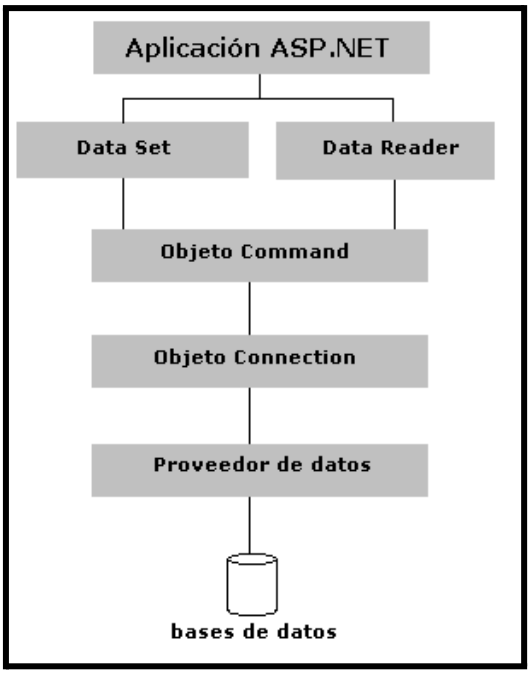
Recuerda:

Este es el esquema de objetos utilizado en ADO.NET:



El procedimiento general para acceder a bases de datos varía dependiendo de si utilizamos un DataSet o un DataReader.

Usando DataSets

1. Conectar a la base de datos utilizando el objeto **Connection.**
2. Almacenar la consulta de la base de datos en un **DataAdapter.**
3. Poblar un DataSet desde el DataAdapter utilizando **Fill**.
4. Crear una nueva DataView para la tabla deseada.
5. Enlazar los datos a un control.

Usando DataReaders

1. Conectar a la base de datos utilizando el objeto **Connection**.
2. Abrir la conexión con el método **Open**.
3. Almacenar la consulta de la base de datos en objetos **Command.**
4. Poblar un **DataReader** desde el Command utilizando el método ExecuteReader.
5. Invocar los métodos Read y Get del DataReader para leer datos.
6. Visualizar manualmente los datos.
7. Cerrar DataReader y conexión. **Close**.

Invocar procedimientos almacenados con ADO.NET

Acceder y manipular datos directamente en una base de datos desde una aplicación puede restar eficacia y crear riesgos de seguridad. Una forma de mejorar la eficacia y seguridad del acceso a las bases de datos es crear procedimientos almacenados en el servidor de la base de datos, e invocar estos procedimientos desde nuestra aplicación.

Un procedimiento almacenado es un procedimiento escrito por un desarrollador de aplicaciones para bases de datos que está asociado a una base de datos concreta. Más tarde, otras aplicaciones podrán invocar estos procedimientos para acceder y manipular datos de la base de datos en lugar de tener que escribir las instrucciones SQL incrustadas en el código de la aplicación.

Los procedimientos almacenados están formados por secuencias de instrucciones Transact-SQL, y funcionan de modo similar a los procedimientos de una aplicación Windows en cuanto a que las instrucciones se invocan por su nombre, y pueden tener tanto parámetros de entrada como de salida.

Es más fácil, más eficaz y más seguro utilizar un procedimiento almacenado que escribir el código Transact-SQL requerido para llevar a cabo la acción que éste realiza. Invocar un procedimiento almacenado no requiere que entendamos cómo está diseñada la base de datos, y sólo se accede a ella mediante un procedimiento que con toda seguridad ya estará probado.

¿Por qué utilizar procedimientos almacenados?

- Programación modular

Los procedimientos almacenados son ejemplos clásicos de programación modular. Creamos el procedimiento una vez, lo probamos una vez, lo almacenamos en el servidor de la base de datos, y lo invocamos varias veces desde múltiples aplicaciones. Las actualizaciones o cambios a la base de datos permanecen ocultas para todas las aplicaciones que acceden mediante el procedimiento almacenado.

- Distribución del trabajo

Un desarrollador especialista en la programación de bases de datos puede crear los procedimientos almacenados, mientras que otros desarrolladores pueden crear paralelamente las aplicaciones Windows/Web/móvil que utilizarán dicho procedimiento. Esta distribución del trabajo permite que cada desarrollador se concentre en su propia especialidad, y cumpla con sus propios plazos.

- Incremento de la seguridad de la base de datos

El uso de procedimientos almacenados proporciona una mayor seguridad para una base de datos al limitar el acceso directo. Únicamente acceden directamente a la base de datos los procedimientos almacenados probados que han sido desarrollados por el propietario de la base de datos. Debido a que las demás aplicaciones y otros desarrolladores no acceden directamente a la base de datos, el riesgo de daño accidental de la estructura o del contenido de la base de datos es mínimo.

Utilizar instrucciones SQL o Transact-SQL directamente en el código de las aplicaciones Windows/Web supone un riesgo para la seguridad, ya que las instrucciones pueden dar información a un hacker sobre la base de datos y su estructura.

➢ Ejecución más rápida

Imagina que tienes el siguiente código:

```
SELECT * FROM Grabaciones
WHERE CodGrabacion >= '060' AND CodGrabacion <= '090'
ORDER BY Titulo
```

Si se pasa esta sentencia SQL a un servidor SQL Server, mediante el método ExecuteReader de un objeto SqlCommand (o cualquier método de este tipo), lo que ocurre es que el servidor tiene que compilar el código antes de ejecutarlo, de forma parecida a como las aplicaciones .NET se tienen que compilar antes de que se puedan ejecutar. Esta compilación resta tiempo al servidor SQL Server, por lo que resulta bastante obvio deducir que, si se reduce el número de compilaciones que tenga que realizar el servidor SQL Server, se incrementa el rendimiento de la base de datos (compara la velocidad de ejecución de una aplicación compilada, en relación con una con código interpretado).

➢ Reducción del tráfico de red

En ocasiones, una operación que requiere cientos de líneas de código Transact-SQL puede realizarse mediante una única instrucción que invoque un procedimiento almacenado. Enviar una llamada a través de la red, en lugar de cientos de líneas de código, reducirá el tráfico de red.

➢ Proporciona flexibilidad

Si el acceso a la base de datos se realiza a través de procedimientos almacenados, el desarrollador de la base de datos puede cambiar la estructura de la base de datos sin romper las aplicaciones que los utilizan. Esta protección permite la continua mejora de la base de datos sin poner en riesgo el resto del sistema.

Para el tema que nos compete, invocar procedimientos almacenados *(Stored Procedure, SP*), clasificamos los SP en tres **tipos:**

1. Procedimientos almacenados que devuelven registros

Los procedimientos almacenados que devuelven registros se utilizan para encontrar registros específicos, clasificar y filtrar esos registros, y devolver el resultado de las operaciones de búsqueda, clasificación y filtrado en un DataSet (clase desconectada) o en un DataReader (clase conectada).

Estos procedimientos almacenados se basan en instrucciones SELECT de SQL. Un ejemplo de un procedimiento almacenado que devuelve registros es una petición de la cantidad, fecha y receptor de los tres últimos movimientos procesados en una cuenta bancaria. Estos datos podrían cargarse en un objeto DataSet para su posterior procesamiento, o mostrarse directamente al usuario en un control listBox.

2. Procedimientos almacenados que retornan un valor, también denominados procedimientos almacenados escalares

Los procedimientos almacenados que retornan un valor se utilizan para ejecutar un comando o función de la base de datos que devuelve un único valor. Un ejemplo de procedimiento almacenado que retorna un valor es el desarrollado para devolver el valor total de los tres últimos movimientos que se han procesado en una cuenta bancaria.

3. Los procedimientos almacenados de acción

Los procedimientos almacenados de acción se utilizan para realizar algunas funciones en la base de datos, pero no devuelven un registro o un valor. Estas funciones de la base de datos pueden incluir actualizar, editar o modificar los datos. Un ejemplo de un procedimiento almacenado de acción es una petición para actualizar la dirección de correo en la base de datos de clientes de una compañía.

El primer paso para utilizar un procedimiento almacenado es identificar el tipo y el nombre del mismo. Podemos utilizar un objeto DataAdapter o un objeto DataReader para invocar los tres tipos de procedimientos almacenados. Los pasos a seguir para ejecutar el procedimiento almacenado, variarán dependiendo del tipo de procedimiento almacenado que invoquemos:

1. Procedimientos almacenados que devuelven registros

Cuando invocamos un procedimiento que devuelve un conjunto de registros, necesitamos almacenar ese conjunto de registros en un DataSet, o por ejemplo directamente en un control enlazado a lista utilizando un DataReader. Si deseamos utilizar un DataSet, debemos utilizar un DataAdapter y el método Fill. Si deseamos utilizar un DataReader, debemos utilizar un objeto Command y el método ExecuteReader.

2. Procedimientos almacenados que retornan un valor

Cuando invocamos un procedimiento almacenado que devuelve un valor, invocamos el método ExecuteScalar del objeto Command, y guardamos el resultado en una variable del tipo de datos apropiado.

3. Procedimientos almacenados que realizan una acción

Cuando invocamos un procedimiento almacenado que realiza alguna acción sobre la base de datos pero no devuelve un conjunto de registros o un valor, utilizamos el método ExecuteNonQuery del objeto Command.

Ejemplo:

Invocar el procedimiento almacenado ProductCategoryList. El procedimiento devuelve una lista de IDs y nombres de categorías de la tabla Categories y tiene la siguiente forma:

```
Procedure ListaCategorias
As
SELECT CategoryID,CategoryName
FROM Categories
```

Este primer código utiliza un objeto **DataAdapter *(clases desconectadas)*** para invocar el procedimiento almacenado:

```
Dim da as New SqlDataAdapter()
da.SelectCommand = New SqlCommand()
da.SelectCommand.Connection = cn
da.SelectCommand.CommandText = "ListaCategorias"
da.SelectCommand.CommandType = CommandType.StoredProcedure

da.Fill(ds, "Categories")
```

Recuerda que también podemos establecer directamente la conexión y el texto del comando cuando creamos el objeto SqlDataAdapter. En tal caso la fora del código es:

```
Dim da As New SqlDataAdapter ("ListaCategorias", cn)
da.SelectCommand.CommandType =CommandType.StoredProcedure

da.Fill(ds, "Categories")
```

Para ejecutar el procedimiento almacenado y guardar los registros devueltos en un DataSet, hemos invocamos el método Fill del objeto SqlDataAdapter. Este método rellena un objeto DataTable con los registros devueltos del procedimiento almacenado. Tras llenar el DataTable con los resultados del SP (ha devuelto un conjunto de filas), podemos vincular el DataTable a un control y mostrar los datos.

El código que sigue invoca al mismo SP pero lo hace con un objeto **DataReader *(clases conectadas):***

```
Dim cmd As SqlCommand = New SqlCommand("ListaCategorias", cn)
cmd.CommandType = CommandType.StoredProcedure
cn.Open()
Dim dr As SqlDataReader = cmd.ExecuteReader()
While dr.Read()
        campo1 = dr("CategoryId")
        campo2 = dr("CategoryName")
       ListBox1.Items.Add(campo1 & vbTab & campo2)
End While

'O bien,
'  dt.Load(dr)
'  DataGridView1.DataSource = dt

dr.Close()
cn.Close()
```

En este caso la ejecución del procedimiento almacenado la realiza el método ExecuteReader del objeto Command. Debemos acordarnos de cerrar tanto el DataReader como la conexión.

Utilizar parámetros

Cuando utilizamos procedimientos almacenados en una base de datos SQL Server o en cualquier otra base de datos basada en procedimientos, se pueden utilizar parámetros para pasar información y recuperar datos del procedimiento almacenado.

- Cuando utilizamos parámetros con una base de datos SQL Server y el proveedor nativo, los ***nombres*** de los parámetros que se agregan a la colección Parameters del objeto Command deben coincidir con los nombres de los parámetros del procedimiento almacenado; no obstante, el orden de los parámetros es flexible.

- Cuando utilizamos parámetros en una base de datos OLE DB, ***el orden*** de los parámetros en la colección Parámetros debe coincidir con el orden de los parámetros definidos en el procedimiento almacenado.

La siguiente tabla describe los tipos de parámetros disponibles en los procedimientos almacenados.

Parámetro	**Uso**
Input	Utilizado por la aplicación para enviar valores de datos específicos a un procedimiento almacenado
Output	Utilizado por un procedimiento almacenado para enviar valores específicos de retorno a la aplicación que lo invoca
InputOutput	Utilizado por un procedimiento almacenado tanto para recuperar información enviada por nuestra aplicación como para enviar valores de retorno específicos a la aplicación.
ReturnValue	Utilizado por un procedimiento almacenado para enviar un valor de retorno a la aplicación que lo invoca.

Una vez identificados los parámetros soportados por un procedimiento almacenado, debemos agregar los parámetros que utilizaremos a la colección Parameters del objeto Command.

Para crear un parámetro, crearemos un nuevo **objeto SqlParameter** con el nombre y tipo de datos del parámetro, según lo especificado por el procedimiento almacenado. A continuación, estableceremos la **propiedad Direction** del nuevo parámetro para indicar la forma en que el procedimiento almacenado utilizará el parámetro. Si el procedimiento almacenado devuelve un valor de retorno, crearemos un parámetro denominado **returnValue.** Si el parámetro es de entrada, estableceremos la **propiedad Value** para especificar los datos que deberían enviarse a SQL Server.

Pasar parámetros de entrada

Ejemplo: el procedimiento almacenado ProductsByCategory toma un parámetro de entrada denominado @CategoryID de tipo int, como muestra el siguiente código

```
Procedure ProductosDeCategoria(@CategoryID int ) As
SELECT ProductID, ModelName, UnitCost, ProductImage, Chairman
FROM Products
WHERE CategoryID=@CategoryID
```

Para invocar la ejecución del SP desde nuestra aplicación con ***clases desconectadas***

```
'Definir comando
Dim da as New SqlDataAdapter()
da.SelectCommand = New SqlCommand()
da.SelectCommand.Connection = cn
da.SelectCommand.CommandText = "ProductosDeCategoria"
da.SelectCommand.CommandType = CommandType.StoredProcedure

'Definir y agregar parámetro de entrada
Dim Param1 As New SqlParameter ("@CategoryID", SqlDbType.Int)
Param1.Direction = ParameterDirection.Input
Param1.Value = Cint(txtStartDate.Text)
da.SelectCommand.Parameters.Add(Param1)

'Ejecutar comando (el procedimiento almacenado)
ds = New DataSet()
da.Fill(ds, "MisProducts")
```

Para invocar el procedimiento almacenado ProductosdeCategoria, creamos un parámetro de entrada denominado @CategoryID y establecemos su valor con el valor de un cuadro de texto.

Deberíamos validar siempre el contenido de este cuadro de texto antes de enviar la entrada del usuario al procedimiento almacenado. Para el cometido del tema nos saltamos esta tarea…

Una vez creado el objeto Parameter, utilizamos el método Add de la colección Parameters del objeto SelectCommand. Una vez creado el objeto Command, utilizamos el método Fill para ejecutar el procedimiento almacenado y recuperar los registros en el datatable MisProducts.

Utilizar parámetros de salida

Ejemplo: El procedimiento almacenado ContarPedidos toma un ID de cliente y devuelve el número de pedidos pendientes del cliente. Utiliza un parámetro de entrada @CustomerID, y un parámetro de salida @ItemCount, ambos del tipo int.

```
Procedure ContarPedidos(@CustomerID nchar(5), @ItemCount int OUTPUT )
As
SELECT @ItemCount=COUNT(OrderID)
FROM Orders
WHERE CustomerID=@CustomerID
```

Para invocar la ejecución del SP desde nuestra aplicación con ***clases conectadas***

```
'Definir comando
Dim cmd As SqlCommand = New SqlCommand("ContarPedidos", cn)
cmd.CommandType = CommandType.StoredProcedure

'Definir y agregar parámetro de entrada a la colección parameter
Dim Param as SqlParameter
Param = New SqlParameter("@CustomerID", SqlDbType.nchar)
Param.Direction = ParameterDirection.Input
Param.Value = txtCustID.Text
cmd.Parameters.Add (Param)

'Definir y agregar parámetro de salida
Param = New SqlParameter("@ItemCount", SqlDbType.Int)
Param.Direction = ParameterDirection.Output
cmd.Parameters.Add (Param)

'Ejecutar comando (el procedimiento almacenado)
cn.Open()
cmd.ExecuteScalar()
cn.Close()

curSales = cmd.Parameters("@ItemCount").Value
```

Ojo!! Para obtener el valor de un parámetro de salida o un valor de retorno de un procedimiento almacenado que devuelve registros, debemos acceder al valor del parámetro de salida en la colección Parameters **después** de que el procedimiento almacenado se haya ejecutado. Podemos referenciar el valor del parámetro de

salida por el nombre o por el índice. El ejemplo recupera el valor del parámetro de salida @ItemCount por el nombre.

Algunas peculiaridades a tener en cuenta...

La sintaxis de los marcadores de posición de parámetros **depende del origen de datos**. Esta sintaxis se personaliza para un origen de datos específico, como se describe en la tabla siguiente.

Proveedor de datos	Sintaxis de nomenclatura de parámetros
System.Data.SqlClient	Usa parámetros con nombre, con el formato `@`*nombreDeParámetro*.
System.Data.OleDb	Usa marcadores de parámetro de posición, indicados por un signo de interrogación (`?`).
System.Data.Odbc	Usa marcadores de parámetro de posición, indicados por un signo de interrogación (`?`).
System.Data.OracleClient	Usa parámetros con nombre, con el formato `:`*nombreDeParámetro* (o *nombreDeParámetro*).

Utilizar parámetros con SqlCommand

Al utilizar parámetros con SqlCommand, los nombres de los parámetros agregados a la colección Parameters deben coincidir con los de los marcadores de parámetro del procedimiento almacenado. *El proveedor de datos de .NET Framework para SQL Server trata los parámetros del procedimiento almacenado como parámetros con nombre y busca los marcadores de parámetros que coinciden con sus nombres.*

El proveedor de datos de .NET Framework para SQL Server no permite usar el marcador de posición de signo de interrogación de cierre (?) para pasar parámetros a una instrucción SQL o a un SP, sino que se deben emplear utilizar parámetros con nombre, como se muestra en el ejemplo siguiente donde @CustomerID es el parámetro con nombre.

```
SELECT * FROM Customers WHERE CustomerID = @CustomerID
```

Utilizar parámetros con OleDbCommand o con OdbcCommand

Al utilizar parámetros con OleDbCommand o con OdbcCommand, el orden de los parámetros agregados a la colección Parameters debe coincidir con el de los parámetros definidos en el procedimiento almacenado. *El proveedor de datos de .NET Framework para OLE DB y el proveedor de datos de .NET Framework para ODBC consideran a los parámetros de un procedimiento almacenado como marcadores de posición y aplican los valores de los parámetros en orden.* Además, los parámetros de valores devueltos deben ser los primeros que se agreguen a la colección Parameters.

Además, el proveedor de datos de .NET Framework para OLE DB y el proveedor de datos de .NET Framework para ODBC no permiten usar parámetros con nombre para pasar parámetros a una instrucción SQL o a un procedimiento almacenado. En este caso, se debe utilizar el marcador de posición de signo interrogación de cierre (?), como se muestra en el ejemplo siguiente.

```
SELECT * FROM Customers WHERE CustomerID = ?
```

Por eso, el orden en que se agregan los objetos Parameter a la colección Parameters debe coincidir exactamente con la posición del marcador de posición de interrogación de cierre correspondiente al parámetro.

Más detalle en:
Configuración de parámetros y tipos de datos de parámetros - APOYO - (MSDN)
Parámetros de dataAdapter - APOYO - (MSDN)