

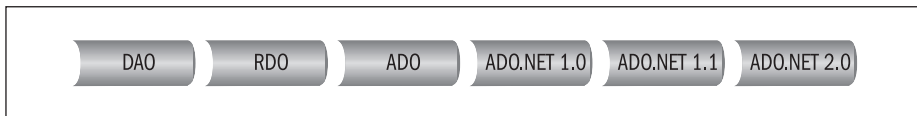
# Acceso a datos

En este capítulo observaremos los principales aspectos del acceso a datos, así como su evolución a través del tiempo. Además, veremos los distintos proveedores de datos que ofrece el .NET Framework, tanto en sus versiones 1.0 como en 1.1 y la actual, 2.0.

<b>Evolución del acceso a datos</b>	<b>14</b>
Distintas versiones de ADO.NET	15
¿Qué son los proveedores de datos de .NET?	15
<b>Aplicaciones conectadas o desconectadas</b>	<b>18</b>
<b>Resumen</b>	<b>19</b>
<b>Actividades</b>	<b>20</b>

# EVOLUCIÓN DEL ACCESO A DATOS

Históricamente, han existido diferentes modos de acceso a las bases de datos. En un principio, las bases de datos permitían, además de manejar el almacenamiento de los propios datos, escribir el código necesario para interactuar con el usuario final; en el otro extremo, empezaron a aparecer lenguajes de programación en donde la finalidad principal era ofrecer una interfaz gráfica rica, pero con sentencias de acceso a datos limitadas o nulas. Cuando **Visual Basic** ganó popularidad como herramienta **RAD** (*Rapid Application Development*), muchos tuvieron que acostumbrarse a su forma de acceso a los datos, la cual lo hacía a través de objetos que encapsulaban su tratamiento. Así fue entonces como se sucedieron **DAO** (*Data Access Objects*), **RDO** (*Remote Data Objects*) y **ADO** (*ActiveX Data Objects*), tal como lo muestra la **Figura 1**.



**Figura 1.** Diferentes tecnologías de acceso a datos de **Microsoft** a través del tiempo.

Desde el punto de vista de la arquitectura, las aplicaciones pasaron de ser monolíticas a cliente-servidor, en donde los datos no se encontraban almacenados dentro de la aplicación, sino en archivos de datos independientes o, mejor aún, en un **sistema de gestión de bases de datos relacionales** (**RDBMS**, por su sigla en inglés). Al poco tiempo, Internet se hizo masiva y las grandes empresas detectaron su potencial para desarrollar aplicaciones en lugar de ser solamente sitios institucionales. Microsoft, por su lado, llevó su tecnología a un estándar propietario, llamado **COM** (*Component Object Model*), en el cual los objetos utilizados por ciertas aplicaciones podían ser reutilizados fácilmente por otras. Incluso, dentro de su filosofía, un programador podía desarrollar un componente o programa reutilizable sin interfaz gráfica, en un lenguaje como **C++**, y ser utilizado o consumido tanto por una aplicación **Win32** desarrollada en **Visual Basic** como por una aplicación web desarrollada en **ASP**. De esta forma es como ADO se consolida, ya que un



## PROVEEDORES DE DATOS .NET

Aunque los proveedores de datos **.NET** de **Odbc** y **Oracle** no estaban integrados al **.NET Framework** 1.0 en su lanzamiento, posteriormente fueron puestos a disposición de los desarrolladores para su descarga desde el sitio web de Microsoft. En el caso de necesitar un proveedor de datos diferente, el mejor lugar para buscar es el sitio web del fabricante de la base de datos.

programador de aplicaciones Win32 o un programador ASP aplicaban los mismos conceptos o, mejor dicho, los mismos objetos para acceder a datos. Cuando ADO estaba siendo utilizado por millones de programadores en todo el mundo, Microsoft se encontraba trabajando en su nueva generación, que en un principio la llamaban **ADO+**; luego se convertiría en **ADO.NET**.

## Distintas versiones de ADO.NET

Se llama **ADO.NET** a todas las clases, interfases, enumeradores y delegados que se encuentran dentro de los espacios de nombres **System.Data** y **System.Xml** del .NET Framework. de Microsoft.

Por cada versión del .NET Framework existe una versión de ADO.NET; dichas versiones van obteniendo mejoras desarrolladas por Microsoft, muchas de ellas por pedido de la comunidad mundial de desarrolladores. Las versiones liberadas al momento de edición de este título son las siguientes:

VERSIÓN	LANZAMIENTO	HERRAMIENTA
.NET Framework 1.0	2002	Visual Studio .NET (7.0)
.NET Framework 1.1	2003	Visual Studio .NET 2003 (7.1)
.NET Framework 2.0	2005	Visual Studio 2005 (8.0)

**Tabla 1.** Relación entre las versiones de las herramientas de desarrollo de Microsoft y las versiones del .NET Framework.

## ¿Qué son los proveedores de datos de .NET?

Una de las características fundamentales que tenía ADO era que funcionaba a través de **OleDb**, una API de acceso a datos creada por Microsoft. La ventaja de esta funcionalidad era que utilizando los mismos objetos podíamos desarrollar una aplicación que trabajara contra cualquier base de datos que tuviera un dispositivo o **driver** de OleDb. Incluso bases de datos más antiguas podían accederse a través de un conector de **OleDb para ODBC**. La principal desventaja de acceder a datos sólo a través de OleDb es que no se pueden aprovechar

### III ¿QUÉ ES UNA API?

Una **API (Application Programming Interface)** es un software de bajo nivel encargado de funcionar como intermediario entre aplicaciones o con el sistema operativo. El **.NET Framework** encapsula múltiples API en forma de clases, organizadas a través de **Namespaces**. De esta manera se minimiza el acceso directo a las API aunque, también puede acceder a ellas a través de **Platform Invoke**.

características específicas de cada **RDBMS** y que existe un consumo extra de recursos al tener que traducir desde COM a OLEDB y desde OLEDB al lenguaje nativo de cada **RDBMS** (*Relational Database Management System*), y viceversa. ADO.NET solucionó estas desventajas, pero manteniendo de cierta manera las ventajas que tenía ADO, a través de los **proveedores de datos de .NET** (**.NET Data Providers** en su idioma original). Los proveedores de datos implementan interfaces que definen cómo deben ser las clases que componen ADO.NET, luego existen clases específicas para ciertos RDBMS que explotan al máximo sus características, incluso comunicándose a través del lenguaje nativo en el caso de algunos RDBMS como **SQL Server** u **Oracle**, así como clases específicas para las API de acceso a datos tales como **Odbc** y **OleDb**.

VERSIÓN	PROVEEDORES DE DATOS
<b>ADO.NET 1.0</b>	SqlClient
	OleDb
<b>ADO.NET 1.1</b>	SqlClient
	OleDb
	Odbc
	OracleClient
<b>ADO.NET 2.0</b>	SqlClient
	OleDb
	Odbc
	OracleClient

**Tabla 2.** Distintas versiones de **ADO.NET** y sus proveedores de datos integrados.

Otra de las ventajas del .NET Framework es que los fabricantes de software de bases de datos pueden crear sus propias clases para extenderlo, como por ejemplo, **MySQL** e **IBM** para sus bases de datos **DB2** e **Informix**. En el gran mercado del software existen también aquellas empresas que se dedican solamente a fabricar drivers, conectores o proveedores de datos para comunicar múltiples bases de datos con múltiples plataformas de desarrollo. Estas empresas, siendo **CoreLab** y **DataDirect** (ex **InterSolv**) las más conocidas, se encargan hacer que la



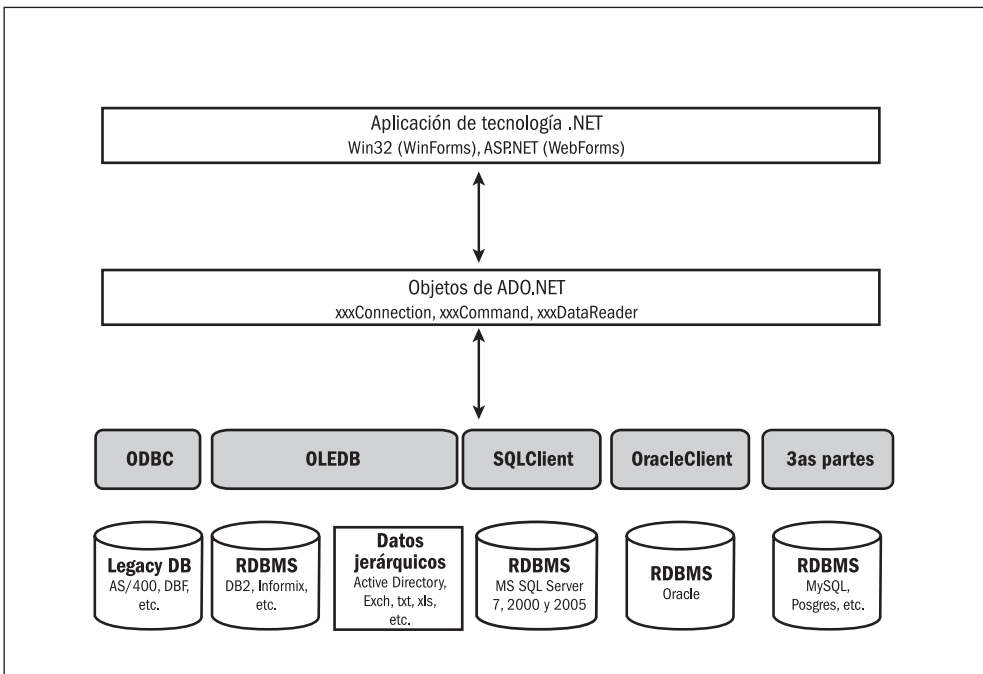
## VERSIONES DE SQL SERVER

Hasta su versión **7.0**, **SQL Server** mantuvo de forma prolija la relación entre su nombre oficial y su versión. Al momento de salir al mercado la versión **8.0**, el equipo de marketing de Microsoft decidió que era mejor asociar el número **2000** al producto. Esta misma razón se aplicó la versión **9.0**, la cual antes de tomar su nombre oficial, **SQL Server 2005**, fue llamada con el nombre en código **Yukon**.

comunicación entre las más distintas plataformas sea cada vez más fácil y rápida pero, obviamente, cobrando por ello.

La elección proveedores de datos **.NET** debería comenzar con la búsqueda de un proveedor de datos nativo para la fuente de datos que estaremos utilizando. Si no encontramos dicho proveedor, ni del mismo fabricante ni de un tercero, deberíamos buscar un controlador o driver OLEDB y, en última instancia, un controlador ODBC. Estos dos últimos a utilizarse con los proveedores de datos **OleDb** y **Odbc** respectivamente.

En el caso de **SQL Server**, la elección es simple: si poseemos la versión 6.5 o anterior, debemos usar los proveedores de datos **OleDb**, y si contamos con la versión 7.0 o posterior, debemos usar el proveedor de datos nativo de SQL Server, es decir, **SqlClient**. Microsoft Access utiliza el motor de base de datos **JET**, por lo cual el proveedor de datos debe ser **OleDb**.



**Figura 2.** ADO.NET y su relación entre las aplicaciones y las fuentes de datos.



## PRINCIPALES SISTEMAS DE BASES DE DATOS

Entre los grandes se destacan **Oracle**, **IBM DB2** e **Informix**, **Sybase** y **Microsoft SQL Server**. En código abierto existen **MySQL** y **PostgreSQL**. Se siguen utilizando, aunque cada vez menos, bases de datos basados en archivos de texto plano o **TXT**, pasando por los archivos basados en **DBase** o **DBF**, hasta los archivos de datos de Microsoft Access, con extensión **MDB**.

Los proveedores de datos .NET no son necesariamente sinónimos de ADO.NET. Existen muchas clases en ADO.NET que no pertenecen a ningún proveedor de datos, tales como los **Dataset** y todas las clases relacionadas con **XML**, es decir que son genéricas. Las clases que están encapsuladas dentro de los proveedores de datos son las siguientes:

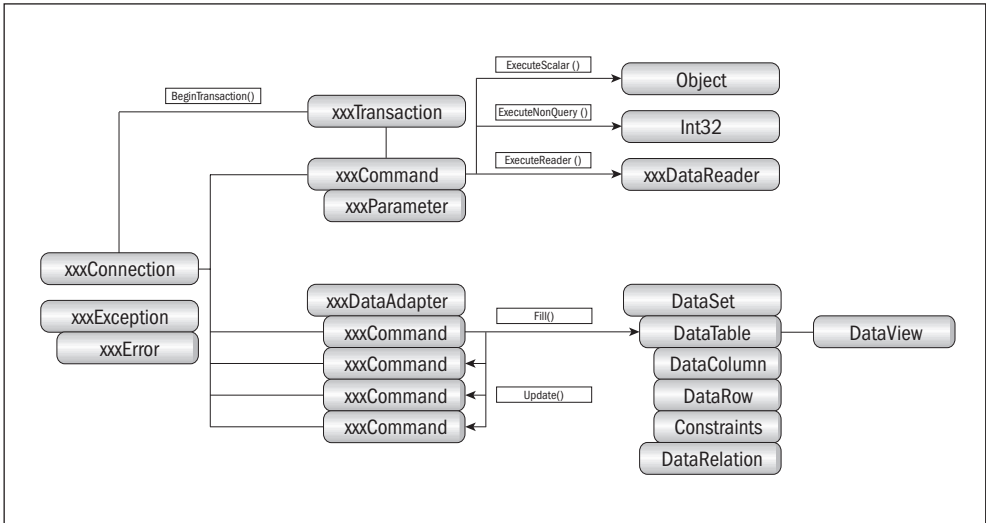
CLASE	DESCRIPCIÓN
<b>xxxConnection</b>	Nos permite la comunicación física entre la base de datos y nuestra aplicación o servicio.
<b>xxxException</b>	Sirve para atrapar excepciones específicas de bases de datos.
<b>xxxError</b>	Muchas veces, una excepción se produce a causa de múltiples errores. Por esto las excepciones poseen colecciones de errores. No está disponible en todos los proveedores de datos.
<b>xxxTransaction</b>	Se utiliza cuando necesitamos asegurarnos que la ejecución de dos o más sentencias SQL sean tomadas como una unidad.
<b>xxxCommand</b>	Es la clase que nos permite establecer y/o ejecutar una consulta o procedimiento almacenado.
<b>xxxParameter</b>	Las consultas o procedimientos almacenados de los comandos suelen tener parámetros de entrada y/o salida, los cuales se establecen a través de una colección.
<b>xxxDataReader</b>	Medio rápido y barato, desde el punto de vista del consumo de recursos, de almacenar datos resultantes de la ejecución de un comando que devuelva un conjunto de resultados.
<b>xxxDataAdapter</b>	Es el puente entre una fuente de datos y un Dataset, permitiendo llenarlo, para luego modificar los datos en memoria y posteriormente refrescar esos cambios en la fuente de datos de origen.
<b>xxxCommandbuilder</b>	Se utiliza para crear consultas de tipo INSERT, UPDATE y/o DELETE en un adaptador de datos.

**Tabla 3.** Las clases que conforman a los proveedores de datos .NET.

## APLICACIONES CONECTADAS O DESCONECTADAS

Desde un principio, ADO ofrecía múltiples combinaciones de parámetros que permitían que los datos, una vez en la memoria de la aplicación, puedan o no enterarse de los cambios que se producían en la fuente de datos original. Para poder enterarse de los cambios, era necesario mantener una conexión abierta todo el tiempo, lo que resulta en un alto consumo de recursos desde todos los puntos de vista posibles, tales como el cliente, el servidor y la red misma. Para solucionar estos problemas, el mismo ADO ofrecía una forma de trabajar desconectados, es decir, conectarse, traer los datos en memoria, cerrar la conexión, trabajar los datos en memoria y luego, si fuera necesario, volver a conectarse para actualizar la fuente de datos original o recibir nuevos datos.

ADO.NET se basó en este último paradigma, aunque para algunas tareas simples es necesario trabajar en forma conectada, aunque durante un pequeño lapso de tiempo. En resumen, se podría decir que ADO.NET trabaja la mayor parte del tiempo en forma desconectada.



**Figura 3.** Mapa de clases de ADO.NET.

Cuando trabajemos con comandos y/o lectores de datos o **DataReaders**, se suele decir que esa es la forma **conectada** de acceso a datos, ya que es necesario establecer una conexión, abrirla explícitamente, ejecutar el comando, recorrer o procesar los datos y, por último, cerrar la conexión, también explícitamente. En cambio, se le llama **desconectado** al hecho de trabajar con los adaptadores de datos o **DataAdapters** y **DataSets**. Estos últimos almacenan datos en memoria durante todo el tiempo que sea posible, pero sin necesitar una conexión abierta, la cual solamente se abre y cierra automáticamente al llenar un **DataSet** o al refrescar la fuente de datos original.

## RESUMEN

Acabamos de ver la historia de las herramientas de acceso a datos utilizadas por Microsoft en sus plataformas y cuáles son las clases que conforman ADO.NET, una de las partes fundamentales del .NET Framework. A través de los capítulos subsiguientes iremos viendo en forma pormenorizada cada una de estas clases y la forma de utilizarlas. A continuación recorreremos en detalle las conexiones.



### TEST DE AUTOEVALUACIÓN

- 1** Visite los sitios de los fabricantes de bases de datos y busque información sobre los proveedores de datos .NET.

---

- 2** Si existe alguna base de datos en la cual el fabricante no haya desarrollado un proveedor de datos nativo, busque sitios alternativos de terceras partes.

---

- 3** Explore los sitios web oficiales de Microsoft y de las comunidades de desarrolladores de .NET, especialmente en la parte de acceso a datos.

---

- 4** Revise el explorador de objetos (object browser) en cualquier versión de Visual Studio que posea, especialmente los espacios de nombres System.Data y System.Xml.

---

- 5** Si posee alguna versión anterior de Visual Studio, como por ejemplo, la versión 6, observe a grandes rasgos cómo funcionan los objetos Connection, Command y Recordset. Recuerde que deberá usar las herramientas Visual Basic 6 o Visual Interdev 6.

---

- 6** Intente realizar acceso a datos con ADO tradicional desde una aplicación Office a través del Editor de Secuencias de Comandos (Script Editor) o el Editor de Visual Basic.

---

- 7** Investigue en distintos sitios de Internet sobre cómo es el acceso a datos en otras tecnologías, como por ejemplo, JDBC en Java.

---

- 8** Infórmese sobre las incipientes bases de datos relacionales orientadas a objetos, y analice sus ventajas y desventajas. Algunas son llamadas ORDBMS.

---

- 9** Busque documentación acerca de los estándares SQL, especialmente las versiones SQL99 y SQL2003.

---