UTN
Facultad Regional Córdoba

TUP

Tecnicatura Universitaria en Programación

PROGRAMACIÓN I

Unidad Temática N°5:
Introducción a ADO.NET

Material Teórico
1° Año – 1° Cuatrimestre

Índice

INTRODUCCIÓN A 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).

Imagen 1: Elaboración propia

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:

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

- **DAO (Data Access Objects):** Interfaz para programar con bases de datos JET/ISAM, utiliza automatización OLE y ActiveX.
- **RDO (Remote Data Objects):** Fuertemente acoplado a ODBC, orientado al desarrollo de aplicaciones cliente/servidor.
- **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++.
- **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).

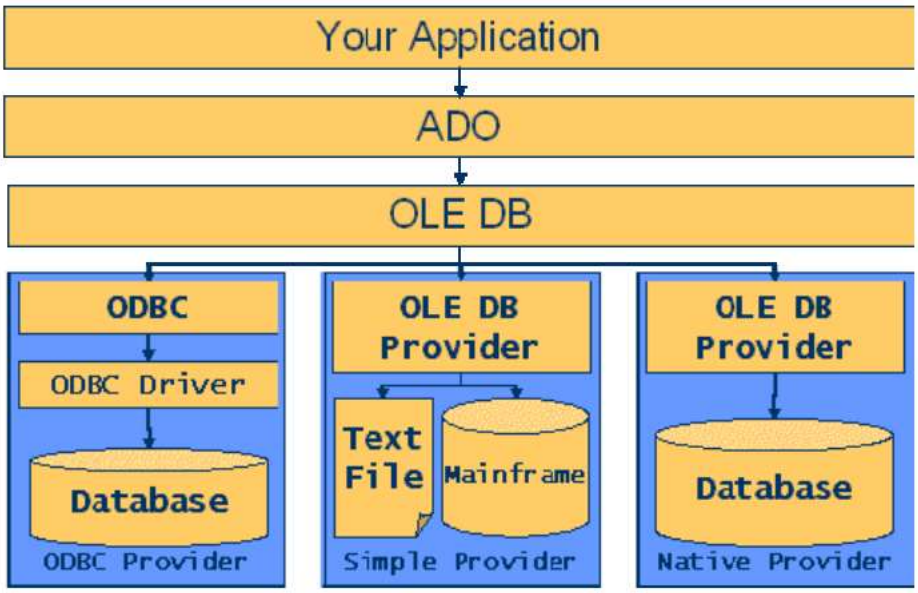

Imagen 2: Elaboración propia

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).

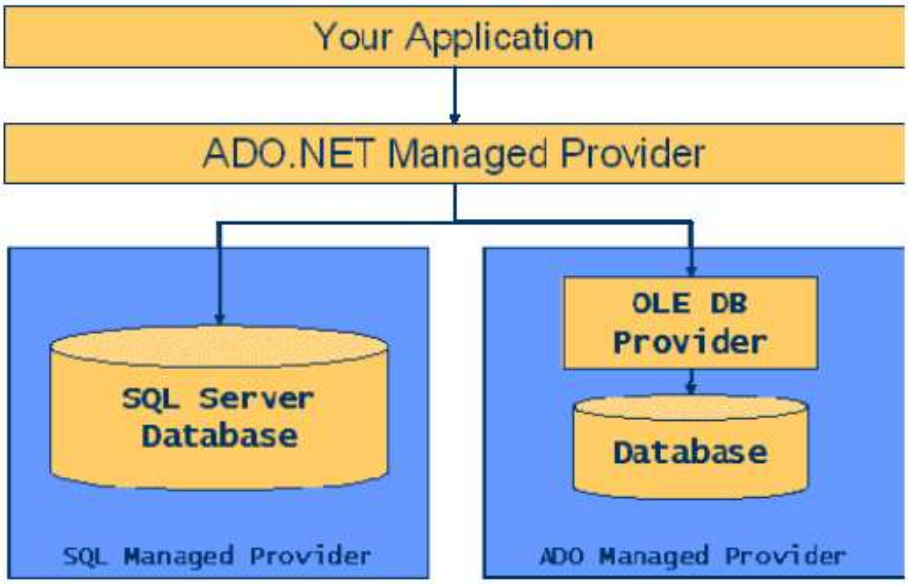

Imagen 3: Elaboración propia

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 sistemas 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 realizar 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 SLQ 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 se 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 (RejectCharges).

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.

La implementación de cada uno de los proveedores de datos es responsabilidad del fabricante de cada DBMS. Así, el creador de SQL Server ofrece un el conjunto de clases exigidas por .net, el creador de Oracle provee su propio conjunto de clases, y así por cada uno de los motores que pretendan que los programas .net puedan acceder a sus datos.

Una interfaz es una definición de clases, métodos y propiedades similares a las clases abstractas, en las que todos los métodos son abstractos y por lo tanto deben ser redefinidos en las clases implementadoras. Para evitar que deba modificarse el código de los programas si se requiere cambiar de motor de bases de datos, la librería OLEDB exige que cada proveedor implemente un conjunto de interfaces, de forma tal que los programadores utilicen los métodos y propiedades de las interfaces.

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. SQLConnection sirve para conectarse a bases de datos SQL Server y OleDBConnection a bases de datos Access. De la misma forma se pueden obtener librerías para conectase a otros motores, por ejemplo, OracleConnection o MySqlConnection.

El uso de esta interfaz es muy simple, para establecer una conexión a una base de datos debe crearse una instancia de la clase concreta correspondiente al motor y pasarle al constructor una cadena denominada cadena de conexión. Esta cadena tiene un formato específico y dependiente del proveedor, y posee todos los datos necesarios para establecer la conexión, entre ellos el nombre del servidor o del archivo, el nombre del usuario y su contraseña.

Luego de construido el objeto connection debe invocarse a su método Open() y en ese momento se intenta la conexión. El proceso de conexión con la base de datos puede fallar por diversos motivos. El servidor puede estar inaccesible, apagado o con problemas de la red. Así mismo puede haber errores en los datos de autenticación, es decir, usuario y contraseña.

En todos esos casos, el método Open no finaliza correctamente y lanza una excepción, la que puede saltar a un bloque catch si existe o interrumpe el programa si no. Por lo tanto, si el programa continúa la ejecución desde la línea siguiente significa que la conexión se abrió correctamente.

Con la conexión abierta ya se pueden comenzar a enviar sentencias SQL a la base de datos, lo que se logra con el uso de los objetos de la interfaz IDbCommand.

Finalizada la ejecución de las sentencias la conexión debe cerrarse mediante una llamada al método Close().

Interfaz IDbCommand

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

Los objetos que implementan IDbCommand poseen un constructor con dos parámetros. El primero es una cadena que contiene el texto de la sentencia SQL que se desea ejecutar. Mientras que el segundo es un objeto Connection correctamente abierto. Esto además significa que si se desea ejecutar más de una sentencia deben crearse tantos objetos Command como sentencias se requieran.

Para efectivamente ejecutar las sentencias deben diferenciarse en dos tipos. Por un lado, se encuentran las sentencias que devuelven filas y por otro las que no lo hacen. Las sentencias que devuelven filas son únicamente las sentencias SELECT y aquellos procedimientos almacenados que finalizan con una sentencia SELECT. Todas las otras no devuelven filas, en este grupo se encuentran INSERT, UPDATE, DELETE y todas las del subconjunto DDL.

Para ejecutar una sentencia del primer grupo se debe invocar al método ExecuteReader(), que luego de enviar la sentencia al motor retorna un objeto de la interfaz IDbReader que permite recorrer el conjunto de filas retornado para obtener sus datos.

Por otro lado, para ejecutar las sentencias que no devuelven filas se invoca al método ExecuteNonQuery() el cual retorna en cambio un número entero que

indica la cantidad de filas afectadas por la ejecución. Por ejemplo, un insert simple devuelve 1 si puede insertar la fila o 0 si no se pudo, pero los DELETE o los UPDATE retornan la cantidad de filas afectadas, es decir, las que cumplieron con la condición de la cláusula WHERE y por lo tanto fueron eliminadas o modificadas.

Adicionalmente, un método menos usado es ExecuteScalar(), el mismo retorna únicamente el valor de la primera columna de la primera fila del conjunto de filas retornado. Es útil principalmente al utilizar funciones sumarizadas como MAX o AVG ya que permite obtener el valor calculado por las mismas, pero sin necesidad de utilizar un objeto DataReader. Esto es porque el método ExecuteScalar devuelve directamente el valor calculado para asignar en una variable.

Interfaz IDataParameter

Al utilizar los objetos Command surge inevitablemente un inconveniente con aquellas sentencias que posean valores diferentes en cada ejecución. El caso más obvio es con la sentencia INSERT, ya que cada vez que se la invoca se intenta insertar datos diferentes.

Frente a ello una práctica muy habitual es la de armar toda la sentencia completa concatenando aquellas porciones de la misma que se ejecutan siempre con las variables que contienen los datos que cambian en cada ejecución. Suponiendo una tabla Personas con columnas Nombre y Apellido, el programa quedaría como:

```
string sentencia = "INSERT into Personas values ('" + nombre + "', '" + apellido + "')";
```

Esta estrategia tiene dos defectos importantes. Por un lado, es muy dificultoso escribirla, hay que tener mucho cuidado con las comillas dobles de C# pero también agregar correctamente las comillas simples del lenguaje SQL, cuidando las comas, paréntesis y demás símbolos. Pero por otro lado hay un riesgo muy alto porque si un usuario malicioso ingresa una sentencia sql en las variables nombre o apellido, la misma puede llegar a ejecutarse, con consecuencias indeseadas o hasta catastróficas.

Para evitar ambas complicaciones el uso más adecuado del objeto Command es el de indicar dentro de la consulta SQL aquellos datos que van a ser reemplazados por valores concretos usando parámetros:

```
string sentencia = "INSERT into Personas values (@nombre,@apellido)";
```

En la misma los parámetros @nombre y @apellido deben ser reemplazados por valores concretos antes de efectivamente ejecutar la sentencia. Para ello se

necesita instanciar un objeto de la interfaz IDataParameter por cada uno de los mismos y agregarlos a la colección Parameters del objeto Command.

```
SqlCommand comm = new SqlCommand(sentencia, conexion);
comm.Parameters.AddWithValue(@nombre, variableNombre);
comm.Parameters.AddWithValue(@apellido, variableApellido);

comm.ExecuteNonQuery();
```

Otro Ejemplo:

```
string connectionString = "Persist Security Info=False;" +
     "User ID=sa;Initial Catalog=MYDB;" +
      "Data Source=MYSERVER";
SqlConnection connection = new SqlConnection(connectionString);

// Ejecución de sentencias SQL
// ---------------------------
string sqlInsert = "INSERT INTO Department(DepartmentName) VALUES (@DepartmentName)";
SqlCommand insertCommand = new SqlCommand(sqlInsert, connection);

SqlParameter param = insertCommand.Parameters.Add (
      new SqlParameter("@DepartmentName", SqlDbType.VarChar, 100));
param.Value = ...
connection.Open();
insertCommand.ExecuteNonQuery();
connection.Close();

// Llamadas a procedimientos almacenados
// -------------------------------------
// C#
string spName = "CREATE_DEPARTMENT"
SqlCommand command = new SqlCommand(spName, connection);
command.CommandType = CommandType.StoredProcedure;
SqlParameter in = command.Parameters.Add (
new SqlParameter("@DepartmentName", SqlDbType.VarChar, 100));
in.Value = ...
```

```
SqlParameter out = command.Parameters.Add (
new SqlParameter("RETVAL", SqlDbType.Int));
out.Direction = ParameterDirection.ReturnValue;

connection.Open();
insertCommand.ExecuteNonQuery();
int newID = command.Parameters("RETVAL").Value;
connection.Close();

// SQL Server
// ----------
CREATE TABLE [dbo].[Department] (
  [DepartmentID] [int] IDENTITY (1, 1) NOT NULL ,
  [DepartmentName] [varchar] (100),
  [CreationDate] [datetime] NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Department] WITH NOCHECK ADD
  CONSTRAINT [PK_Department] PRIMARY KEY CLUSTERED
  (
 [DepartmentID]
  ) ON [PRIMARY]
GO

CREATE PROCEDURE dbo.CreateDepartment
@DepartmentName varchar(100),
AS
  INSERT INTO Department (DepartmentName, CreationDate)
  VALUES (@DepartmentName, GetDate())
  RETURN scope_identity()
GO
```

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.

Cuando un Command ejecuta una sentencia que devuelve filas con el método ExecuteReader, se retorna un objeto de esta interfaz IDataReader ya creado y por lo tanto debe ser asignado a una referencia de ese tipo.

El objetivo del DataReader es el de permitir recorrer en forma secuencial todo el conjunto de filas retornado, de a una fila por vez. Luego, por cada fila existen métodos para acceder a cada una de las columnas. Para esto, el DataReader se encuentra en todo momento recorriendo una fila y para avanzar a la siguiente se debe invocar al método Read().

El método Read retorna un booleano indicando si pudo avanzar a la siguiente fila, en tal caso retorna true. Cuando se llega a a la última fila y se la procesa, la próxima llamada a Read retorna false indicando que ya no hay más filas.
Para acceder a la primera fila, sin embargo, es necesario invocar una vez a Read(). Esta lógica simplifica mucho el recorrido, ya que se puede utilizar el retorno como condición de corte de un ciclo while:

```
SqlDataReader dr = comm.ExecuteReader();

while (dr.Read()) {

// Este ciclo da una vuelta por cada fila.

}
```

Para acceder a cada columna se dispone de un conjunto de métodos llamados GetXXX donde XXX es el nombre de un tipo de datos. Debe invocarse el correspondiente al tipo de datos de la columna que se intenta leer, la cual es indicada mediante un número entero que señala el orden de la columna en la lista de columnas de la sentencia SELECT.

Por ejemplo, si se ejecuta la sentencia “SELECT Id, Nombre FROM Personas”, para obtener el Id se ejecuta el método GetInt32(0), ya que es de tipo int y es la primera columna de la lista del select. Consecuentemente, para obtener el nombre se ejecuta GetString(1).

Los métodos GetXXX retornan un dato del tipo correcto, el cual puede ser asignado directamente en una variable sin necesidad de casting.

Ejemplo:

```
string connectionString = "Provider=SQLOLEDB.1;" +
     "User ID=sa;Initial Catalog=Northwind;" +
     "Data Source=MYSERVER";
OleDbConnection connection = new OleDbConnection(connectionString);
string sqlQuery = "SELECT CompanyName FROM Customers";
OleDbCommand myCommand = new OleDbCommand(sqlQuery, connection);
connection.Open();
OleDbDataReader myReader = myCommand.ExecuteReader();
while (myReader.Read()) {
   Console.WriteLine(myReader.GetString(0));
}
myReader.Close();
connection.Close();
```

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 súper conjunto 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).

```
string connectionString = "Persist Security Info=False;" +
     "User ID=sa;Initial Catalog=Northwind;" +
     "Data Source=MYSERVER";
   SqlConnection connection = new SqlConnection(connectionString);
   SqlDataAdapter myDataAdapter = new SqlDataAdapter();
   DataSet myDataSet = new DataSet();
```

```
string sqlQuery = "SELECT * FROM Customers";

myDataAdapter.SelectCommand = new SqlCommand(sqlQuery, connection);


connection.Open();

myDataAdapter.Fill(myDataSet);


conn.Close();
```

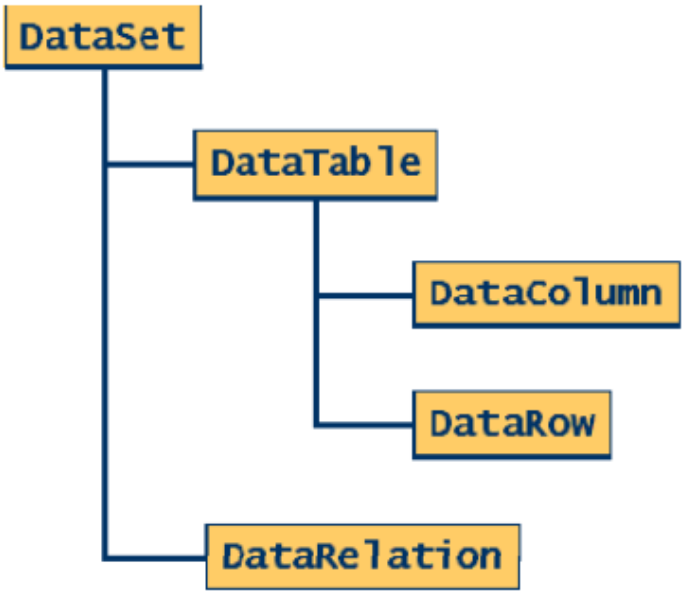

Imagen 4: Elaboración propia

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).

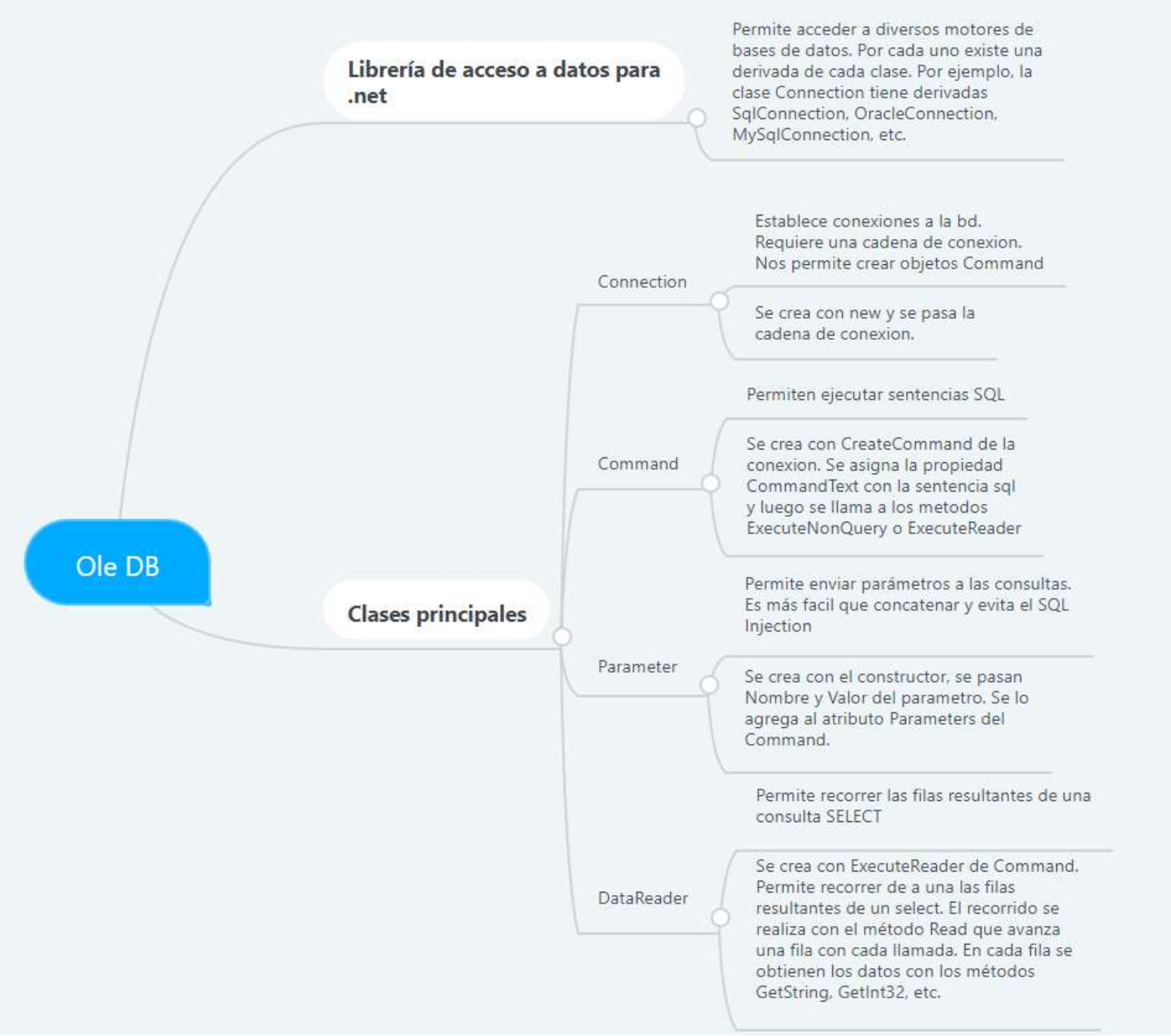

Imagen 5: Elaboración propia

BIBLIOGRAFÍA

Bishop, P. (1992) Fundamentos de Informática. Anaya.

Brookshear, G. (1994) Computer Sciense: An Overview. Benjamin/Cummings.

De Miguel, P. (1994) Fundamentos de los Computadores. Paraninfo.

Joyanes, L. (1990) Problemas de Metodología de la Programación. McGraw Hill.

Joyanes, L. (1993) Fundamentos de Programación: Algoritmos y Estructura de Datos. McGraw Hill.

Norton, P. (1995) Introducción a la Computación. McGraw Hill.

Prieto, P. Lloris A. y Torres J.C. (1989) Introducción a la Informática. McGraw Hill.

Tucker, A. Bradley, W. Cupper, R y Garnick, D (1994) Fundamentos de Informática (Lógica, resolución de problemas, programas y computadoras). McGraw Hill.