ADO.NET 4.5

What is ADO.Net?

ADO.NET. ADO.NET is a data access technology from the Microsoft .NET Framework that provides communication between relational and **non**-relational systems through a common set of components. ADO.NET is a set of computer software components that programmers can use to access data and data services from a database.

Lessons Overview

Lesson 1:- Introduction to ADO.Net 4.5

This lesson will cover the following topics

- Overview of ADO.Net
- ADO.Net Architecture
- ADO.Net Connected and Disconnected Architecture
- ADO.Net Generic Classes

Overview of ADO.NET

ADO.NET is a set of classes that expose data access services to the .NET programmer.

ADO.NET provides functionality to developers writing managed code similar to the functionality Provided to native COM developers by ADO.

ADO.NET provides consistent access to data sources such as Microsoft SQL Server, as well as data sources exposed through OLE DB and XML.

Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, manipulate, and update data.

ADO.NET provides a rich set of components for creating distributed, data-sharing applications.
It is an integral part of the .NET Framework, providing access to relational data, XML, and application data.

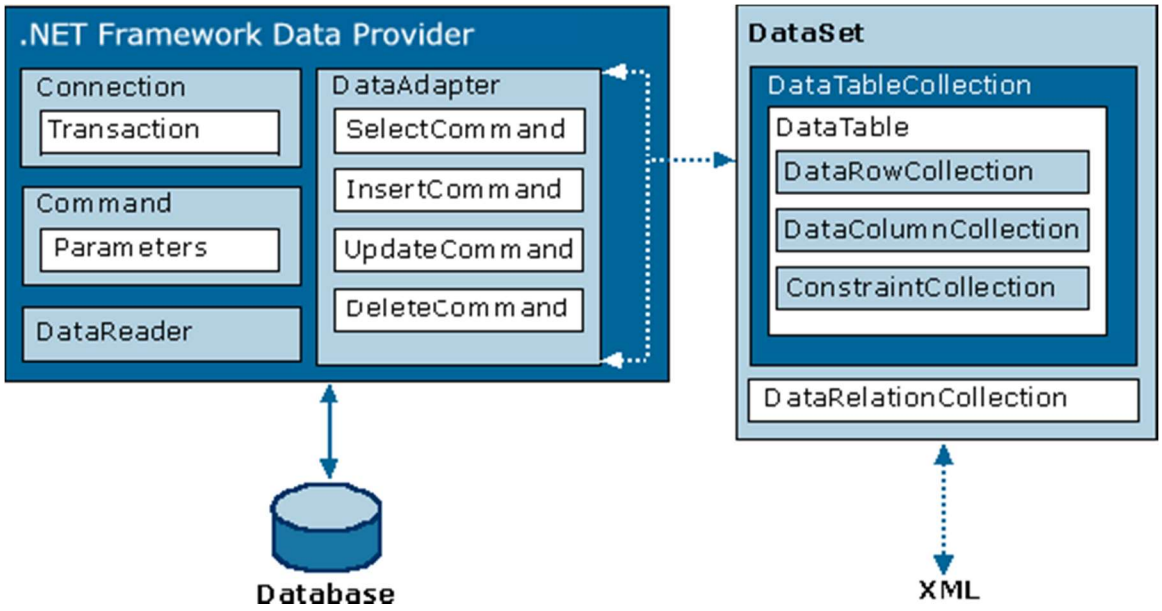
ADO.NET supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, or Internet browsers.

ADO.Net Architecture

The ADO.NET components have been designed to factor data access from data manipulation. There are two central components of ADO.NET that accomplish this: the DataSet, and the .NET Framework data provider, which is a set of components including the Connection**,** Command**,** DataReader, and DataAdapter objects.

The ADO.NET DataSet is the core component of the disconnected architecture of ADO.NET.
The DataSet is explicitly designed for data access independent of any data source. As a result it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application. The DataSet contains a collection of one or more DataTable objects made up of rows and columns of data, as well as primary key, foreign key, constraint, and relation information about the data in the DataTable objects.

The other core element of the ADO.NET architecture is the .NET Framework data provider, whose components are explicitly designed for data manipulation and fast, forward-only, read-only access to data. 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 made to the data in the DataSet back to the data source.



List of .Net Data Providers

-.Net Data Provider for SQL Server

-.Net Data Provider for Oracle

-.Net Data Provider for ODBC

-.Net Data Provide for OLEDB

ADO.Net Connected Architecture

The architecture of ADO.net, in which connection must be opened to access the data retrieved from database is called as connected architecture.

Connected architecture was built on the classes connection, command, datareader and transaction.

Connected architecture is when you constantly make trips to the database for any CRUD (Create, Read, Update and Delete) operation you wish to do.

This creates more traffic to the database but is normally much faster as you should be doing smaller transactions.

Connected Architecture

Disconnected Architecture

The architecture of ADO.net in which data retrieved from database can be accessed even when connection to database was closed is called as disconnected architecture.

Disconnected architecture of ADO.net was built on classes connection, dataadapter, commandbuilder and dataset and dataview.

Disconnected architecture is a method of retrieving a record set from the database and storing it giving you the ability to do many CRUD (Create, Read, Update and Delete) operations on the data in memory, then it can be re-synchronized with the database when reconnecting.

A method of using disconnected architecture is using a Dataset.

ADO.Net Generic Classes

ADO.NET 2.0 introduced new base classes in the System.Data.Common namespace. The base classes are abstract, which means that they can't be directly instantiated.

They include DbConnection, DbCommand, and DbDataAdapter and are shared by the .NET Framework data providers, such as System.Data.SqlClient and System.Data.OleDb.

The addition of base classes simplifies adding functionality to the .NET Framework data providers without having to create new interfaces.

ADO.NET 2.0 also introduced abstract base classes, which enable a developer to write generic data access code that does not depend on a specific data provider.

Lesson 2:- Working with Connected Architecture

This lesson will cover the following topics

- Overview of Connected Architecture Classes
- Data manipulation using connected architecture

Overview of Connected Architecture Classes

SqlConnection Class:-

SqlConnection class represents a connection to SQL Server Database. This class cannot be inherited.

A SqlConnection object represents a unique session to a SQL Server data source.

With a client/server database system, it is equivalent to a network connection to the server.

SqlConnection is used together with SqlDataAdapter and SqlCommand to increase performance when connecting to a Microsoft SQL Server database.

SqlCommand Class:-

SqlCommand class represents a Transact-SQL statement or stored procedure to execute against a SQL Server database. This class cannot be inherited.

Following are some of the property and method SqlCommand Class

CommandText :- Gets or sets the Transact-SQL statement, table name or stored procedure to execute at the data source.

Connection :- Gets or sets the SqlConnection used by this instance of the SqlCommand.

CommandType :- Gets or sets a value indicating how the CommandText property is to be interpreted. The possible values for Command Type are Text, StoredProcedure and TableDirect

ExecuteReader() :- Sends the CommandText to the Connection and builds a SqlDataReader. Its is used to read data from the Database table

ExecuteNonQuery() :- Executes a Transact-SQL statement against the connection and returns the number of rows affected. This is used for execute Insert, Update and Delete Statement.

ExecuteScalar() :- Executes the query, and returns the first column of the first row in the result set returned by the query. Additional columns or rows are ignored.

SqlDataReader class:-

SqlDataReader class provides a way of reading a forward-only stream of rows from a SQL Server database. This class cannot be inherited.

Following are some of the property and method SqlDataReader Class

HasRows:- Gets a value that indicates whether the SqlDataReader contains one or more rows.

Read() :- Advances the SqlDataReader to the next record.

NextResult() :- Advances the data reader to the next result, when reading the results of batch Transact-SQL statements.

Passsing Parameters to Command Object

Every Command object has an associated collection of parameter objects
The Parameter object is a provider-specific object
SqlCommand uses a SqlParameter, an Ole Db Command uses an Ole Db Parameter, and so on
SQL statements and stored procedures can take input, output, and bidirectional parameters
Stored procedures can also return a value
You must configure your command object so that it handles parameters and return values correctly

Lesson 3 :- Working with Disconnected Architecture

This lesson will cover the following topics

- Classes used in Disconnected Architecture
- Creation and use of Data Set to retrieve data
- Manipulation of database using Data Set
- Implementing Transactions in ADO.NET

SqlDataAdapter Class:-

The SqlDataAdapter, serves as a bridge between a DataSet and SQL Server for retrieving and saving data.

The SqlDataAdapter provides this bridge by mapping Fill, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet, using the appropriate Transact-SQL statements against the data source.

The update is performed on a by-row basis. For every inserted, modified, and deleted row, the Update method determines the type of change that has been performed on it (Insert, Update and Delete). Depending on the type of change, the Insert, Update and Delete command template executes to propagate the modified row to the data source.

When the SqlDataAdapter fills a DataSet, it creates the necessary tables and columns for the returned data if they do not already exist.

However, primary key information is not included in the implicitly created schema unless the MissingSchemaAction property is set to AddWithKey.

You may also have the SqlDataAdapter create the schema of the DataSet, including primary key information, before filling it with data using FillSchema

The SqlDataAdapter also includes the SelectCommand, InsertCommand, DeleteCommand, UpdateCommand and TableMappings properties to facilitate the loading and updating of data.

DataSet :-

The DataSet is a memory-resident representation of data that provides consistent relational programming model regardless of the source of the data it contains.

A DataSet represents a complete set of data including the tables that contain columns, rows and constrain the data, as well as the relationships between the tables.

It is disconnected from the data source.

The principal class used by the ADO.NET disconnected model is System.Data.DataSet.

A DataSet object provides an in-memory cache of data. The data in a DataSet is held in a format that is independent of any underlying data source.

Type of DataSet

Typed Dataset :-

A typed dataset is a dataset that is first derived from the base DataSet class and then uses information from the Dataset Designer, which is stored in an .xsd file, to generate a new strongly-typed dataset class.

Information from the schema (tables, columns, and so on) is generated and compiled into this new dataset class as a set of first-class objects and properties.

Because a typed dataset inherits from the base DataSet class, the typed class assumes all of the functionality of the DataSet class and can be used with methods that take an instance of a DataSet class as a parameter.

There are some drawbacks to using strongly typed DataSet objects:

Using typed classes adds some overhead to code execution. If the strongly typed functionality isn't required, application performance can be improved slightly using an untyped DataSet.

The strongly typed DataSet will need to be regenerated when the data structure changes. Applications using the typed DataSet will need to be rebuilt using a reference to the new typed DataSet.

This can be especially significant in a multitier application or distributed where any clients that use the typed DataSets will have to be rebuilt using a reference to the updated version, even those that would not otherwise have been affected by the change if an untyped DataSet was used.

UnTyped DataSet :-

An UnTyped DataSet is an instance of class System.Data.DataSet.

It is binded with the tables at runtime and there no corresponding built-in schema.

You are not aware of the schema of the dataset at design time and there is no error checking facility at the design time as they are filled at run time when the code executes.

DataTable Class :-

A DataTable, which represents one table of in-memory relational data, can be created and used independently, or can be used by other .NET Framework objects, most commonly as a member of a DataSet.

You can create a DataTable object by using the DataTable constructor, or by passing constructor arguments to the Add method of the Tables property of the DataSet, which is a DataTableCollection.

DataColumn Class:-

The DataColumn is the fundamental building block for creating the schema of a DataTable.

You build the schema by adding one or more DataColumn objects to the DataColumnCollection. For more information, see Adding Columns to a DataTable.

Each DataColumn has a DataType property that determines the kind of data the DataColumn contains. For example, you can restrict the data type to integers, or strings, or decimals.

Because data that is contained by the DataTable is typically merged back into its original data source, you must match the data types to those in the data source.

DataRow Class :-

The DataRow class represents a single row of data in the DataTable.

The DataRow class can retrieve, update, insert, and delete a row of data from the DataTable.

Using the DataRow class, each column value for the row can be accessed.

The DataRow maintains the RowState property that is used by ADO.NET to track the changes that have been made
to a DataRow.

This property allows changed rows to be identified, and the appropriate update command to be used to update the data source with the changes.

A DataRow is created by calling the NewRow() method of a DataTable, a method that takes no arguments.

The DataTable supplies the schema, and the new DataRow is created with default or empty values for the fields.

RowState Property :-

The RowState property is used by ADO.NET to track the changes that have been made to a DataRow, which allows changes made to the data while disconnected to be updated back to the data source.

The RowState property indicates whether the row belongs to a table, and if it does, whether it's newly inserted, modified, deleted, or unchanged since it was loaded.

Data Relation:-

A DataRelation is used to relate two DataTable objects to each other through DataColumn objects. For example, in a Customer/Orders relationship, the Customers table is the parent and the Orders table is the child of the relationship. This is similar to a primary key/foreign key relationship. For more information, see Navigating DataRelations.

Relationships are created between matching columns in the parent and child tables. That is,
the DataType value for both columns must be identical.

Relationships can also cascade various changes from the parent DataRow to its child rows. To control how values are changed in child rows, add a ForeignKeyConstraint to the ConstraintCollection of the DataTable object. The ConstraintCollection determines what action to take when a value in a parent table is deleted or updated.

When a DataRelation is created, it first verifies that the relationship can be established. After it is added to the DataRelationCollection, the relationship is maintained by disallowing any changes that would invalidate it. Between the period when a DataRelationis created and added to the DataRelationCollection, it is possible for additional changes to be made to the parent or child rows. An exception is generated if this causes a relationship that is no longer valid.
Data View Class:-

A DataView enables you to create different views of the data stored in a DataTable, a capability that is often used in data-binding applications.

Using a DataView, you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression.

A DataView provides you with a dynamic view of a single set of data, much like a database view, to which you can apply different sorting and filtering criteria.

In some respects, a DataView is similar to a view in a database. However, a DataView is subject to several restrictions that do not apply for database views, including:

You cannot treat a DataView as a table.
You cannot use a DataView to provide a view of joined DataTable objects.
A DataView cannot exclude columns that exist in the source DataTable.

SqlCommandBuilder Class:-

The SqlDataAdapter does not automatically generate the Transact-SQL statements required to reconcile changes made to a DataSet with the associated instance of SQL Server.

However, you can create a SqlCommandBuilder object to automatically generate Transact-SQL statements for single-table updates if you set the SelectCommand property of the SqlDataAdapter. Then, any additional Transact-SQL statements that you do not set are generated by the SqlCommandBuilder.

The SqlCommandBuilder registers itself as a listener for RowUpdating events whenever you set the DataAdapter property. You can only associate one SqlDataAdapter or SqlCommandBuilder object with each other at one time.

To generate INSERT, UPDATE, or DELETE statements, the SqlCommandBuilder uses the SelectCommand property to retrieve a required set of metadata automatically. If you change the SelectCommand after the metadata has been retrieved, such as after the first update, you should call the RefreshSchema method to update the metadata.

The SelectCommand must also return at least one primary key or unique column. If none are present, an **InvalidOperation** exception is generated, and the commands are not generated.

Transaction :-

A transaction is a unit of work

You use transactions to ensure the consistency and integrity of a database

If a transaction is successful, all of the data modifications performed during the transaction are committed and made permanent

If an error occurs during a transaction, you can roll back the transaction to undo the data modifications that occurred during the transaction

Local transactions. A local transaction applies to a single data source, such as a database. It is common for these data sources to provide local transaction capabilities. Local transactions are controlled by the data source, and are efficient and easy to manage

Distributed transactions. A distributed transaction spans multiple data sources. Distributed transactions enable you to incorporate several distinct operations, which occur on different systems

Lesson 4:- XML Support in ADO.NET

This lesson will cover the following topics

- System.XML Namespace
- XML and Relational Data

Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data.

XML enables a new generation of Web-based data viewing and manipulation applications. XML is the universal language for data on the Web.

XML gives developers the power to deliver structured data from a wide variety of applications to the desktop for local computation and presentation.

It is used extensively in applications that are written by using general-purpose languages like C# or VB.NET.

The XML API provided through System.Xml namespace provide a comprehensive and integrated set of classes, allowing you to work with XML documents and data.

This namespace has a comprehensive set of XML classes for parsing, validation, and manipulation of XML data using readers, writers, and World Wide Web Consortium (W3C) DOM-compliant components.

It also covers XML Path Language (XPath) queries and Extensible Stylesheet Language Transformations (XSLT). You should note that the XML namespace allows you to get similar results in a number of different ways.

Classes available in System.XML Namespace

XMLReader - An abstract reader class that provides fast, non-cached XML data. XmlReader is forward-only.

XMLWriter - Represents an abstract writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

XMLTextReader – This class extends XmlReader and conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.

XMLTextWriter - This class extends the XMLWriter. Represents a writer that provides a fast, non-cached, forward-only way of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations

XMLNode – An abstract class that represents a single node in an XML document. XmlNode is the base class in the .NET implementation of the DOM.

XMLNodeReader – This class represents a reader that provides fast, non-cached forward only access to XML data in an XmlNode and extends the XMLReader. The XmlNodeReader has the ability to read an XML DOM subtree. This class does not support DTD or schema validation.

XMLDocument – Represents an XML document and it extends XMLNode. It provides a tree representation in memory of an XML document, enabling navigation and editing.

XMLDataDocument – This class extends the XMLDocument. Allows structured data to be stored, retrieved, and manipulated through a relational DataSet. Allows the mixing of XML and relational data in the same view.

XMLNodeList – To represents an ordered collection of nodes you can use XMLNodeList. The XmlNodeList collection reflects changes to the children of the node object that it was created from immediately in the nodes returned by the XmlNodeList properties and methods. XmlNodeList supports iteration and indexed access.

XMLNamedNodeMap – This class represents a collection of nodes that can be accessed by name or index.

XML and Relational Data:-

XML often has a relational structure as well as structured data, we can store this data centrally and providing different views on this data, either as XML or relationally as tables, columns and rows with relationships.

This disconnected store of data which, for example, could represent a business object in the middle tier that enforces business rules, could provide its data as XML to a browser via Extensible Stylesheet

Language Transformations (XSLT), across the Internet to another Web site, or to a local application via the relational tables. However, the data gets amended, you can update that data to a database on a transaction basis.

A DataSet represents an in-memory cache of data as a collection of tables and relationships between those tables. It is, in effect, a locally cached database. This provides a disconnected cache of data, like a message, that enables dealing with chunks of data.

The DataSet has no knowledge of where the data came from. It may have come from a file, a database connection, or from a stream. A DataSet provides a relational view onto this stored data.The XmlDataDocument provides XML APIs for accessing this in-memory cache of data, as well as supporting reading and writing XML.

The XmlDataDocument is a DataSet-aware object. Creation of an XmlDataDocument implicitly creates a DataSet (accessed as a property) that provides a relational view onto the data XML data.

This symbiotic relationship between these two objects provides a powerful technique for accessing data either relationally or as XML, irrespective of the mechanism by which the data was sourced.

With ADO.NET you can fill a DataSet from an XML stream or document. XmlDataDocument is a DataSet-aware object. Relationship between these two objects provides a powerful technique to access data either relationally or as XML, irrespective of the mechanism by which the data is sourced.

You can use the XML stream or document to supply to the DataSet either data, schema information, or both. The information supplied from the XML stream or document can be combined with existing data or schema information already present in the DataSet.

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. In an XML representation of a DataSet, the data is written in XML and the schema, if it is included inline in the representation, is written using the XML Schema definition language (XSD). XML and XML Schema provide a convenient format for transferring the contents of a DataSet to and from remote clients.

Dataset methods to work with XML Data

GetXml: Returns a string value of the XML representation of the data stored in the DataSet.

GetXmlSchema: Returns the XML Schema for the XML representation of the data stored in the DataSet.

ReadXml: Populates a DataSet object with the specified XML data read from a stream or a file.

ReadXmlSchema: Loads the specified XML schema information into the current DataSet object.

WriteXml: Writes the XML data, and optionally the schema, that represents the DataSet. Can write to a stream or a file.

WriteXmlSchema: Writes the string being the XML schema information for the DataSet .Can write to a stream or a file.

Lesson 5 :- Asynchronous Data Access

This lesson will cover the following topic

- Asynchronous Data Access in ADO.NET 4.5
- Sql Credential

Asynchronous Data Access in ADO.NET 4.5

Asynchronous programming is ability to execute several tasks on different threads at the same time. Through Asynchronous call the main thread never blocked and another thread calls the method and gives the output to the main thread.

There are situations in application development when you need to perform some time consuming operation. Synchronously performing these time consuming operations freezes the UI of the application and frustrates the end user.Asynchronously performing these time consuming operations provides following benefits:

- UI of the application remains responsive.
- Creates a better end user experience

In earlier version of .Net Framework, Data providers exposed Sql Command methods like BeginExecuteReader and EndExecuteReader.

BeginExecuteReader

Starts an asynchronous query to the data source that is expected to return some rows. The return value is a reference to an object that implements the IAsyncResult interface, in this case an instance of the SqlAsyncResult class, which is used to monitor and access the process as it runs and when it is complete.
Ex : async-result = command.BeginExecuteReader(callback, state)

It takes an AsyncCallback instance that specifies the callback routine to be executed when the process is complete, plus an Object that defines state information for the process.

EndExecuteReader

Once the command execution started by a BeginExecute Reader call has completed, this method is called to access the results. The single parameter is a reference to the SqlAsyncResult for the command, and the method returns a DataReader that references the rowset(s) returned by the query (in the same way the ExecuteReader method does for synchronous processes).
Ex: data-reader = command.EndExecuteReader(async-result).

Iasync Result type of implementation requires methods to be exposed in pairs of Begin Operation and End Operation. The Sql Command exposes End Execute Non Query pair for the asynchronous operation to complete. Iasync Result type is required while invoking the End Execute Non Query method which will block unless the process of executing the command is complete.

ADO.NET 4.5 introduces an overly simplified async programming model where you can perform asynchronous calls without using callbacks. The async and await modifiers in .NET Framework 4.5 are used for this purpose.

The following piece of code illustrates how you can implement asynchronous operations in ADO.Net.

```
using (SqlConnection connection = new SqlConnection(connectionString)

{
      await connection.OpenAsync();
           //Some code


      await reader.ReadAsync();
            //Other code
}
```

SqlCredential Class:-

SqlCredential provides a more secure way to specify the password for a login attempt using SQL Server Authentication.

SqlCredential is comprised of a user id and a password that will be used for SQL Server Authentication. The password in a SqlCredential object is of type SecureString.

SqlCredential cannot be inherited.

Use Credential to get or set a connection's SqlCredential object. Use ChangePassword to change the password for a SqlCredential object.

Lession 6:- Implementing Data Access Layer in ADO.Net

This lesson will cover the following topic

- Application Architecture – Tier based architecture
- Introduction to Data Access Layer
- DAL – Design considerations
- Data Access Layer Components
- Writing code for implementing DAL Components

Application Architecture:-

Application architecture is the Organizational Design of an entire software application, including all sub-components and external applications interchanges. A software application is a system designed to automate specific tasks in a logical manner to satisfy a set of requirements. Software applications rely on underlying operating systems and databases to store and perform tasks within the application.

It consists of the following layers

- Presentation Layer :- Application UI
- Business Logic Layer :- Business Rules and validations
- Data Access Layer :- Logic of accessing data

Data Access Layer:-

Being an application developer, you have to design many UIs which comprises of Web Pages, Web Forms etc. that perform database CRUD (Create, Read, Update and Delete) operations directly from the code associated with the page. This is a quick but wrong and not so suggested approach that often results in applications that are difficult to maintain, and worse, tend to be inconsistent in their execution of error handling and transactional integrity.

By separating and centralizing code for the activities associated with the specific tasks like data access, you gain the ability to reuse the code, not only within a single project, but across multiple projects, as well. This can greatly simplify maintaining application logic since code for specific tasks is easy to find and changes only need to be made in one place.

In an ideal world, Web pages, or any app for that matter, should be unaware of how the data from a DAL is retrieved. They should only make polite requests to the DAL. It should not matter whether the data comes from Oracle, SQL Server.

A correct approach to designing the data access layer will reduce development time and assist in maintenance of the data layer after the application is deployed. This part of the lesson briefly outlines an effective design approach for the data layer and the different important components of DAL.

A typical Data Access Layer consists of following main components.

Data access logic component:-

Data access components abstract the logic necessary to access your underlying data stores.

Doing so centralizes the data access functionality, which makes the application easier to configure and maintain.

Data helpers/utilities:-

Helper functions and utilities assist in data manipulation, data transformation, and data access within the layer.

They consist of specialized libraries and/or custom routines especially designed to maximize data access performance.

DAL –Design Consideration

Security – The security plays major role in designing DAL. We can design secure data access layer by creating separate database accounts for your applications with no direct permissions on tables within the database. This can be done by providing separate stored procedures for each Select, Update, Insert and Delete action that can be performed on an entity

Data Integrity – We always have databases designed with all the necessary constraints to ensure the integrity or correctness of the data. However, we certainly don't want to wait until we submit the data to the database to find out that it doesn't meet the constraints, like referential integrity. Therefore, our DAL logic should be configured to reflect these same constraints and raise errors before we get to the database.

Transaction Integrity - If you are inserting multiple pieces of data in one or more tables, you may need to ensure that all the data is successfully saved. A DAL should support the ability to rollback a series of database transactions if any one of them fails.

Concurrency Control - When many people attempt to modify data in a database at the same time, a system of controls must be implemented so that modifications made by one person do not adversely affect those of another person. This is called concurrency control. We can handle concurrency in three different ways, first, we do nothing, allowing the changes of the last user to overwrite the changes of the first without warning. This is typically known as the "last in wins" scenario. Secondly, you can implement optimistic concurrency control when users do not lock data when they read it. When a user updates data, the system checks to see if another user changed the data after it was read. If another user updated the data, an error is raised. Typically, the user receiving the error rolls back the transaction and starts over. In the third option, you take a pessimistic stand. In this scenario, you predict that it is likely that two users will be vying for the same record so you place a lock on the record when the first person accesses it to prevent any others from accessing it until the first edit is complete.

Error Handling - Since we want our DAL consumers to be unaware of the inner workings of our data retrieval, we certainly don't want unhandled SQL Server exceptions bubbling up to the presentation layer. Instead, the DAL should be providing more useful custom exceptions about the nature of the error. The DAL needs to present these exceptions to the consumer so that the consumer can decide whether it is OK to proceed, try again, or simply give up.

Maintainability – No application code is static or permanent. According to requirements we have to keep on changing application logic/code to meet challenges. Most of the times we needed to change or remove a field in a table and for that to happen we have to search through all our code to find every place that it is being referenced. Such types are changes are always required to be made in our application code. By centralizing your code, you limit the number of places this can happen. Also, by using data objects that provide hard-coded properties that match fields for our entities, the application will not compile if one is removed.

Advantages of implementing DAL

Applications using a data access layer can be either database server dependent or independent. If the data access layer supports multiple database types, the application becomes able to use whatever databases the DAL can talk to.

This layer encapsulates and compartmentalizes all our data access code within nice, clean components that interact with our database.

The DAL might return a reference to an object complete with its attributes instead of a row of fields from a database table. With this benefit we can create client modules with a higher level of abstraction.

Instead of using commands such as insert, delete, and update to access a specific table in a database, a class and a few stored procedures could be created in the database. The procedures would be called from a method inside the class, which would return an object containing the requested values.

Business logic methods from an application can be mapped to the Data Access Layer. For example, instead of making a query into a database to fetch all users from several tables the application can call a single method from a DAL which abstracts those database calls.