not last longer than a method call. Connections should not be passed between methods—in addition to creating performance problems and limiting scalability, this can lead to security vulnerabilities.

The SQLConnection object acts a link through which other objects like the DataAdaptor and the Command objects communicate with the database and submit queries and retrieve results. Whenever SqlConnection goes out of scope, it remains open. Therefore, you must explicitly close the connection by calling Close or Dispose methods. Close and Dispose are functionally equivalent.



1.4: Connecting To Database

SQLConnection Class And Its members

- The SQLConnection class cannot be inherited
- Following are some commonly used members of SQLConnection Class :-

ConnectionString	BeginTransaction()
ConnectionTimeout	ChangeDatabase()
Database	Close()
DataSource	CreateCommand()
Open()	Dispose()

SQLConnection class and its members:

The SQLConnection class cannot be inherited. As specified earlier, the SQLConnection represents open connection to SQL server database. The SQLConnection class exposes various properties and methods which allows a .NET programmer the ease to work with the Connection Object. Some of the members are as follows:

ConnectionString: Gets or sets the string to open a SQL Server database.
ConnectionTimeout: The wait time while trying to establish a connection before the attempt is terminated and error is generated
Database: Obtains 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
Open(): Opens a database connection with the property settings specified by the ConnectionString
BeginTransaction(): Starts 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.

Working with the InfoMessage Event

You can retrieve warnings and informational messages from a SQL Server data source using the InfoMessage event of the SqlConnection object.

The following code example shows how to add an event handler for the InfoMessage event.

```
// Assumes that connection represents a SqlConnection object.
connection.InfoMessage += new
SqlInfoMessageEventHandler(OnInfoMessage);

protected static void OnInfoMessage(object sender,
SqlInfoMessageEventArgs args)
{
     foreach (SqlError err in args.Errors)
     {
          Console.WriteLine( "The {0} has received a severity {1}, state {2}
          error number {3}\n" + "on line {4} of procedure {5} on server
          {6}:\n{7}", err.Source, err.Class, err.State, err.Number,
          err.LineNumber,err.Procedure, err.Server, err.Message);
    }
}
```

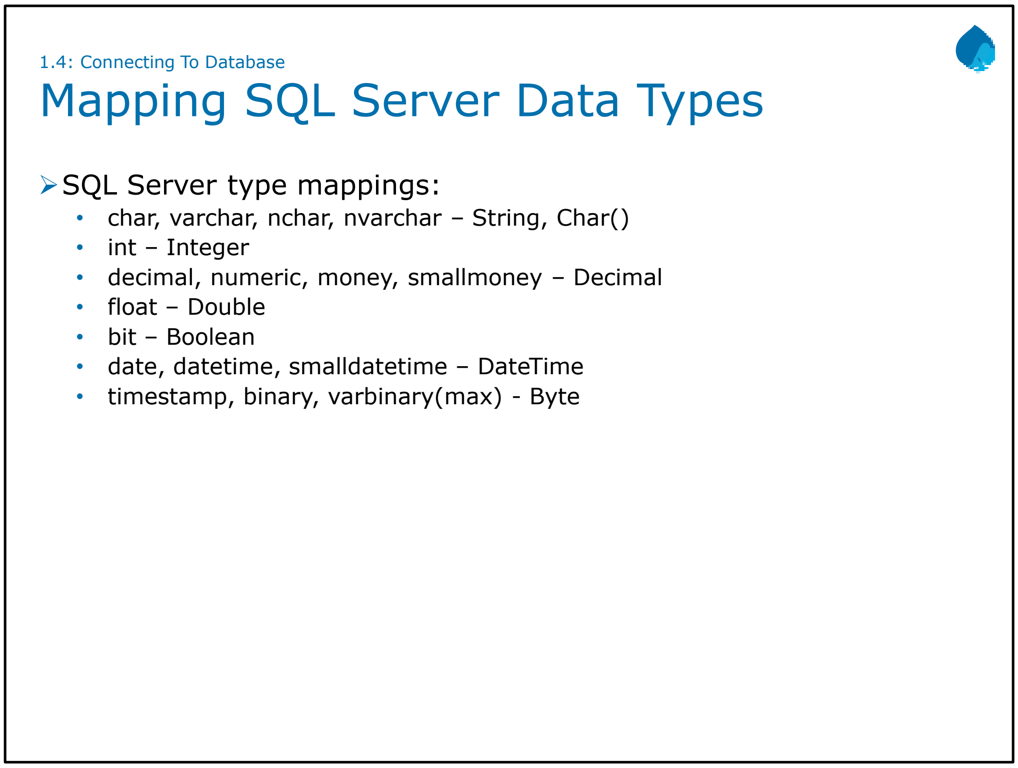
Working with the StateChange Event

The StateChange event occurs when the state of a Connection changes. The StateChange event receives StateChangeEventArgs that enable you to determine the change in state of the Connection by using the OriginalState and CurrentState properties. The OriginalState property is a ConnectionState enumeration that indicates the state of the Connection before it changed. CurrentState is a ConnectionState enumeration that indicates the state of the Connection after it changed.

The following code example uses the StateChange event to write a message to the console when the state of the Connection changes.

```
// Assumes connection represents a SqlConnection object.
connection.StateChange += new StateChangeEventHandler(OnStateChange);

protected static void OnStateChange(object sender, StateChangeEventArgs
args)
 {
     Console.WriteLine( "The current Connection state has changed from {0} to
     {1}.", args.OriginalState,args.CurrentState);
 }
```



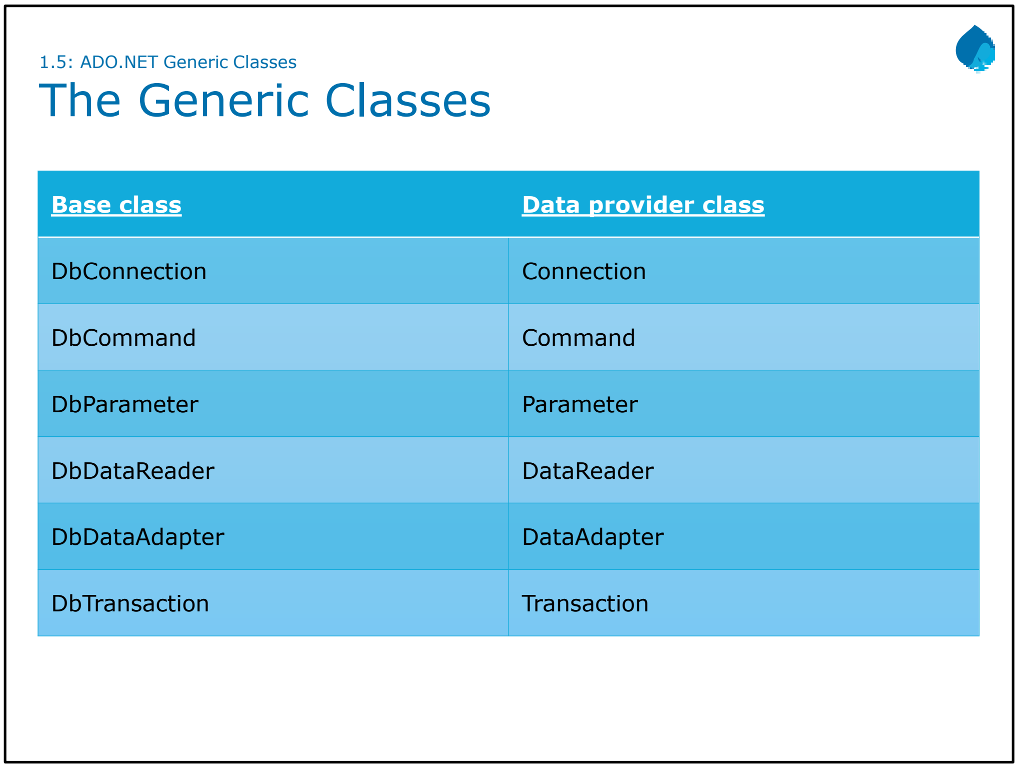
1.4: Connecting To Database

Mapping SQL Server Data Types

- SQL Server type mappings:
  - char, varchar, nchar, nvarchar – String, Char()
  - int – Integer
  - decimal, numeric, money, smallmoney – Decimal
  - float – Double
  - bit – Boolean
  - date, datetime, smalldatetime – DateTime
  - timestamp, binary, varbinary(max) - Byte

Mapping SQL Server Data Types to .Net Types:

Databases and .Net Framework are based on different type systems. Hence each .Net data provider defined mappings between database types and .Net Framework types to maintain data integrity when reading and writing.
In our course, since we are dealing with examples based on SQL Server we take a look at the Mapping SQL Server Datatypes to .Net Framework Datatypes.



1.5: ADO.NET Generic Classes

The Generic Classes

Base class	Data provider class
DbConnection	Connection
DbCommand	Command
DbParameter	Parameter
DbDataReader	DataReader
DbDataAdapter	DataAdapter
DbTransaction	Transaction

The Generic Classes:

The table represents the Generic Base Classes and their corresponding Provider Specific classes. That means SqlConnection and OracleConnection classes inherit from a common base class called DbConnection. As we progress along in the course, we will see how to use the provider specific class.

You can take advantage of polymorphism via inheritance and write generic code capable of dealing with multiple database in neutral manner. ADO.NET offers much more than common base classes. In addition to the classes mentioned in the table shown on the slide, ADO.NET also provides a set of classes called "Factory" classes. These factory classes help you to create instances of these base classes dynamically. This ability makes it possible to store the choice of your data provider in a configuration file and then at run time create instances of corresponding data provider classes. For example, SQL server data provider has a factory class called SqlClientFactory that allows you to create instances of SqlConnection, SqlCommand and so on. Similar classes exists for other data providers also.

The SqlClientFactory class has various methods shown as follows:-

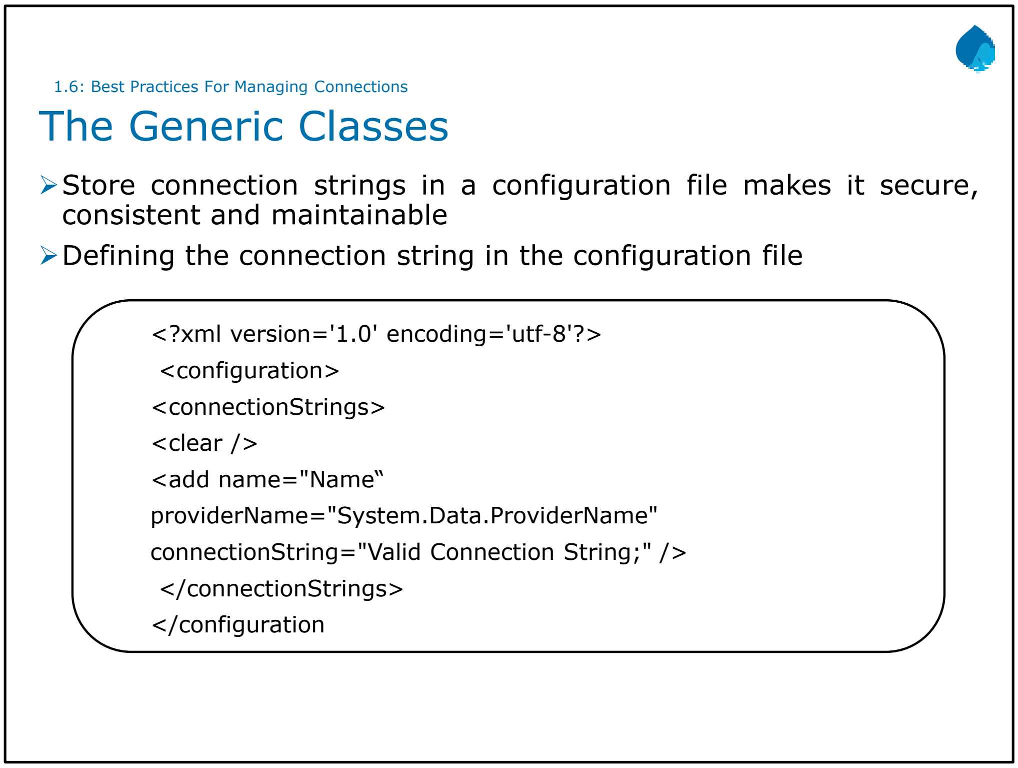
CreateConnection - Creates an instance of SqlConnection class
CreateCommand - Creates an instance of SqlCommand class
CreateParameter - Creates an instance of SqlParameter class
CreateDataAdapter - Creates an instance of SqlDataAdapter class
CreateCommandBuilder - Creates an instance of SqlCommandBuilder class

The Generic Classes:

Consider the following code snippet:

```
public void ExecuteQuery(string sql,string provider)
{
  DbConnection cnn=null;
  DbCommand cmd=null;
  DbProviderFactory factory = null;
       switch(provider)
        {
          case "sqlclient":
          factory = SqlClientFactory.Instance;
          break;
          case "oracleclient":
          factory = OracleClientFactory.Instance;
          break;
      }
   cnn = factory.CreateConnection();
   cmd = factory.CreateCommand();
  //now use cnn and cmd as usual to execute a query
}
```

Note, the usage of Instance property of SqlClientFactory and OracleClientFactory classes to get an instance of corresponding factory class.



1.6: Best Practices For Managing Connections

The Generic Classes

- Store connection strings in a configuration file makes it secure, consistent and maintainable
- Defining the connection string in the configuration file

```
<?xml version='1.0' encoding='utf-8'?>
 <configuration>
<connectionStrings>
<clear />
<add name="Name‟
providerName="System.Data.ProviderName"
connectionString="Valid Connection String;" />
 </connectionStrings>
</configuration
```

Storing Connections in Configuration file:

Database connections are a critical and limited resource. Connections must be managed to ensure that an application not only performs well but is also scalable. Data Providers use a connection string to establish connection to the database. Where should a developer store the connections string? Ideally the connection string should be stored in such a place wherein it increases maintainability of the application and also eliminates the need to recompile the application when it is modified.
The popular option would be store the connection string in the configuration file. An application configuration file is an XML based file to store application specific settings. Storing the connection string in the configuration file makes it consistent, secure and maintainable.
The connectionStrings Section - Connection strings can be stored as key/value pairs in the connectionStrings section of the configuration element of an application configuration file. Child elements include add, clear, and remove. The connection string can be placed in the configuration file as shown on the slide.

```
// Retrieves a connection string by name.
// Returns null if the name is not found.
static string GetConnectionStringByName(string name)
{
// Assume failure.
string returnValue = null;
// Look for the name in the connectionStrings section. ConnectionStringSettings
settings = ConfigurationManager.ConnectionStrings[name];
 // If found, return the connection string.
 if (settings != null)
returnValue = settings.ConnectionString;
return returnValue;
}
```

Connecting to a Database

- Open connections late and close them early
- Store connection strings securely
- Use Connection String Builder to create the type safe connection string
- Storing Connection Strings in external files like .config
- UDL (Universal Data Link) are considered to be good options as they offer much flexibility

Managing Connections:
Database connections represents a very critical, expensive, and limited resource. On optimizing how you use them is therefore very important for any application; not just for .NET Framework applications, but in particular for multi-tier Web applications. The bottom line for database connections can be summarized in the following two points:

Store connection strings securely
Open connections late and close them early

For connecting to a database, you can follow the steps mentioned below:
Define a string and assign the connection information to it.
Create the connection object using the connection string.
Invoke the open method on the Connection object.
Close the Connection object when not in use.

For creating a type safe connection you can use the ConnectionBuilder object as follows:

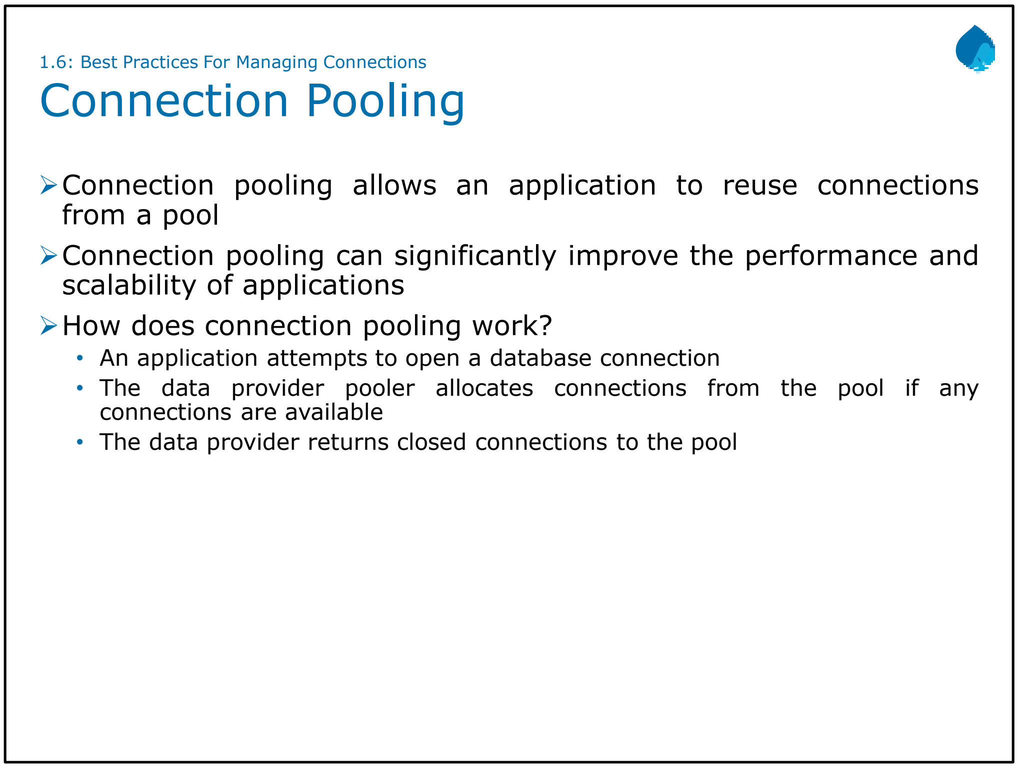
```
SqlConnectionStringBuilder sb= New
SqlConnectionStringBuilder()
sb("Data Source") = “192.166.67.176”
sb("Initial Catalog") = “Northwind"
String  conn=sb.ConnectionString;
```

Keep connections open to the database only when required. Reduce the number of times you open and close a connection for multiple operations.

1.6: Best Practices For Managing Connections

Connecting to a Database

- Every technique of storing connection string suffers from a security threat
- We can have a tailor-made engine for encrypting and decrypting connection details



1.6: Best Practices For Managing Connections

Connection Pooling

- Connection pooling allows an application to reuse connections from a pool
- Connection pooling can significantly improve the performance and scalability of applications
- How does connection pooling work?
  - An application attempts to open a database connection
  - The data provider pooler allocates connections from the pool if any connections are available
  - The data provider returns closed connections to the pool

Connection Pooling:

Connection pooling enables connections to be reused efficiently instead of repeatedly creating and destroying new connections. Data Providers pool connections that have same security context. SQL Server and Oracle data providers provide connection pooling. If Integrated Security is used then Connection Pool is created for each user accessing the client system, and if userid and password is used then single connection pool is maintained across for the application. In case of using user id and password, each user can use the connections of the pool created and then released to the pool by other users. This option is recommended for better performance experience for the end user.

A connection pool is created for each unique connection string. An algorithm associates items in the pool based on an exact match with the connection string; this includes capitalization, order of name/value pairs, and even spaces between name/ value pairs.

Properties used while specifying connection string to implement Connection Pooling
.

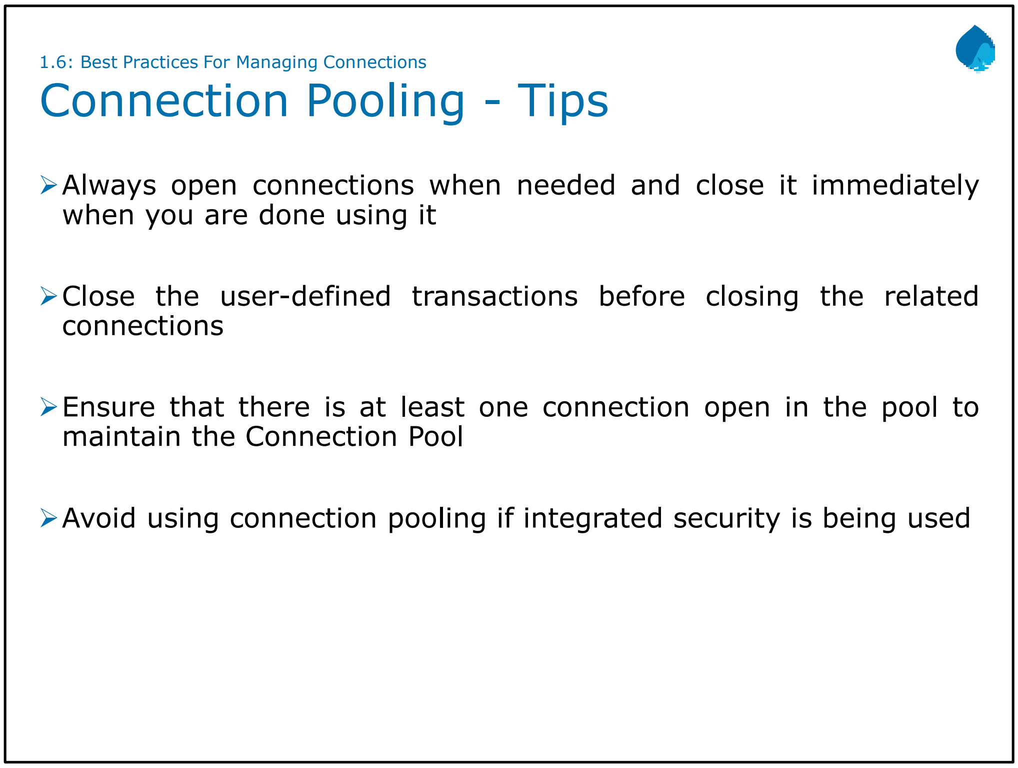
Connection Timeout – Default is 15. Maximum Time (in secs) to wait for a free connection from the pool.

Min Pool Size – Default is 0. The minimum number of connections allowed in the pool.

Max Pool Size – Default is 100. The maximum number of connections allowed in the pool.

Incr Pool Size – Default is 5. Controls the number of connections that are established when all the connections are used.

Decr Pool Size – Default is 1. Controls the number of connections that are closed when an excessive amount of established connections are unused.



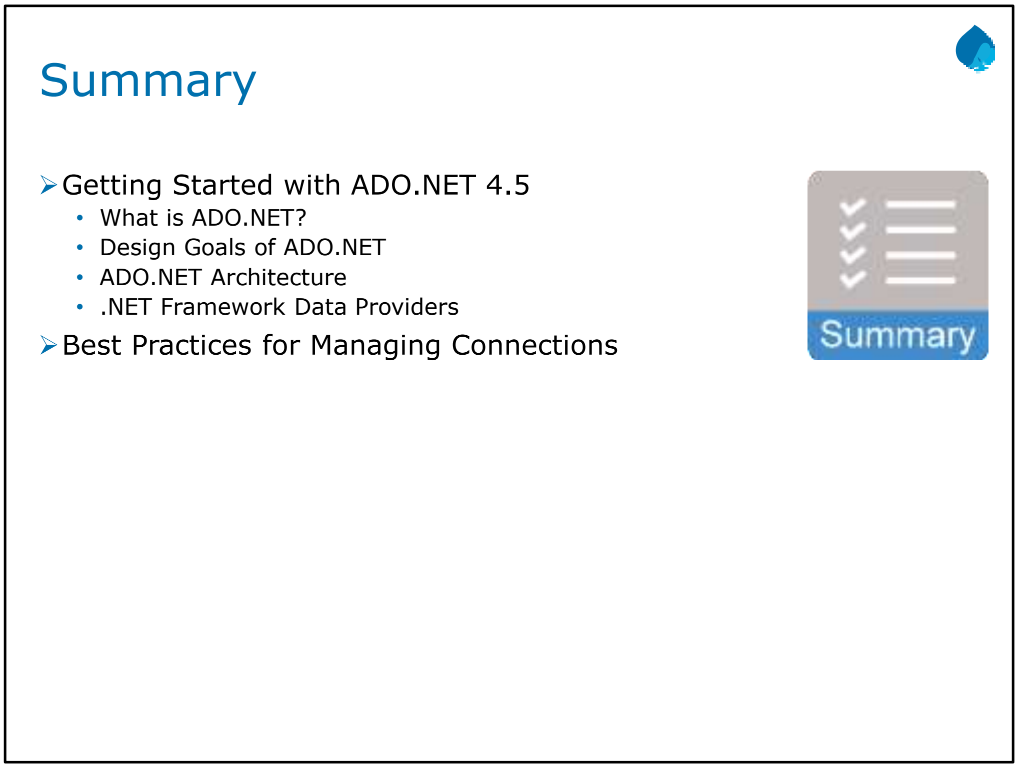
1.6: Best Practices For Managing Connections

Connection Pooling - Tips

- Always open connections when needed and close it immediately when you are done using it
- Close the user-defined transactions before closing the related connections
- Ensure that there is at least one connection open in the pool to maintain the Connection Pool
- Avoid using connection pooling if integrated security is being used

Connection Pooling – Tips

Only open connections when needed. That is, timing is everything, so open a connection just before you need it and not any sooner. Also, close that connection as soon as you are finished with it—don't wait for the garbage collector to do it.
Close user-defined transactions before closing related connections.
To maintain the connection pool, you should keep at least one connection open. Therefore, do not close all your connections in the pool. If server resources become a problem, you may close all connections, and the pool will be recreated with the next request.
Do not use connection pooling if integrated security is utilized. This results in a unique connection string per user, so each user has a connection pool that is not available to other users. The end result is poor performance, so pooling should be avoided in this scenario.



Summary

- Getting Started with ADO.NET 4.5
  - What is ADO.NET?
  - Design Goals of ADO.NET
  - ADO.NET Architecture
  - .NET Framework Data Providers
- Best Practices for Managing Connections

Add the notes here.

Review Question

- Question 1: Which of the following are design goals of ADO.NET?
  - Using current knowledge of ADO
  - Support for N-Tier programming
  - None of the above

Add the notes here.

Review Question: Match the Following

1. Numeric	DateTime
2. Float	Double
3. Bit	Decimal
4. Date	Byte
5. TimeStamp	Boolean

Add the notes here.