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.