ADO .NET

Bases de datos

Cualquier aplicación de interés requiere el almacenamiento y posterior recuperación de los datos con los que trabaje (pedidos en aplicaciones de comercio electrónico, datos de personal para las aplicaciones de recursos humanos, datos de clientes en sistemas CRM, etc.). Los sistemas de gestión de bases de datos (DBMSs) nos permiten almacenar, visualizar y modificar datos, así como hacer copias de seguridad y mantener la integridad de los datos, proporcionando una serie de funciones que facilitan el desarrollo de nuevas aplicaciones.

Desde un punto de vista intuitivo, una base de datos no es más que un fondo común de información almacenada en una computadora para que cualquier persona o programa autorizado pueda acceder a ella, independientemente de su lugar de procedencia y del uso que haga de ella. Algo más formalmente, una base de datos es un conjunto de datos comunes a un "proyecto" que se almacenan sin redundancia para ser útiles en diferentes aplicaciones.

El Sistema de Gestión de Bases de Datos (DBMS) es el software con capacidad para definir, mantener y utilizar una base de datos. Un sistema de gestión de bases de datos debe permitir definir estructuras de almacenamiento, así como acceder a los datos de forma eficiente y segura. Ejemplos: Oracle, IBM DB2, Microsoft SQL Server, Interbase...

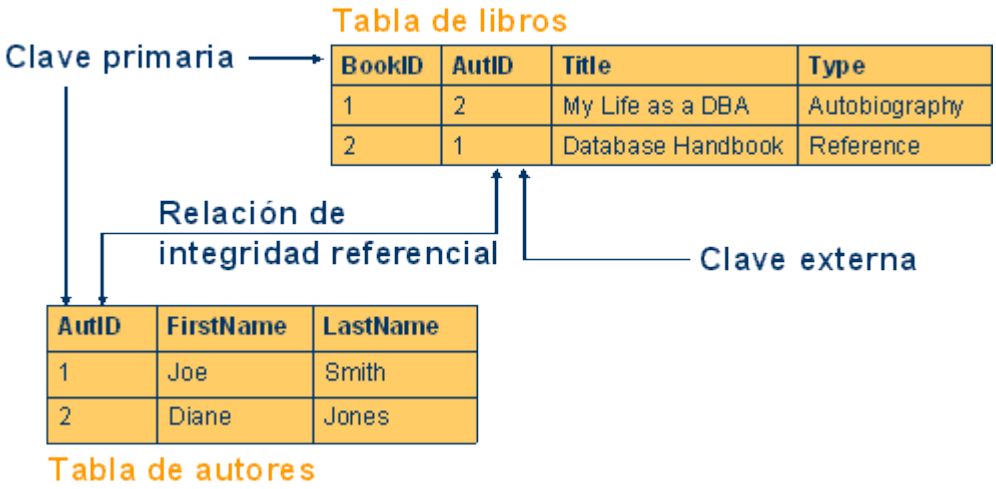
En una base de datos, los datos se organizan independientemente de las aplicaciones que los vayan a usar (independencia lógica) y de los ficheros en los que vayan a almacenarse (independencia física). Además, los datos deben ser accesibles a los usuarios de la manera más amigable posible, generalmente mediante lenguajes de consulta como SQL o Query-by-example. Por otro lado, es esencial que no exista redundancia (esto es, los datos no deben estar duplicados) para evitar problemas de consistencia e integridad.

Bases de datos relacionales

- **Tabla o relación**: Colección de registros acerca de entidades de un tipo específico (p.ej. alumnos).
- **Atributo, campo o columna**: Propiedad asociada a una entidad (p.ej. nombre, apellidos...). Cada atributo tiene un tipo asociado (p.ej. entero, cadena de caracteres...) y puede tomar el valor nulo (null).
- **Tupla, registro o fila**: Datos relativos a un objeto distinguible de otros (p.ej. un alumno concreto).

Se pueden establecer relaciones entre las tablas de una base de datos relacional mediante el uso de claves primarias y claves externas (p.ej. cada libro tiene, al menos, un autor).

- **Clave primaria**: Conjunto de atributos que nos permiten identificar unívocamente a una entidad dentro de un conjunto de entidades (p.ej. número de matrícula). La clave primaria garantiza la unicidad de una tupla, pues no se permite la existencia de varios registros que compartan su clave primaria.
- **Clave externa**: Conjunto de atributos que hacen referencia a otra tabla, lo que nos permite establecer relaciones lógicas entre distintas tablas. Los valores de los atributos de la clave externa han de coincidir con los valores de los atributos de una clave (usualmente la primaria) en una de las tuplas de la tabla a la que se hace referencia (integridad referencial).



SQL

Lenguaje estándar para acceder a una base de datos relacional, estandarizado por el American National Standards Institute (ANSI): SQL-92. En gran parte, los distintos DBMS utilizan todos el mismo SQL, si bien cada vendedor le ha añadido sus propias extensiones. El lenguaje SQL se divide en:

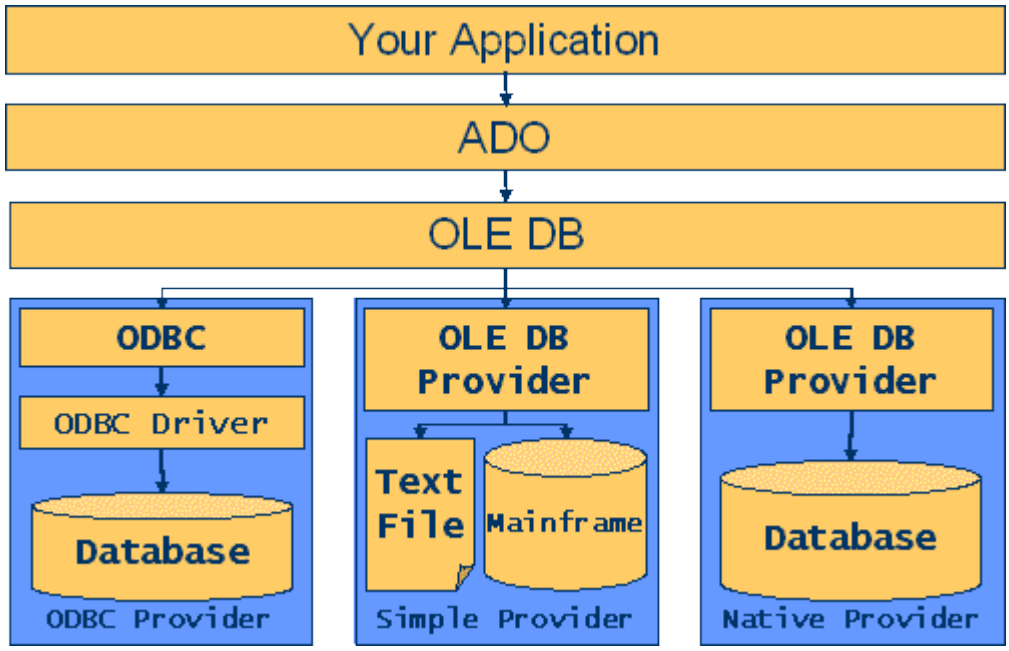
- DDL (Data Definition Language), utilizado para crear y modificar la estructura de la base de datos (p.ej. CREATE TABLE).
- DML (Data Manipulation Language), empleado para manipular los datos almacenados en la base de datos (p.ej. consultas con la sentencia SELECT).
- DCL (Data Control Language), para establecer permisos de acceso (GRANT, REVOKE, DENY) y gestionar transacciones (COMMIT y ROLLBACK).

Interfaces de acceso a bases de datos

Evolución histórica de los "estándares" propuestos por Microsoft:

1. **ODBC (Open Database Connectivity)**: API estándar ampliamente utilizado, disponible para múltiples DBMSs, utiliza SQL para acceder a los datos.

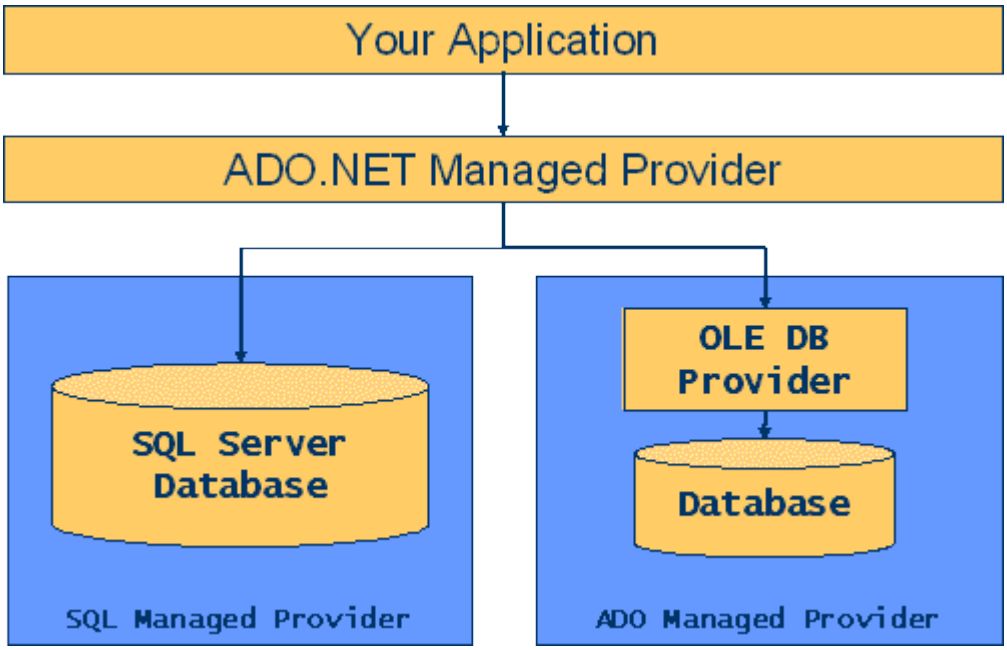
2. **DAO (Data Access Objects)**: Interfaz para programar con bases de datos JET/ISAM, utiliza automatización OLE y ActiveX.
3. **RDO (Remote Data Objects)**: Fuertemente acoplado a ODBC, orientado al desarrollo de aplicaciones cliente/servidor.
4. **OLE DB**: Construido sobre COM, permite acceder a bases de datos tanto relacionales como no relacionales (no está restringido a SQL). Se puede emplear con controladores ODBC y proporciona un interfaz a bajo nivel en C++.
5. **ADO (ActiveX Data Objects)**: Ofrece un interfaz orientado a objetos y proporciona un modelo de programación para OLE DB accesible desde lenguajes distintos a C++ (p.ej. Visual Basic).



ADO se diseñó para su uso en arquitecturas cliente/servidor con bases de datos relacionales (no jerárquicas, como es el caso de XML). Su diseño no está demasiado bien factorizado (ya que existen muchas formas de hacer las cosas y algunos objetos acaparan demasiadas funciones) y ADO no estaba pensado para arquitecturas multicapa en entornos distribuidos.

ADO .NET es una colección de clases, interfaces, estructuras y tipos enumerados que permiten acceder a los datos almacenados en una base de datos desde la plataforma .NET. Si bien se puede considerar una versión mejorada de ADO, no comparte con éste su jerarquía de clases (aunque sí su funcionalidad).

ADO .NET combina las capas ADO y OLE DB en una única capa de proveedores (*managed providers*). Cada proveedor contiene un conjunto de clases que implementan interfaces comunes para permitir el acceso uniforme a distintas fuentes de datos. Ejemplos: ADO Managed Provider (da acceso a cualquier fuente de datos OLE DB), SQL Server Managed Provider (específico para el DBMS de Microsoft), Exchange Managed Provider (datos almacenados con Microsoft Exchange)...



ADO .NET usa XML. De hecho, los conjuntos de datos se almacenan internamente en XML, en vez de almacenarse en binario como sucedía en ADO. Al estar los datos almacenados en XML, se simplifica el acceso a los datos a través de HTTP (algo que ocasiona problemas en ADO si los datos tienen que pasar cortafuegos). Por otro lado, se simplifica la comunicación entre aplicaciones al ser XML un formato estándar (p.ej. comunicación con applets Java).

Con ADO .NET se puede acceder a los datos de dos formas distintas:

- Acceso conectado: Acceso sólo de lectura con cursores unidireccionales ("firehose cursors"). La aplicación realiza una consulta y lee los datos conforme los va procesando con la ayuda de un objeto `DataReader`.
- Acceso desconectado: La aplicación ejecuta la consulta y almacena los resultados de la misma para procesarlos después accediendo a un objeto de tipo `DataSet`. De esta forma, se minimiza el tiempo que permanece abierta la conexión con la base de datos.

Al proporcionar conjuntos de datos de forma desconectada, se utilizan mejor los recursos de los servidores y se pueden construir sisyemas más escalables que con ADO (que mantenía abierta la conexión con la base de datos la mayor parte del tiempo). Este enfoque resulta más adecuado en sistemas distribuidos como Internet.

Arquitectura ADO.NET

El funcionamiento de ADO.NET se basa esencialmente en utilizar los siguientes componentes:

- **Data Provider** (proveedor de datos): Proporciona un acceso uniforme a conjuntos de datos (bases de datos relacionales o información ID3 de ficheros MP3). Su papel el similar al de un controlador ODBC o JDBC.
- **DataSet**: El componente más importante, puede almacenar datos provenientes de múltiples consultas (esto es, múltiples tablas).
- **DataAdapter**: Sirve de enlace entre el contenedor de conjuntos de datos (DataSet) y la base de datos (Data Provider).

Los componentes anteriores se completan con **DataReader** (para realizae eficientemente lecturas de grandes cantidades de datos que no caben en memoria), **DataRelation** (la forma de establecer una reunión entre dos tablas), **Connection** (utilizada por DataAdapter para conectarse a la base de datos) y **Command** (que permite especificar las órdenes, generalmente en SQL, que nos permiten consultar y modificar el contenido de la base de datos: select, insert, delete y update).

Un proveedor de datos debe proporcionar una implementación de Connection, Command, DataAdapter y DataReader.

El modo de funcionamiento típico de ADO.NET es el siguiente:

- Se crean un objeto Connection especificando la cadena de conexión.
- Se crea un DataAdapter.
- Se crea un objeto Command asociado al DataAdapter, con la conexión adecuada y la sentencia SQL que haya de ejecutarse.
- Se crea un DataSet donde almacenar los datos.
- Se abre la conexión.
- Se rellena el DataSet con datos a través del DataAdapter.
- Se cierra la conexión.
- Se trabaja con los datos almacenados en el DataSet.

Como los conjuntos de datos se almacenan en memoria y trabaja con ellos de forma desconectada, cuando hagamos cambios sobre ellos (inserciones, borrados o actualizaciones) debemos actualizar el contenido de la base de datos llamando al método Update del DataAdapter y, posteriormente, confirmar los cambios realizados en el DataSet (con AcceptChanges) o deshacerlos (con RejectChanges).

Clases ADO.NET

ADO .NET define una serie de interfaces que proporcionan la funcionalidad básica común a las distintas fuentes de datos accesibles a través de ADO .NET. La implementación de estos interfaces por parte de cada proveedor proporciona acceso a un tipo concreto de fuentes de datos y puede incluir propiedades y métodos adicionales.

Interfaz IDbConnection

Establece una sesión con una fuente de datos. Permite abrir y cerrar conexiones, así como comenzar transacciones (que se finalizan con los métodos `Commit` y `Rollback` de `IDbTransaction`. Las clases `SqlDbConnection` y `OleDbConnection` implementan el interfaz de `IDbConnection`.

Interfaz IDbCommand

Representa una sentencia que se envía a una fuente de datos (usualmente en SQL, aunque no necesariemente). Las clases `SqlDbCommand` y `OleDbCommand` implementan el interfaz de `IDbCommand`.

`IDbCommand` nos permite definir la sentencia que se ha de ejecutar, ejecutar la sentencia, pasarle parámetros y prepararla (crear una versión "compilada" de la misma para que su ejecución sea más eficiente cuando ha de repetirse varias veces). El método `ExecuteReader` devuelve un conjunto de tuplas (véase el interfaz `IDataReader`), mientras que `ExecuteScalar` devuelve un valor único (p.ej. ejecución de procedimientos almacenados) y `ExecuteNonQuery` no devuelve nada (p.ej. borrados y actualizaciones).

Interfaz IDataReader

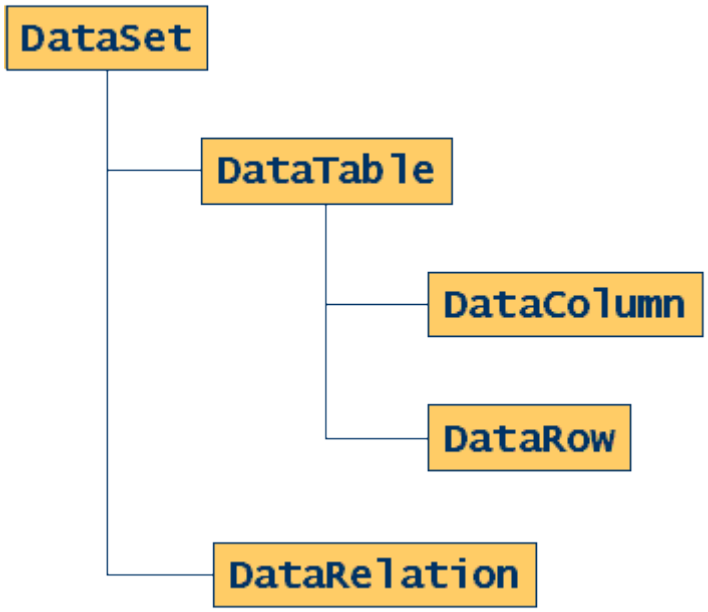
Proporciona acceso secuencial de sólo lectura a una fuente de datos. Las clases `SqlDataReader` y `OleDbDataReader` implementan el interfaz de `IDataReader`. Al utilizar un objeto `IDataReader`, las operaciones sobre la conexión `IDbConnection` quedan deshabilitadas hasta que se cierre el objeto `IDataReader`.

Clase DataSet

Un objeto `DataSet` encapsula un conjunto de tablas independientemente de su procedencia y mantiene las relaciones existentes entre las tablas. El contenido de un `DataSet` puede serializarse en formato XML. Además, se permite la modificación dinámica de los datos y metadatos del conjunto de datos representado por el objeto `DataSet`.

El interfaz `IDataAdapter` implementado `OleDbDataAdapter` y `SqlDataAdapter` se utiliza para construir el conjunto de datos y actualizarlo cuando sea necesario. Los conjuntos de datos con los que se trabaja de esta forma utilizan un enfoque asíncrono en el que no se mantiene abierta la conexión con la base de datos a la que se está accediendo. Al trabajar

con conjuntos de datos de esta forma, se dispone de un superconjunto de los comandos que se permiten cuando se emplea el interfaz `IDataReader`. De hecho se pueden realizar operaciones de consulta (`select`), inserción (`insert`), actualización (`update`) y borrado (`delete`).



Clase DataTable

Representa una tabla en memoria (Columns & Rows) cuyo esquema viene definido por su colección de columnas `Columns`. La integridad de los datos se conserva gracias a objetos que representan restricciones (`Constraint`) y dispone de eventos públicos que se producen al realizar operaciones sobre la tabla (p.ej. modificación o eliminación de filas).

Clase DataColumn

Define el tipo de una columna de una tabla (vía su propiedad `DataType`) e incluye las restricciones (`Constraints`) y las relaciones (`Relations`) que afectan a la columna. Además, posee propiedades útiles como `AllowNull`, `Unique` o `ReadOnly`.

Clase DataRow

Representa los datos de una tabla (almacenados en la colección `Rows` de un objeto `DataTable`), de acuerdo con el esquema definido por las columnas de la tabla (`Columns`). Además, incluye propiedades para determinar el estado de una fila/tupla particular (p.ej. nuevo, cambiado, borrado, etc.).

Clase DataRelation

Relaciona dos `DataTables` vía `DataColumns` y sirve para mantener restricciones de integridad referencial. Obviamente, el tipo de las columnas relacionadas ha de ser idéntico.

Para acceder a los datos relacionados con un registro concreto basta con emplear el método `GetChildRecords` de la tupla correspondiente (`DataRow`).

Transacciones en ADO.NET

Las transacciones son conjuntos de operaciones que han de efectuarse de forma atómica. La acidez de una transacción hace referencia a sus propiedades deseables: atomicidad, consistencia, aislamiento y durabilidad (*ACID = Atomicity, Consistency, Isolation, Durability*).

En ADO.NET, los límites de las transacciones se indican manualmente. Los objetos de la clase `Connection` tienen un método `BeginTransaction` que devuelve una transacción (objeto de tipo `Transaction`). La transacción finaliza cuando se llama al método `Commit` o `Rollback` del objeto `Transaction` devuelto por `BeginTransaction`.

Data Binding

Éste es el nombre por el que se conoce el mecanismo que nos permite asociar el contenido de un conjunto de datos a los controles de la interfaz de nuestra aplicación, algo que facilitan los entornos de programación visual (como es el caso del Visual Studio .NET).

Data Binding en Visual Studio .NET

Se puede optar por asistentes del tipo de **DataForm wizard**... o, cuando éstos no nos ofrecen la funcionalidad suficiente para nuestras aplicaciones, podemos programarlo nosotros mismos (algo que no es difícil y nos da bastante más control sobre nuestra aplicación):

- Se crean los objetos ADO.NET necesarios (Connection, DataSet, Command y DataAdapter).
- Se enlazan los controles de la interfaz con las columnas del DataSet, lo que se consigue añadiendo un objeto `System.Windows.Forms.Binding` a la propiedad `DataBindings` del control. Los objetos de tipo `Binding` nos permiten enlazar una propiedad de un control a una columna de un `DataSet`.
- Se implementa la forma de recorrer los registros, haciendo uso del objeto `BindingContext` asociado al formulario (cuyas propiedades `Position` y `Count` nos permiten movernos por el conjunto de datos).

- Se implementan las operaciones sobre la base de datos (con los comandos asociados al `DataAdapter`.

NOTA: Cuando los conjuntos de datos sean grandes, es recomendable utilizar ADO.NET con paginación para no tener que leer todos los datos de la base de datos (véase "ADO.NET data paging"). En situaciones como ésa, también suele ser recomendable añadir capacidades de búsqueda a nuestras aplicaciones (véase el método `Find`).