**Validaciones de datos y Acceso a datos con Net Framework**

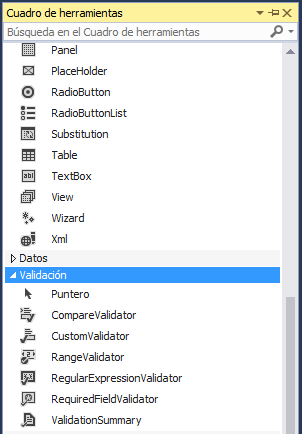
## **Validaciones de datos en el cliente**

Supongamos que tenemos un TextBox donde deseamos que el usuario ingrese un valor. Supongamos también que dicho valor no puede superar los 30 caracteres pero tampoco puede estar en blanco.

Los controles del tipo TextBox poseen la propiedad **MaxLength** que permiten regular la cantidad de caracteres ingresados pero no hay forma de impedir que el usuario no ingrese ninguno. En este caso deberíamos efectuar un **PostBack**, preguntar si la longitud es 0 y luego devolver un mensaje de error al cliente.

Si bien esto funciona, tiene un costo en código y en velocidad de respuesta: el envío y posterior devolución del formulario. Para ahorrar dicho viaje al servidor deberíamos validar el **TextBox** del lado del cliente, utilizando **JavaScript**.

A fin de que el desarrollador no deba escribir dichas validaciones existe una familia de controles llamadas **Validators** que generan el código **JavaScript** necesario. Dichos controles son:



### **RequiredFieldValidator**

### Valida si hemos ingresado un valor en un TextBox.

### **CompareValidator** Compara el valor de 2 controles TextBox.

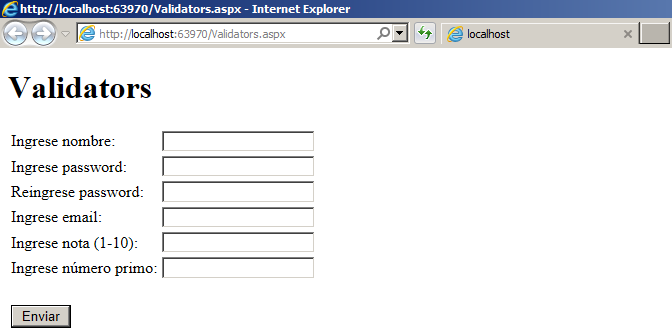
### **RangeValidator** Verifica si el valor ingresado está dentro del rango estipulado.

### **RegularExpressionValidator** Valida lo ingresado según una expresión regular.

### **CustomValidator** Valida según una función JavaScript escrita por nosotros.

### **ValidationSummary** Agrupa en una zona todos los mensajes de error generados por los demás controles Validators.

En el proyecto **Validators** de la sección de descargas, tenemos un ejemplo de uso de estos controles. Podrá hacer el ejercicio correspondiente de la sección de laboratorio de esta clase.

****

**Ado.net**

**ADO.NET** es un conjunto de clases que exponen servicios de acceso a datos para programadores de .NET Framework. ADO.NET ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de datos distribuidas. Es una librería de clases que constituye una parte integral de .NET Framework y proporciona acceso a datos relacionales, XML y de aplicaciones. También hoy en día puede conectarse a bases de datos **NOSql** (**N**ot **O**nly **Sql**), que son bases de datos de estructura no relacional.

ADO.NET satisface diversas necesidades de desarrollo, como la creación de clientes front-end de base de datos y objetos empresariales de nivel medio (lógica de negocio), que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet.

## <https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/>

### 

### **Componentes de ADO.NET**

Los dos componentes principales de ADO.NET para el acceso a los datos y su manipulación son los proveedores de datos .NET Framework y DataSet.

**Nota Importante**: si bien mencionaremos básicos de DataSet, DataAdapter y DataTable, sólo se recomienda su uso en caso en forma esporádica y para tareas de administración de la base de datos que uno deba hacer desde código C#, por ejemplo, necesitar recuperar parte del esquema de la base de datos (parte de la estructura de la base de datos), o necesitar crear con permisos adecuados una tabla nueva en una base de datos. **Nos centraremos en esta clase y en la siguiente en los objetos *Connection*, *Command*, *Parameter* y *DataReader*, adecuados para una aplicación en capas que consume datos.**

En el desarrollo con arquitectura en capas o con APIs de acceso a datos (microservicios), **es recomendable crear la lista de entidades correspondientes a la lógica de negocio de la aplicación**, que recuperen los datos desde un DataReader de ADO.Net en la capa de acceso a datos y los pasa a la capa de negocio en una lista de objetos, de esta forma la capa de negocio queda totalmente libre de los objetos ADO.Net, y no estaría perdiendo performance o haciendo lenta  la aplicación o API, dado que el trabajo con DataSet/DataTable es pesado y genera sobrecargas.

## **Proveedores de datos .NET Framework**

Los proveedores de datos .NET Framework son componentes diseñados explícitamente para la manipulación de datos y el acceso rápido a datos de solo lectura y solo avance.

El objeto **Connection** proporciona conectividad a un origen de datos.

El objeto **Command** permite tener acceso a comandos de base de datos para devolver datos, modificar datos, ejecutar procedimientos almacenados y enviar o recuperar información sobre parámetros.

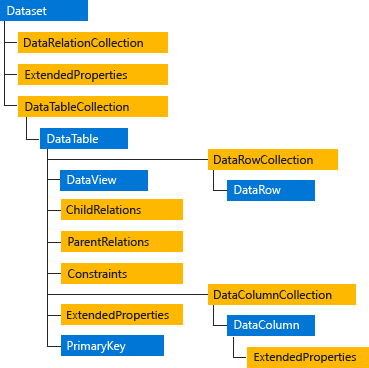
El objeto **DataReader** proporciona un flujo de datos de alto rendimiento desde el origen de datos. Por último, el objeto **DataAdapter** proporciona el puente entre el objeto **DataSet** y el origen de datos. **DataAdapter** utiliza objetos **Command** para ejecutar comandos SQL en el origen de datos tanto para cargar **DataSet** con datos y reconciliar en el origen de datos los cambios aplicados a los datos incluidos en el **DataSet**.

### **DataSet**

El DataSet de ADO.NET está expresamente diseñado para el acceso a datos independientemente del origen de datos. Como resultado, se puede utilizar con múltiples y distintos orígenes de datos, con datos XML/JSON o para administrar datos locales de la aplicación.

DataSet contiene una colección de uno o más objetos DataTable formados por filas y columnas de datos, así como información sobre claves principales, claves externas, restricciones y de relación relacionada con los datos incluidos en los objetos DataTable.

En el diagrama siguiente se ilustra la relación entre un proveedor de datos .NET Framework y un DataSet.



Arquitectura de ADO.NET

**Nota**: La serialización de un DataSet a formato JSON es un tema avanzado pero alternativo y conveniente para manejo de datos. Recomendamos este link: [Serialize a DataSet](https://www.newtonsoft.com/json/help/html/SerializeDataSet.htm)

### 

### **Elegir un DataReader o un DataSet**

A la hora de decidir si su aplicación debe utilizar un DataReader o un DataSet, debe tener en cuenta el tipo de funcionalidad que su aplicación requiere. Use un DataSet para hacer lo siguiente:

* Almacene datos en la memoria caché de la aplicación para poder manipularlos. Si solamente necesita leer los resultados de una consulta, el DataReader es la mejor elección, es más ágil y rápido.
* Utilizar datos de forma remota entre un nivel y otro o desde un servicio Web XML/JSON.
* Interactuar con datos dinámicamente, por ejemplo para enlazar con un control de Windows Forms o para combinar y relacionar datos procedentes de varios orígenes de datos.
* Realizar procesamientos exhaustivos de datos sin necesidad de tener una conexión abierta con el origen de datos, lo que libera la conexión para que la utilicen otros clientes.

Si no necesita la funcionalidad proporcionada por el DataSet, **puede mejorar el rendimiento de su aplicación si utiliza el DataReader** para devolver sus datos de solo avance y de solo lectura. Aunque DataAdapter utiliza DataReader para rellenar el contenido de un DataSet , al utilizar el DataReader puede mejorar el rendimiento porque no usará la memoria que utilizará el DataSet, además de evitar el procesamiento necesario para crear y rellenar el contenido de DataSet.

Si llegara a necesitar más información sobre DataSet, se recomienda este link:

[Objetos DataSet, DataTable y DataView](https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/dataset-datatable-dataview/)

### **LINQ to DataSet**

Proporciona capacidades de consulta y comprobación de tipo en tiempo de compilación de los datos almacenados en caché de un objeto DataSet. Permite escribir consultas en uno de los lenguajes de desarrollo de .NET Framework.

## 

### **LINQ to SQL**

Admite consultas en un modelo de objetos asignado a las estructuras de datos de una base de datos relacional sin utilizar un modelo conceptual intermedio. Cada tabla se representa mediante una clase distinta, acoplando de manera precisa el modelo de objetos al esquema de la base de datos relacional. LINQ to SQL convierte las consultas de Language-Integrated Query del modelo de objetos a Transact-SQL y las envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados, LINQ to SQL los vuelve a traducir a objetos.

### **ADO.NET Entity Framework**

Está diseñado para permitir que los desarrolladores creen aplicaciones de acceso a los datos programando en un modelo de aplicación conceptual en lugar de programar directamente en un esquema de almacenamiento relacional. El objetivo es reducir la cantidad de código y mantenimiento que se necesita para las aplicaciones orientadas a datos.

#### 

### **Servicios de datos de WCF**

Describe cómo se usa Servicios de datos de WCF para implementar servicios de datos en web o en una intranet. Los datos se estructuran como entidades y relaciones de acuerdo a las especificaciones de Entity Data Model. Los datos implementados en este modelo se pueden direccionar mediante el protocolo HTTP estándar.

### **XML y ADO.NET**

ADO.NET aprovecha la eficacia de XML para proporcionar acceso a datos sin conexión. ADO.NET fue diseñado teniendo en cuenta las clases de XML incluidas en .NET Framework; ambos son componentes de una única arquitectura.

ADO.NET y las clases de XML incluidas en .NET Framework convergen en el objeto DataSet. DataSet se puede rellenar con datos procedentes de un origen XML, ya sea éste un archivo o una secuencia XML. DataSet se puede escribir como XML conforme al consorcio World Wide Web (W3C), que incluye su esquema como esquema lenguaje de definición de esquemas XML, independientemente del origen de los datos incluidos en DataSet. Puesto que el formato nativo de serialización del DataSet es XML, es un medio excelente para mover datos de un nivel a otro, por lo que DataSet es idóneo para usar datos y contextos de esquemas de interacción remota desde y hacia un servicio Web XML.

# 

## **Uso básico de Sql Managment Studio**

**SQL Server Management Studio** (SSMS) es un entorno integrado para obtener acceso, configurar, administrar y desarrollar todos los componentes de SQL Server. SSMS combina un amplio grupo de herramientas gráficas con una serie de editores de script enriquecidos que permiten a desarrolladores y administradores de todos los niveles obtener acceso SQL Server.

SSMS combina las características del Administrador corporativo, el Analizador de consultas y Analysis Manager, herramientas incluidas en versiones anteriores de SQL Server, en un único entorno. Además, SSMS funciona con todos los componentes de SQL Server, como Reporting Services e Integration Services. De este modo, los desarrolladores pueden disfrutar de una experiencia familiar y los administradores de bases de datos disponen de una herramienta única y completa que combina herramientas gráficas fáciles de usar con funciones avanzadas de scripting.

Es un frontend que permite conectarse a bases de datos de la familia SQL Server. De manera de poder trabajar y probar más fácilmente sus consultas. De todas maneras Visual Studio posee un entorno similar para poder ejecutar dichas queries.

## 

## **Uso de cadenas de conexión (connection Strings)**

Una cadena de conexión contiene información de inicialización que se transfiere como un parámetro desde un proveedor de datos a un origen de datos. La sintaxis depende del proveedor de datos y la cadena de conexión se analiza mientras se intenta abrir una conexión. Las cadenas de conexión que usa Entity Framework contienen la información que se emplea para conectar con el proveedor de datos ADO.NET subyacente que Entity Framework admite. También contienen información sobre los archivos del modelo y de asignación necesarios.

El proveedor de EntityClient utiliza la cadena de conexión al obtener acceso a los metadatos del modelo y de asignación y al conectar con el origen de datos. Se puede obtener acceso a la cadena de conexión o establecerse a través de la propiedad ConnectionString de EntityConnection. La clase EntityConnectionStringBuilder se puede utilizar para construir mediante programación los parámetros de la cadena de conexión o tener acceso a ellos.

Las herramientas de Entity Data Model generan una cadena de conexión que se almacena en el archivo de configuración de la aplicación. ObjectContext recupera esta información de conexión automáticamente al crear consultas de objetos. Se puede tener acceso al elemento EntityConnection que usa una instancia de ObjectContext desde la propiedad Connection.

## **Parámetros de la cadena de conexión**

El formato de una cadena de conexión es una lista de pares de parámetros de clave y valor delimitados por punto y coma:

keyword1=value; keyword2=value;

El signo igual (=) asocia cada palabra clave a su valor. Las palabras clave no distinguen entre mayúsculas y minúsculas y los espacios entre los pares clave-valor se omiten. Sin embargo, los valores pueden distinguir entre mayúsculas y minúsculas, en función del origen de datos. Los valores que contengan un punto y coma, caracteres de comilla sencilla o caracteres de comilla doble deben colocarse entre comillas dobles.

Para detalles de cadenas de conexión de todos los orígenes de datos relacionales y no relacionales, consulte el siguiente link:

<https://www.connectionstrings.com/>

### **Ejemplo de Connection String con SQL Server**

Server=myServerAddress;Database=myDataBase;User Id=myUsername;

Password=myPassword;

### **Ejemplo de Connection String con Oracle**

Data Source=MyOracleDB;User Id=myUsername;Password=myPassword;

Integrated Security=no;

En caso de No Usar Archivos TNS, se puede utilizar su mismo formato:

SERVER=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=MyHost)(PORT=MyPort))(CONNECT\_DATA=(SERVICE\_NAME=MyOracleSID)));

uid=myUsername;pwd=myPassword;

### **Ejemplo de Connection String con MySQL**

**Estándar:**

Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;

### **Especificando Puerto TCP/IP[:](https://www.connectionstrings.com/mysql-connector-net-mysqlconnection/specifying-tcp-port/)**

Server=myServerAddress;Port=1234;Database=myDataBase;Uid=myUsername; Pwd=myPassword;

## **Uso de acceso a datos conectado - Datareader**

La recuperación de datos mediante DataReader implica crear una instancia del objeto Command y de un DataReader a continuación, para lo cual se llama a Command.ExecuteReader a fin de recuperar filas de un origen de datos. En el ejemplo siguiente se muestra cómo se utiliza un DataReader, donde reader representa un DataReader válido y command representa un objeto Command válido.

reader = command.ExecuteReader();

Puede utilizar el método Read del objeto DataReader para obtener una fila a partir de los resultados de una consulta. Para tener acceso a cada columna de la fila devuelta, puede pasar a DataReader el nombre o referencia numérica de la columna en cuestión. Sin embargo, el mejor rendimiento se logra con los métodos que ofrece DataReader y que permiten tener acceso a los valores de las columnas en sus tipos de datos nativos (GetDateTime, GetDouble, GetGuid, GetInt32, etc.). Para obtener una lista de métodos de descriptor de acceso con tipo para DataReadersde proveedores de datos específicos, vea las secciones OleDbDataReader y SqlDataReader. Si se utilizan los métodos de descriptor de acceso con tipo, dando por supuesto que se conoce el tipo de datos subyacentes, se reduce el número de conversiones de tipo necesarias para recuperar el valor de una columna.

En el ejemplo de código siguiente se repite por un objeto DataReader y se devuelven dos columnas de cada fila:

static void HasRows(SqlConnection connection)  
{  
 using (connection)  
 {  
 SqlCommand command = new SqlCommand(  
 "SELECT CategoryID, CategoryName FROM Categories;",  
 connection);  
 connection.Open();  
  
 SqlDataReader reader = command.ExecuteReader();  
  
 if (reader.HasRows)  
 {  
 while (reader.Read())  
 {  
 Console.WriteLine("{0}\t{1}", reader.GetInt32(0),  
 reader.GetString(1));  
 }  
 }  
 else  
 {  
 Console.WriteLine("No rows found.");  
 }  
 reader.Close();  
 }  
}

**DataReader** proporciona un flujo de datos sin búfer que permite a la lógica de los procedimientos procesar eficazmente y de forma secuencial los resultados procedentes de un origen de datos. **DataReader** es la mejor opción cuando se trata de recuperar grandes cantidades de datos, ya que éstos no se almacenan en la memoria caché.

## 

### **Cerrar el DataReader**

Siempre debe llamar al método **Close** cuando haya terminado de utilizar el objeto **DataReader**.

Si **Command** contiene parámetros de salida o valores devueltos, éstos no estarán disponibles hasta que se cierre el **DataReader**.

Tenga en cuenta que mientras está abierto un **DataReader**, ese **DataReader** utiliza de forma exclusiva el objeto **Connection**. No se podrá ejecutar ningún comando para el objeto **Connection** hasta que se cierre el **DataReader** original, incluida la creación de otro **DataReader**.

## **Elementos que necesita ADO.NET para ejecutar una consulta**

**SQLCommand**: Contiene el comando a ejecutar en la base de datos.

**SQLConnection**: Contiene la cadena de conexión y administra el estado de apertura de dichas conexiones.

**SQLParameters**: En caso de utilizar un Stored Procedure podrá pasar los parámetros usando este tipo de dato.

### **Pasos para ejecutar la Lectura de datos**

1. Crear un nuevo objeto **SQLCommand** (en caso de que el proveedor de datos fuese otro, por ejemplo Oracle deberá utilizar la clase correspondiente, por ejemplo OracleCommand).
2. Crear un objeto **SQLConnection** y asignar al mismo la **connectionString**.
3. Asignar el objeto **Connection** a la propiedad correspondiente en el Objeto **Command**.
4. Abrir la conexión en el Objeto **Connection** en caso de que utilice **DataReader**, si en su lugar usa **DataSets** el comando **Fill** se encargará de la conexión automáticamente.
5. Asignar al **SQLCommand** el **CommandType** y **CommandText**. Si no asigna **CommandType** el tipo por defecto será “**Text**”.
6. Sobre el objeto **Command** ejecutar la consulta colocando el valor devuelto en un objeto resultado.
7. En caso de Ejecutar **DataReader** deberá efectuar un **Read()** para cada registro. Para esto, el comando **Read()** de un **ExecuteReader()** devolverá Verdadero cuando pueda efectuar una lectura y false cuando no pueda. Es por esto que se lo coloca en un bloque while.

### **Comandos Execute de ADO.NET**

**ExecuteNonQuery**:

Ejecuta una instrucción de Transact-SQL en la conexión y devuelve el número de filas afectadas.

**ExecuteScalar**:

Ejecuta la consulta y devuelve la primera columna de la primera fila del conjunto de resultados devuelto por la consulta. Las demás columnas o filas no se tienen en cuenta.

**ExecuteReader**:

Envía la propiedad CommandText a Connection y crea un objeto SqlDataReader.

**ExecuteXMLReader**:

Envía CommandText a Connection y crea un objeto XmlReader.

### **Comando Prepare y ejecución de Stored Procedures**

El comando **Prepare** crea una versión preparada del comando en una instancia de SQL Server:

* Debería usarse cuando un comando se ejecuta múltiples veces.
* Origina una sobrecarga inicial debida a la creación de un procedimiento almacenado en el SGBD para la ejecución del comando.
* Se rentabiliza en las siguientes ejecuciones del comando.
* La ejecución de **Command.Prepare()** necesita una conexión abierta y disponible.

private static void SqlCommandPrepareEx(string connectionString)  
{  
 using (SqlConnection connection = new SqlConnection(connectionString))  
 {  
 connection.Open();  
 SqlCommand command = new SqlCommand(null, connection);  
  
 // Create and prepare an SQL statement.  
 command.CommandText =  
 "INSERT INTO Region (RegionID, RegionDescription) " +  
 "VALUES (@id, @desc)";  
 SqlParameter idParam = new SqlParameter("@id", SqlDbType.Int, 0);  
 SqlParameter descParam =   
 new SqlParameter("@desc", SqlDbType.Text, 100);  
 idParam.Value = 20;  
 descParam.Value = "First Region";  
 command.Parameters.Add(idParam);  
 command.Parameters.Add(descParam);  
  
 // Call Prepare after setting the Commandtext and Parameters.  
 command.Prepare();  
 command.ExecuteNonQuery();  
  
 // Change parameter values and call ExecuteNonQuery.  
 command.Parameters[0].Value = 21;  
 command.Parameters[1].Value = "Second Region";  
 command.ExecuteNonQuery();  
 }  
}

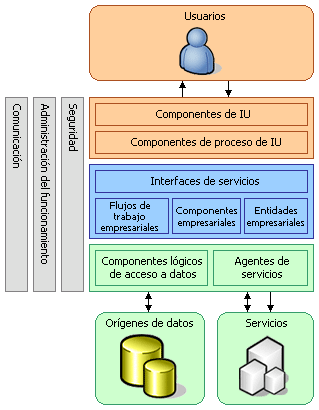
### **Buenas Prácticas**

Siempre que use ADO.NET conectado, se recomienda utilizar un bloque **Using** tal cual se ve en el último ejemplo de código. Esto es, debido a que el **GC** (Garbage collector) elimina explícitamente todo objeto declarado dentro del bloque using una vez que se ejecuta liberando así memoria y recursos. Por ejemplo el caso de las conexiones abiertas que se omiten cerrar queda resuelto mediante un bloque using.

## 

## **Aplicación en 3 capas y pasaje de información entre capas**

El diseño de aplicaciones distribuidas no es una tarea sencilla. Es necesario tomar un gran número de decisiones a nivel de arquitectura, diseño e implementación. Estas decisiones tendrán un impacto en las "capacidades" de la aplicación (seguridad, escalabilidad, disponibilidad y mantenimiento, entre otras), así como en la arquitectura, el diseño y la implementación de la infraestructura de destino. La guía le ayudará a comprender las distintas opciones que se presentan a la hora de diseñar las capas de una aplicación distribuida; estas opciones se presentan como un conjunto de capas de componentes que se podrán utilizar para modelar la aplicación. En la figura 1 se muestran las capas de los componentes lógicos que este documento utiliza para estructurar sus instrucciones.



Capas de componentes de servicios y aplicaciones distribuidas creadas con .NET

La programación en capas es una arquitectura cliente-servidor en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, solo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación sería el modelo de interconexión de sistemas abiertos.

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suelen usar las arquitecturas multinivel o Programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

La capa de negocio contiene la lógica principal de procesamiento de datos dentro de nuestra aplicación. Se comunica con la capa de presentación para obtener las entradas del usuario y presentar la información resultante, así como la capa de acceso a datos o directamente con servicios para realizar sus operaciones.

### **GUI - Graphical User Interface - Capa de presentación o interfaz gráfica de Usuario**

Se refiere a la presentación del programa frente al usuario, esta presentación debe cumplir su propósito con el usuario final, una presentación fácil de usar y amigable. También las interfaces deben ser consistentes con la información dentro del software (Por ejemplo; en los formularios no debe haber más que lo necesario), tomar en cuenta los requerimientos del usuario, la capa de presentación va de la mano con capa de la lógica de negocio.

### **BLL - Business Logic Layer - Capa Lógica de Negocios**

En esta capa es donde se encuentran los programas que son ejecutados, recibe las peticiones del usuario y posteriormente envía las respuestas tras el proceso. Esta capa es muy importantes pues es donde se establecen todas aquellas reglas que se tendrán que cumplir, decía anteriormente que la capa de presentación tiene comunicación con la capa de lógica de negocio ya que se tienen que comunicar para recibir las solicitudes y presentar los resultados.

### **DAL - Data Access Layer - Capa de Acceso a datos**

Esta capa es la que se encarga de hacer las transacciones con la base de datos y con otros sistemas para descargar o insertar información al sistema. La consistencia en los datos es sumamente importante, es decir, los datos que se ingresan o insertan deben ser precisos y conscientes. Aquí definimos las consultas que vamos a realizar en la base de datos, o consultas para reporteo. La comunicación de esta capa con la capa de lógica de negocio se refiere a que la capa de datos es la que le enviara información a la capa de negocio para que sea procesada e ingresada en objetos según sea necesario (encapsulamiento).

**Recomendaciones**:

Para profundizar y complementar los contenidos expuestos en esta clase, se sugiere abordar los siguientes cursos:

* [Introducción a Bases de Datos y SQL](https://www.educacionit.com/curso-de-introduccion-sql)
* [Fundamentos de Programación con SQL Server 2017](https://www.educacionit.com/curso-de-sql-server-writingqueries)
* [Entity Framework y LINQ](https://www.educacionit.com/curso-de-linq)