

Capítulo 2

Implementación de acceso a datos con ADO.NET

Objetivo

Al finalizar el capítulo, el alumno:

- Conocer la arquitectura de ADO.NET.
- Identificar los NET Data Providers.
- Utilizar las interfaces y clases del modelo de objetos de ADO.NET.
- Implementar una conexión a la base de datos usando la clase SqlConnection.
- Realizar consultas a la base de datos.
- Entender las ventajas y desventajas de las estrategias de trabajo (conectado y desconectado)

Temas

1. Definición de ADO.NET
2. Arquitectura de ADO.NET
3. .NET Data Providers
4. Modelo de datos conectado y desconectado
5. Datasets y Datatables
6. Transacciones

1. Definición de ADO.NET

Definición de ADO.NET


ADO.NET es un conjunto de clases en .NET que permite recuperar y manipular datos de diferentes fuentes.

Características

- Interoperabilidad
- Rendimiento
- Escalabilidad

2 - 4

Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

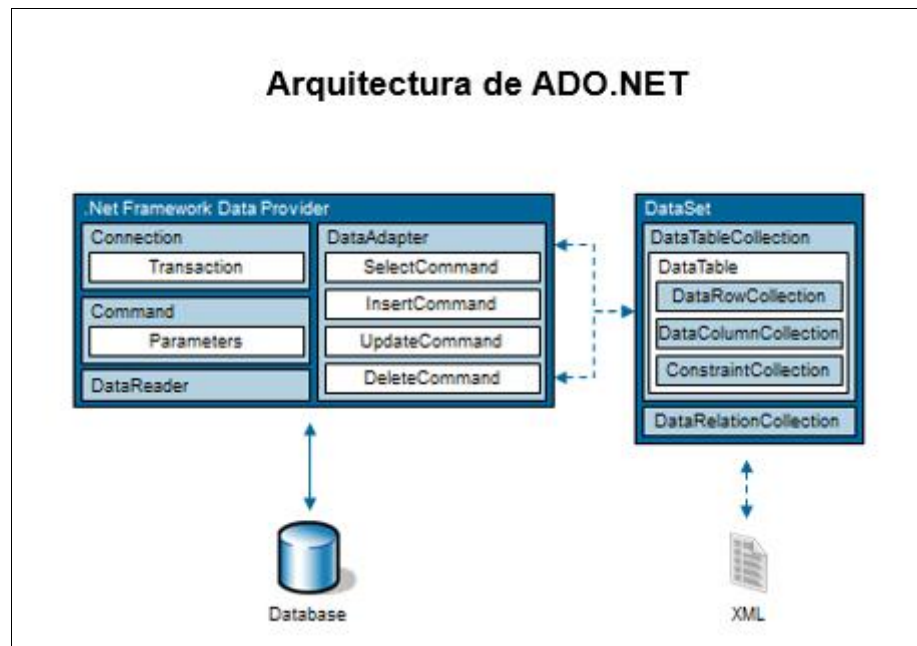


ADO.NET es un extenso conjunto de clases de .Net que permite recuperar y manipular los datos provenientes de múltiples fuentes. Además, incorpora un modelo de programación que incluye accesos nativos a fuentes de datos y XML.

Del mismo modo, ADO.NET permite alcanzar los siguientes objetivos en el desarrollo de una aplicación:

- **Interoperabilidad.** Porque emplea XML como el formato estándar, para transferir información desde el origen de datos hacia los aplicativos clientes. Dado que XML es un estándar de la industria, su utilización hace que la comunicación de datos, entre fuentes distintas, sea mucho más sencilla.
- **Rendimiento.** En su modo conectado, ADO.NET trabaja directamente con los tipos de datos de la base de datos, por lo tanto, evita el trabajo de realizar conversiones de tipos. Además, los conectores nativos de ADO.NET permiten establecer conexiones más rápidas con las fuentes de datos.
- **Escalabilidad.** ADO.NET conserva los recursos de un sistema, pues permite el trabajo en entornos desconectados que no requieren consumir recursos de conexiones duraderas con el servidor.

2. Arquitectura de ADO.NET



Existen dos componentes de ADO.NET que se pueden utilizar para acceder y manipular data:

- **Proveedores de datos de .NET Framework**

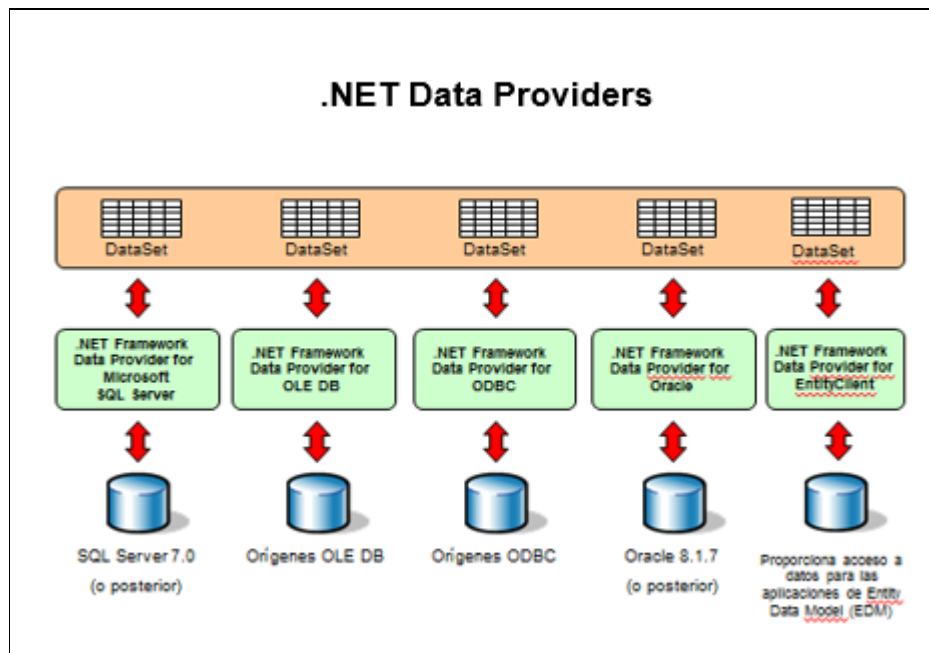
Los proveedores de datos de .NET Framework son componentes diseñados explícitamente para la manipulación de datos y el acceso rápido a datos de solo lectura y solo avance. Ofrece los siguientes objetos:

- **Connection**, proporciona conectividad a un origen de datos.
- **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.
- **DataReader**, permite recuperar información que será de solo lectura, y "forward only". Los resultados se devuelven como se ejecuta la consulta, y se almacenan en el buffer de la red en el cliente hasta que se solicita mediante el método Read del DataReader. Utilizando el DataReader puede aumentar el rendimiento de aplicaciones, tanto por la recuperación de datos tan pronto como esté disponible.
- **DataAdapter**, proporciona el puente entre el objeto DataSet y el origen de datos. Asimismo, utiliza objetos **Command** para ejecutar comandos SQL en el origen de datos, tanto para cargar **DataSet** con datos, así como para reconciliar en el origen de datos, los cambios aplicados a los datos incluidos en el **DataSet**.

- **DataSet**

Los DataSet de ADO.NET están expresamente diseñados para el acceso a datos, independientemente del origen. Como resultado, se puede utilizar con múltiples y distintos orígenes de datos, con datos XML o para administrar datos locales de la aplicación. Asimismo, 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, las cuales se encuentran referidas con los datos incluidos en los objetos DataTable.

3. .NET Data Providers



ADO.NET incluye los Framework Data Providers los cuales permiten conectarse a una base de datos, ejecutar comandos y retornar resultados. Del mismo modo, crean una mínima capa entre la fuente de datos y el código, lo cual incrementa performance y no sacrifica funcionalidad.

- **.Net Framework Data Provider para SQL Server**

Provee de acceso optimizado a SQL Server 7.0 o superior y a Microsoft Data Engine (MSDE). Para versiones inferiores a la 7.0, se podrá utilizar el .Net Framework Data Provider para OLE DB (Object Linking and Embedding for Databases).

El .Net Framework Data Provider para SQL Server, utiliza su propio protocolo ligero para comunicarse directamente con el SQL Server, sin la necesidad de una capa OLE DB u ODBC (Open Data Base Connectivity).

Las clases para este proveedor están ubicadas en el namespace **System.Data.SqlClient**.

- **.Net Framework Data Provider para OLE DB**

Utiliza OLE DB nativo, a través de interoperabilidad COM, para habilitar el acceso a datos. Para utilizar el .Net Framework Data Provider para OLE DB, se debe usar un proveedor OLE DB.

Para ello, Microsoft ha probado los siguientes proveedores OLE DB con ADO.Net:

- Microsoft OLE DB Provider for SQL Server (SQLOLEDB)
- Microsoft OLE DB Provider for Oracle (MSDAORA); and
- OLE DB Provider for Microsoft Jet (Microsoft.Jet.OLEDB.4.0).

Las clases para este proveedor están ubicadas en el namespace **System.Data.OleDb**.

- **.Net Framework Data Provider para ODBC**

Utiliza el nativo ODBC Driver Manager, a través de interoperabilidad COM, para habilitar el acceso a datos. Para utilizar el .Net Framework Data Provider para ODBC, se debe usar un driver ODBC. Los siguientes ODBC drivers han sido probados con ADO.Net:

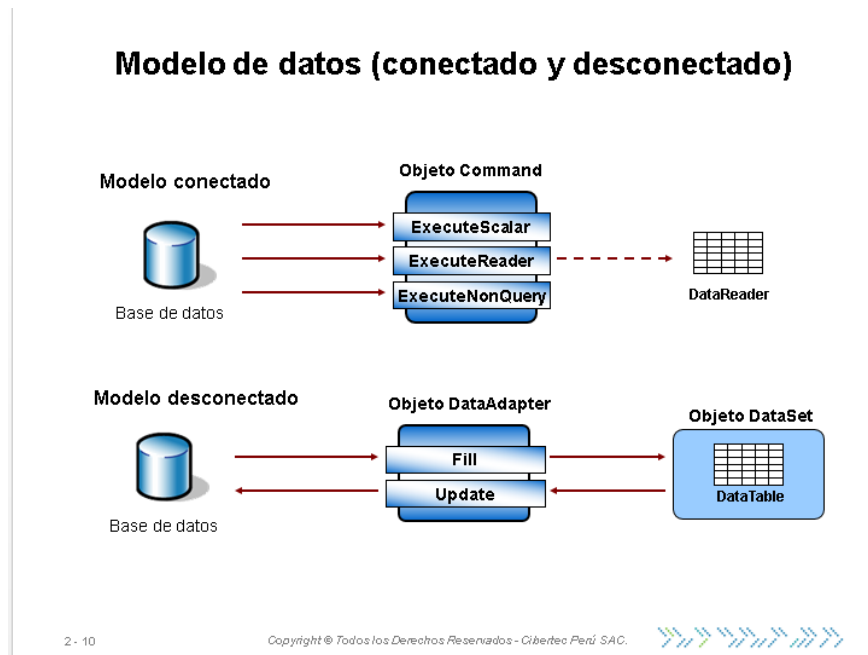
- SQL Server
- Microsoft ODBC for Oracle
- Microsoft Access Driver (*.mdb).

Las clases para este proveedor están ubicadas en el namespace **System.Data.ODBC**.

ADO.NET especifica un modelo común de objetos para los .Net Framework Data Providers. Las interfaces implementadas más comunes son:

- **IDbConnection**. Esta interface permite establecer una conexión con una fuente de datos específica. Cada .Net Framework Data Provider incluye una clase que implementa IDbConnection. Por ejemplo, el .Net Framework Data Provider para SQL Server, incluye una clase llamada **SqlConnection**.
- **IDbCommand**. Esta interface permite ejecutar un comando contra la fuente de datos. Los comandos pueden incluir parámetros y pueden ejecutarse dentro de una transacción. Cada .Net Framework Data Provider incluye una clase que implementa el IDbCommand. Por ejemplo, el .Net Framework Data Provider para SQL Server, incluye una clase llamada **SqlCommand**.
- **IDataReader**. Esta interface permite lecturas del tipo forward-only y read-only de una fuente de datos.
- **IDataAdapter**. Esta interface permite poblar de data a un DataSet, el cual almacena los cambios que se realicen sobre la data; cualquier cambio es grabado de vuelta a la fuente de datos.

4. Modelo de datos conectado y desconectado



Modelo conectado

Este modelo permite usar un Data Provider, como el .NET Framework Data Provider para Microsoft SQL Server, para conectarse a una base de datos y ejecutar sentencias SQL en esa base de datos.

La conexión a la base de datos permanece abierta, mientras el comando se ejecuta y se leen los datos. Este modelo utiliza, principalmente, los objetos Command y Datareader. Algunas de las ventajas dentro de este modelo son:

- Mayor control sobre la ejecución de sentencias SQL. Usando comandos, se obtiene control directo sobre cómo y cuándo una sentencia SQL o procedimiento almacenado es ejecutado.
- Menos sobrecarga. Leyendo y escribiendo directamente sobre la base de datos, se evita el almacenamiento de datos en un DataSet en memoria, lo cual reduce el consumo de recursos de la aplicación.
- Funcionalidad extra. Existen ciertas operaciones, como los comandos DDL (Data Definition Language) que solo se pueden ejecutar a través de objetos del tipo Command.
- Menos programación en algunas circunstancias. Principalmente en aplicaciones Web, se requiere algo de programación extra para guardar el estado de un dataset. Si se usa un objeto DataReader, se evitan los pasos extras requeridos para manipular un dataset.

Modelo desconectado

Este modelo permite crear un caché en memoria con los datos obtenidos desde una fuente de datos y manejarlos, sin mantener una conexión activa a la base de datos. Se puede ver, modificar y eliminar datos en el caché, y luego, transmitir los cambios realizados en el caché, a la fuente de datos original. Se usan principalmente los objetos `DataAdapter` y `DataSet`, como base de este modelo.

En un entorno desconectado, el usuario o aplicación, no mantiene una conexión abierta al servidor de base de datos. Algunas de las ventajas dentro de este modelo son:

- Intercambio de datos con otras aplicaciones. Un `DataSet` es un objeto con el cual se puede transportar de manera sencilla datos entre los componentes de la propia aplicación u otras aplicaciones.
- Movimiento de datos entre capas en una aplicación distribuida. Manipulando datos en un `DataSet`, se puede mover estos datos entre la capa de acceso a datos a la capa de negocio y a la capa de presentación, de manera fácil.
- Trabajo con múltiples tablas. Un `DataSet` puede contener múltiples tablas, por lo cual, se puede trabajar con las tablas individualmente o se puede navegar, a través de ellas, como tablas padre-hijo.
- Mantenimiento de un conjunto de registros para su reutilización. Un `DataSet` permite trabajar con el mismo conjunto de datos repetidamente, sin necesidad de volver a consultar a la fuente de datos.
- Manipulación de datos provenientes de distintas fuentes de datos. Las tablas dentro de un `DataSet` pueden contener datos de distintas fuentes. Una vez que se encuentran dentro de un mismo `DataSet`, se pueden manipular y relacionar como si fueran datos que provienen de una misma fuente.
- Enlace de datos (data binding). Si se está trabajando con formularios, normalmente es más sencillo enlazar controles del formulario a datos contenidos en un `DataSet` en lugar de cargar valores, a través de programación.
- Facilidad en la programación. Cuando se trabaja con un `DataSet` se pueden crear clases que representan la estructura específica del objeto, lo cual hace más fácil la programación, porque estas clases son soportadas por herramientas del Visual Studio como Intellisense, Data Adapter Configuration Wizard, etc.

5. Datasets y Datatables

Datasets y Datatables

- Dataset: contenedor de “tablas” (entidades/datatables) con relaciones, similares al modelo que se tiene en una base de datos relacional, pero existe en memoria de la aplicación. No depende de ningún proveedor de base de datos.
- Datatable: representa a las “tablas” de un dataset. Está compuesto de DataRow y DataColumn. Un Dataset puede acceder a sus tablas por medio de su colección Tables:
 - ds.Tables[“Person”]
 - ds.Tables[0]

2 - 11

Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.



ADO.NET brinda soporte para acceso desconectado, a través del objeto Dataset, el cual sirve como una representación en memoria de una estructura relacional de datos. El objeto Dataset es, por diseño, un objeto desconectado, pues utiliza una conexión para obtener datos, y luego de obtenerlos y mantenerlos en memoria, no usa más la conexión, salvo para llevar cambios a la fuente de datos original.

El DataSet es el principal objeto para el trabajo con el modelo desconectado de ADO .NET, puesto que es una representación en memoria de los datos que pueden provenir de distintas fuentes. Además, representa un completo conjunto de datos donde se pueden tener tablas relacionadas, restricciones y relaciones entre tablas.

Los objetos más importantes de este modelo son:

DataTable: representa una tabla dentro del DataSet. Cada tabla es contenida en el DataTableCollection (Tables) del DataSet. Su esquema se define por objetos del tipo DataColumn, los cuales están contenidos dentro del DataColumnCollection (Columns), así como por restricciones contenidas en un ConstraintCollection (Constraints). Un DataTable también contiene un conjunto de objetos del tipo DataRow contenidos en el DataRowCollection (Rows). Los datos se almacenan en los objetos del tipo DataRow.

DataView: representa una vista personalizada de un DataTable. Su principal utilización está relacionada al ordenamiento, filtración, búsqueda, edición y navegación.

DataRow: representa una fila de datos dentro de un DataTable. Su principal uso es insertar, eliminar y actualizar datos dentro de un DataTable.

DataColumn: es el objeto base para la construcción del esquema de un DataTable. De tal forma que, para cada DataColumn, se define un tipo de dato.

DataRelation: vienen a ser cada una de las relaciones que se pueden implementar (así como en una BD física) en función de las llaves foráneas que apuntan hacia una llave primaria, dentro de las tablas de un Dataset. Cada relación formará parte de la colección DataRelations del Dataset

6. Transacciones

Las transacciones de ADO.NET se utilizan cuando se desea enlazar varias tareas para que se ejecuten como una sola unidad de trabajo. Por ejemplo, imagine que una aplicación realiza dos tareas. Primero, actualiza una tabla con información de pedidos. Luego, actualiza una tabla que contiene la información de inventario, cargando en cuenta los elementos pedidos. Si alguna de las tareas da error, ambas actualizaciones se revierten. Para llevar a cabo una transacción debe considerarse los siguientes pasos:

- Llamar al método `BeginTransaction` del objeto `SqlConnection` para marcar el comienzo de la transacción. El método `BeginTransaction` devuelve una referencia a la transacción. Esta referencia se asigna a los objetos `SqlCommand` que están inscritos en la transacción.
- Asignar el objeto `Transaction` a la propiedad `Transaction` del objeto `SqlCommand` que se va a ejecutar. Si el comando se ejecuta en una conexión con una transacción activa y el objeto `Transaction` no se ha asignado a la propiedad `Transaction` del objeto `Command`, se inicia una excepción.
- Ejecutar los comandos necesarios.
- Llamar al método `Commit` del objeto `SqlTransaction` para completar la transacción, o al método `Rollback` para finalizarla. Si la conexión se cierra o elimina antes de que se hayan ejecutado los métodos `Commit` o `Rollback`, la transacción se revierte.