Microsoft Visual Studio .net

Capítulo 05
ADO.Net

Agenda

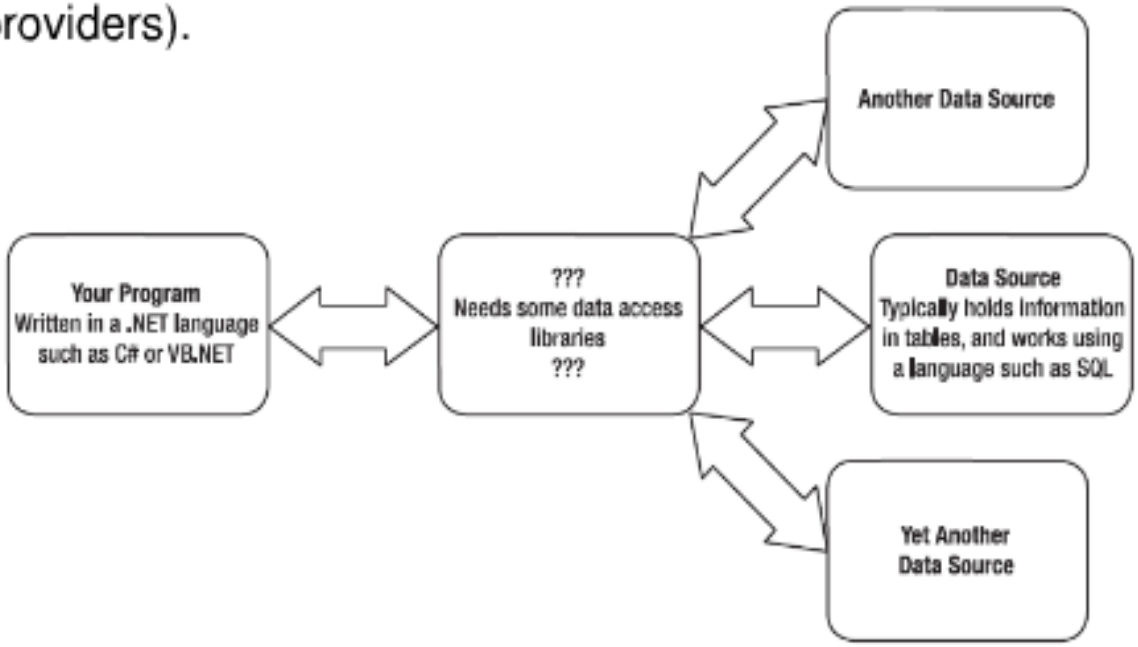
Microsoft Visual Studio .net

- Repaso sesión anterior.
- ADO.Net
  - Introducción
  - Arquitectura
  - Data Providers
    - Connection
    - Command
    - DataReader
    - DataAdapter
  - DataSet
  - LINQ
  - Clases Comunes
  - Escenario Conectado
  - Escenario Desconectado
  - Ejemplos
- Que veremos la próxima sesión?

ADO.Net

Microsoft Visual Studio.net

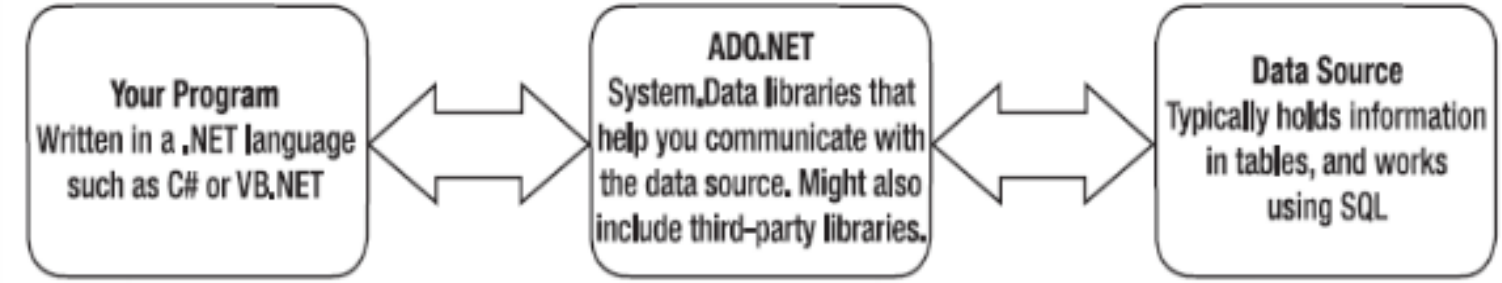
- Idea de acceso a datos universal
  - Comunicación entre los lenguajes de programación y las bases de datos o fuentes de datos.
  - Modelo de programación uniforme mediante una API estándar.
  - Implementaciones especializadas para las fuentes de datos (providers).



ADO.Net

Microsoft Visual Studio .net

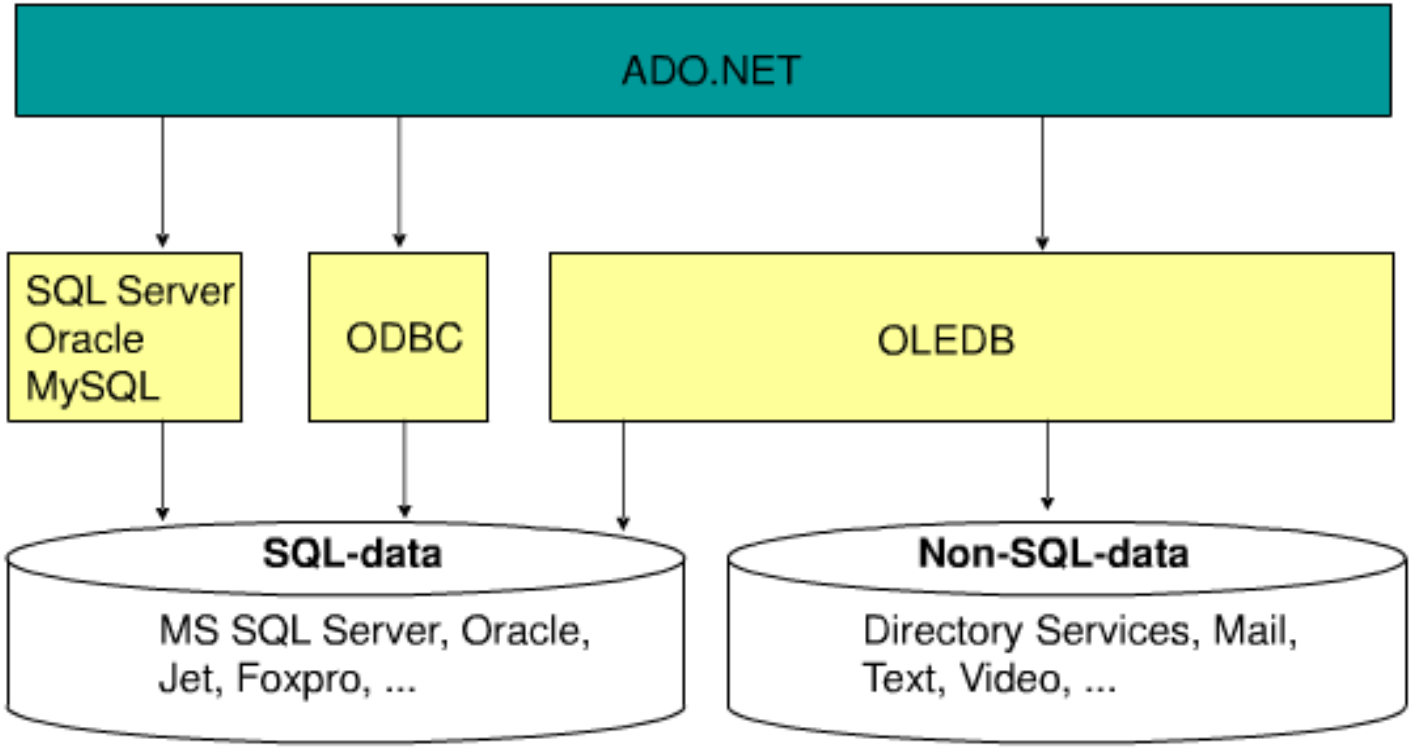
- Qué propone NET Framework?



ADO.Net

Microsoft Visual Studio .net

- ADO.NET



ADO.Net

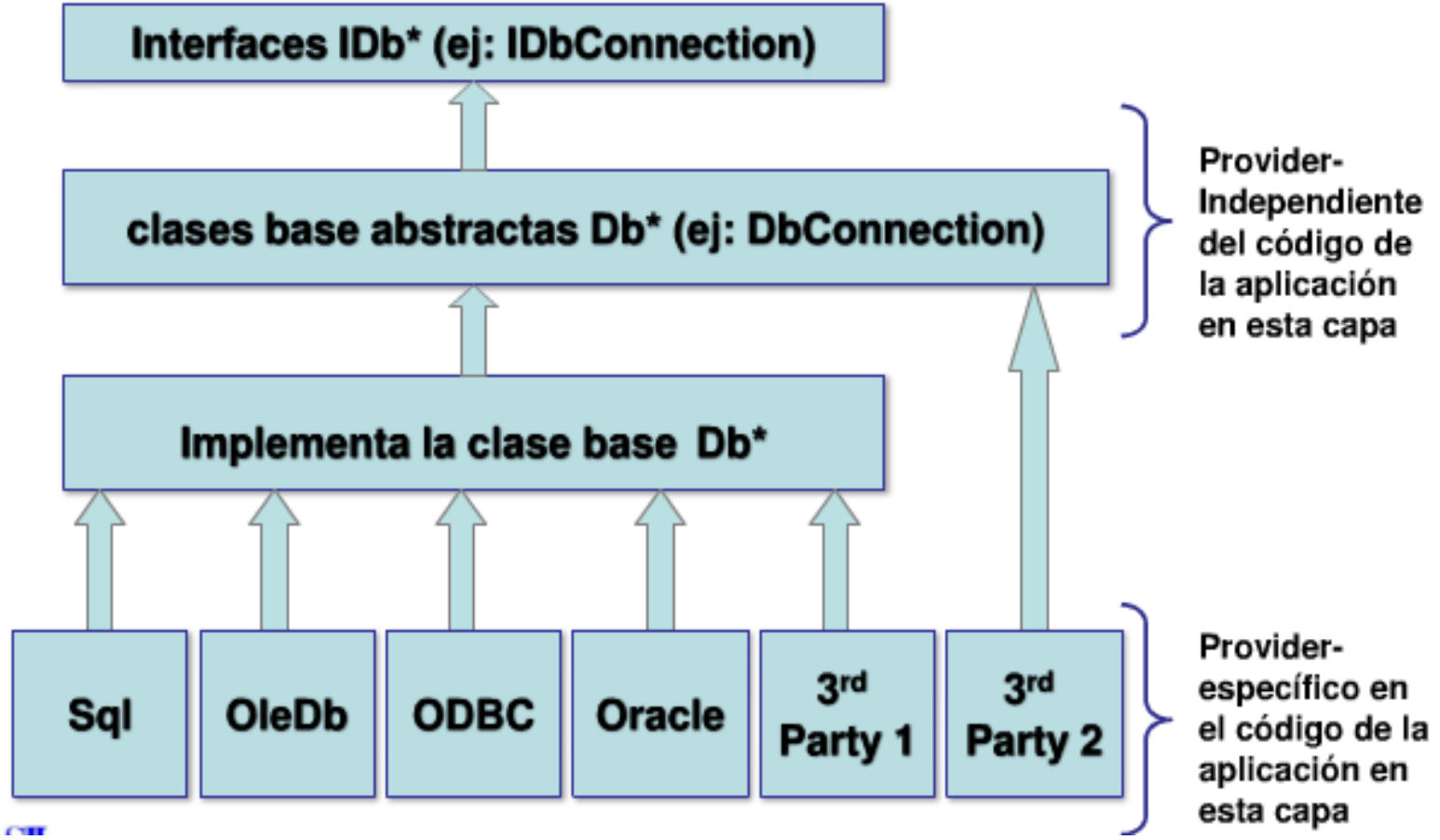
Microsoft Visual Studio.net

- Qué es ADO.Net?
  - Es parte fundamental de NET Framework.
  - Es un modelo de objetos desarrollado por Microsoft.
  - Provee servicios de acceso a datos para múltiples fuentes de datos.
    - Bases de datos relacionales (RDBMS)
    - XML
    - Archivos Excel, Access, etc…
  - Provee componentes para crear aplicaciones web y distribuidas sobre diversas fuentes de datos.
  - Se accede mediante el namespace System.Data.
  - Provee escenarios conectados y desconectados.
  - Es la evolución de Microsoft ADO (Activex Data Objects).
  - Provee una serie de interfaces y clases abstractas para implementaciones de terceros. (API Independendiente)

ADO.Net

Microsoft Visual Studio.net

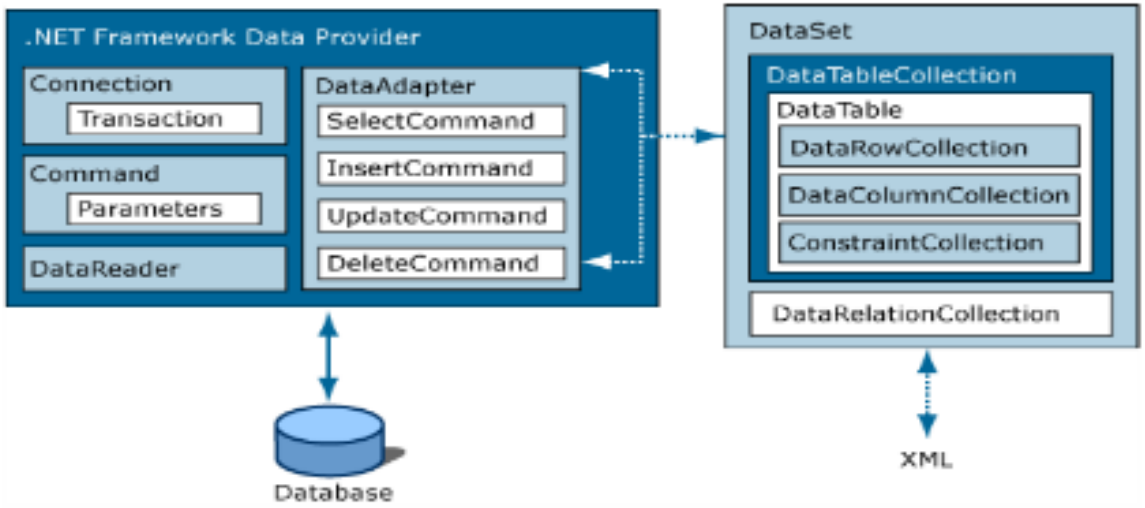
- ADO.Net 2.0 es una API Independiente



ADO.Net

Microsoft Visual Studio .net

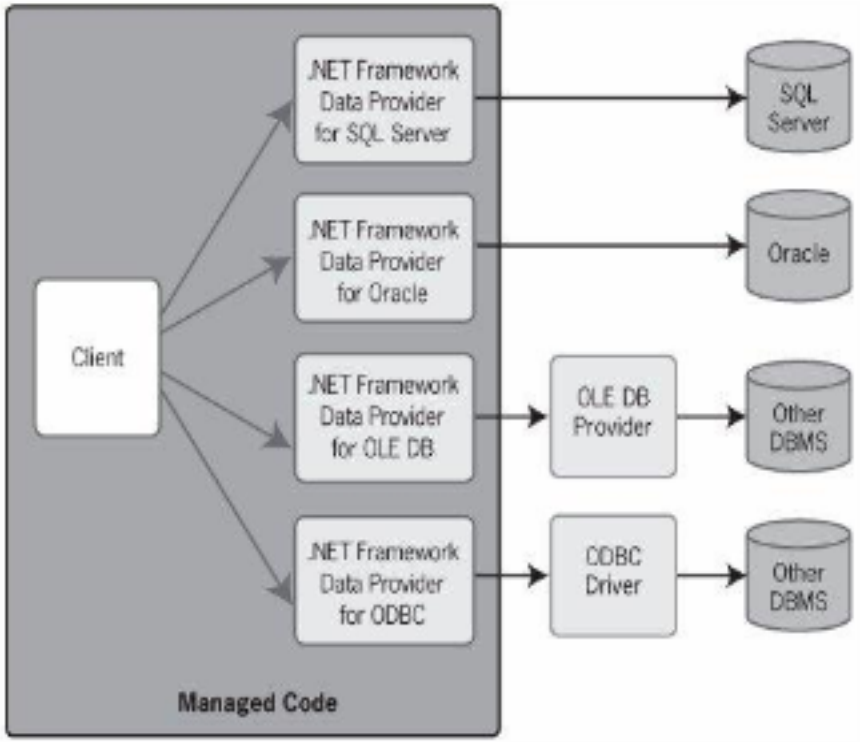
- **Arquitectura de ADO.NET** (2 Componentes fundamentales)
  - **Los NET Framework Data Providers**
    - Son un conjunto de objetos (Connection, Command, DataReader, DataAdapter).
    - Permiten la conexión a la fuente de datos.
    - Permiten obtener, modificar y datos.
    - Pueden ejecutar "Stored Procedures" y enviar/recibir parámetros
  - **El DataSet**
    - Son representaciones "en memoria" de los datos.
    - Independiente de la fuente de datos.
    - Puede utilizar múltiples fuentes de datos.
    - Proporciona un conjunto de objetos (DataTables, DataRow, etc.)
    - Utilizan XML para su persistencia.
    - Diseñado para escenarios "desconectados"



ADO.Net

Microsoft Visual Studio.net

- Data Providers
  - Cualquier aplicación que utiliza ADO.Net accede a la fuente de datos mediante un Provider.
  - Están programados en código manejado, pero pueden haber excepciones.
  - Permiten conectar a la BD, ejecutar comandos y obtener resultados
  - Está disponibles mediante sus respectivos Namespaces.
    - System.Data.SqlClient
    - System.Data.OracleClient
    - System.Data.OleDb
  - Cada Provider implementa interfaces para sus objetos.
  - Se pueden implementar Providers por terceros (MySQL, PostgreSQL, DB2, etc.)



ADO.Net

Microsoft Visual Studio.net

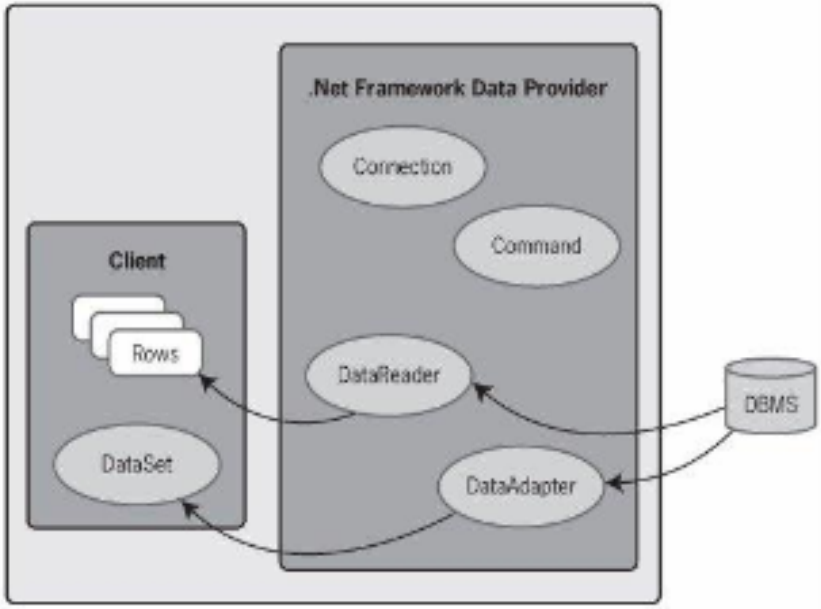
- Data Providers - Implementaciones

.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	Provides data access for Microsoft SQL Server version 7.0 or later. Uses the System.Data.SqlClient namespace.
.NET Framework Data Provider for OLE DB	For data sources exposed using OLE DB. Uses the System.Data.OleDb namespace.
.NET Framework Data Provider for ODBC	For data sources exposed using ODBC. Uses the System.Data.Odbc namespace.
.NET Framework Data Provider for Oracle	For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the System.Data.OracleClient namespace.

ADO.Net

Microsoft Visual Studio .net

- **Data Providers - Objetos**
  - **Connection** (Establece y libera conexiones a la BD)
  - **Command** (Ejecuta comandos como sentencias SQL, Stored Procedures, actualizaciones sobre la BD, etc.)
  - **DataReader** (Provee acceso directo, secuencial y de sólo lectura a los datos)
  - **DataAdapter** (Permite poblar DataSets y realizar sincronización sobre la BD)



- Data Providers – Objetos adicionales

Object	Description
Transaction	Enables you to enlist commands in transactions at the data source. The base class for all **Transaction** objects is the DbTransaction class.
CommandBuilder	A helper object that will automatically generate command properties of a **DataAdapter** or will derive parameter information from a stored procedure and populate the **Parameters** collection of a **Command** object. The base class for all **CommandBuilder** objects is the DbCommandBuilder class.
ConnectionStringBuilder	A helper object that provides a simple way to create and manage the contents of connection strings used by the **Connection** objects. The base class for all **ConnectionStringBuilder** objects is the DbConnectionStringBuilder class.
Parameter	Defines input, output, and return value parameters for commands and stored procedures. The base class for all **Parameter** objects is the DbParameter class.
Exception	Returned when an error is encountered at the data source. For an error encountered at the client, .NET Framework data providers throw a .NET Framework exception. The base class for all **Exception** objects is the DbException class.
Error	Exposes the information from a warning or error returned by a data source.
ClientPermission	Provided for .NET Framework data provider code access security attributes. The base class for all **ClientPermission** objects is the DBDataPermission class.

- Data Providers – Escogiendo el apropiado

Provider	Notes
.NET Framework Data Provider for SQL Server	Recommended for middle-tier applications using Microsoft SQL Server 7.0 or later. Recommended for single-tier applications using Microsoft Database Engine (MSDE) or SQL Server 7.0 or later. Recommended over use of the OLE DB Provider for SQL Server (SQLOLEDB) with the .NET Framework Data Provider for OLE DB. For SQL Server 6.5 and earlier, you must use the OLE DB Provider for SQL Server with the .NET Framework Data Provider for OLE DB.
.NET Framework Data Provider for OLE DB	Recommended for middle-tier applications using SQL Server 6.5 or earlier. For SQL Server 7.0 or later, the .NET Framework Data Provider for SQL Server is recommended. Also recommended for single-tier applications using Microsoft Access databases. Use of an Access database for a middle-tier application is not recommended.
.NET Framework Data Provider for ODBC	Recommended for middle and single-tier applications using ODBC data sources.
.NET Framework Data Provider for Oracle	Recommended for middle and single-tier applications using Oracle data sources.

- Data Providers
  - **Objeto Connection**
    - Provee la capacidad de conexión a cualquier fuente de datos.
    - Cada Provider proporciona su implementación:
    - La información necesaria para realizar la conexión se entrega mediante la propiedad **ConnectionString**.
    - Abre la conexión con la BD mediante el método **Open().**
    - Cierra la conexión con la BD mediante el método **Close()**.

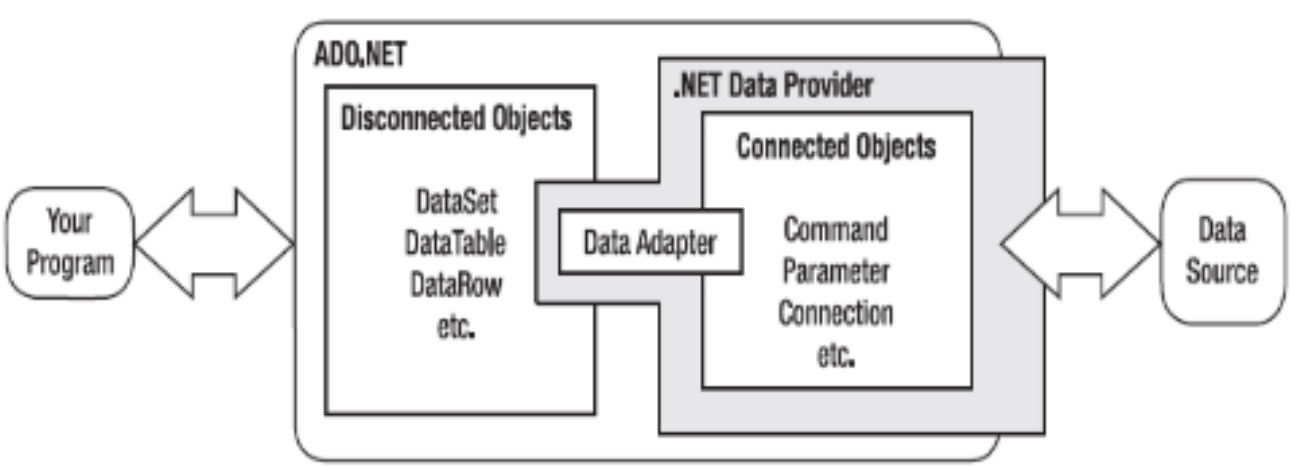
- Data Providers
  - **Objeto Command**
    - Una vez establecida la conexión, se utiliza el objeto Command.
    - Puede ejecutar consultas de inserción, modificación, eliminación de datos, etc.
    - Puede invocar “Stored Procedures”.
    - Maneja el envío y recepción de parámetros.
    - Métodos:
      - ExecuteReader() : ejecuta una consulta que devuelve un conjunto de datos (DataReader) .
      - ExecuteScalar(): ejecuta una consulta que devuelve un simple valor escalar.
      - ExecuteXmlReader(): ejecuta una consulta que devuelve un conjunto de datos en formato XML (Sólo para SqlCommand).
      - ExecuteNonQuery() : ejecuta una actualización de datos o similar.

- Data Providers
  - **Objeto DataReader**
    - Proporcionan acceso rápido a la data.
    - Sólo se mueven hacia adelante y es read-only.
    - Sólo pueden leer una fila a la vez.
    - Para leer una fila se utiliza el método Read().
    - Para leer los valores de la fila se utilizan los métodos Get: GetString(n), GetInt32(n), etc.

ADO.Net

Microsoft Visual Studio .net

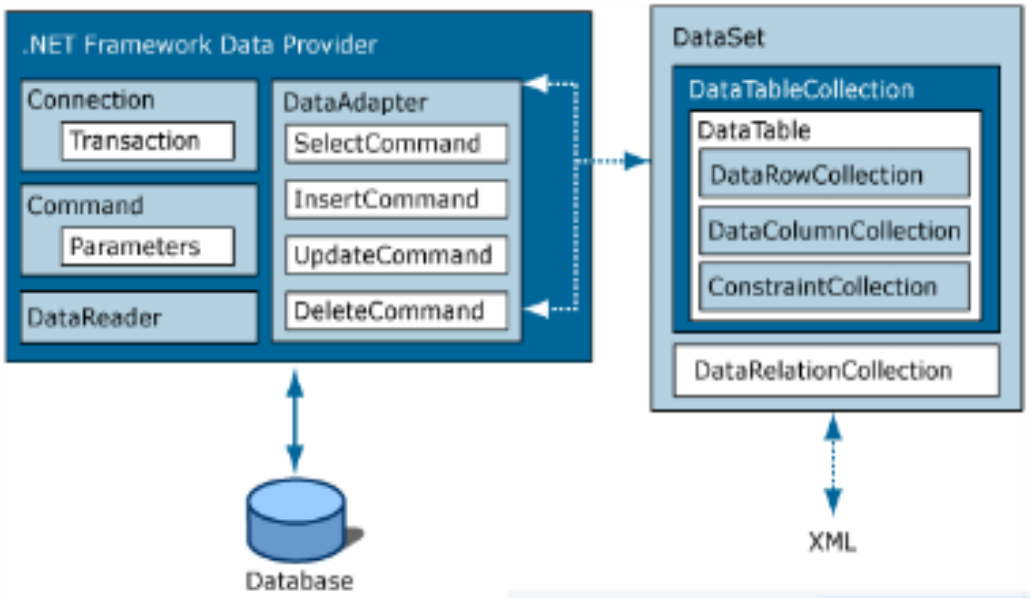
- Data Providers
  - **Objeto DataAdapter**
    - Es el puente entre las partes desconectadas y conectadas de ADO.Net
    - Utiliza los objetos Command para poblar los objetos desconectados (DataSet).
    - Utiliza los objetos Command para actualizar la BD en base a las modificaciones del DataSet.
    - El método Fill() puede llenar DataSets o DataTables.

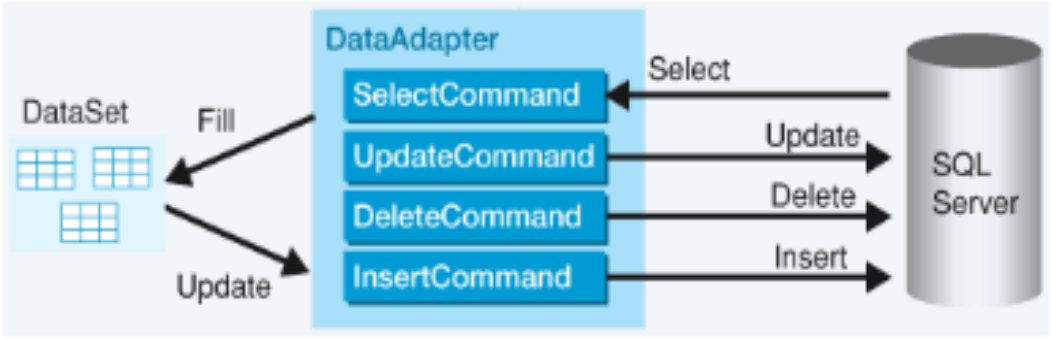


ADO.Net

Microsoft Visual Studio.net

- El DataAdapter (panorama)

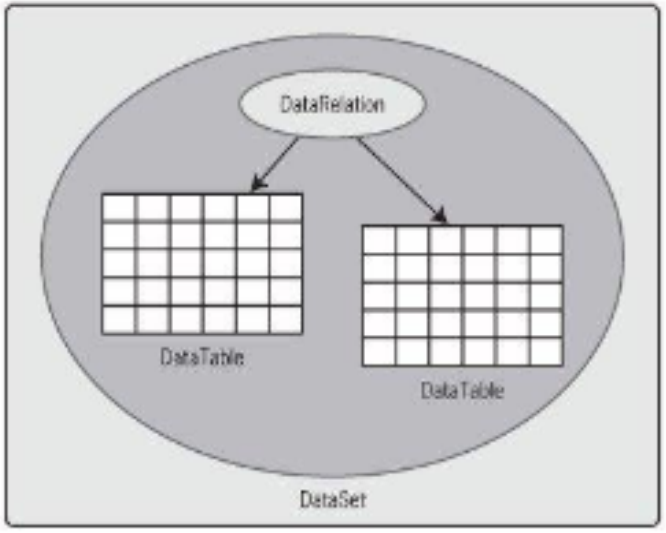




ADO.Net

Microsoft Visual Studio.net

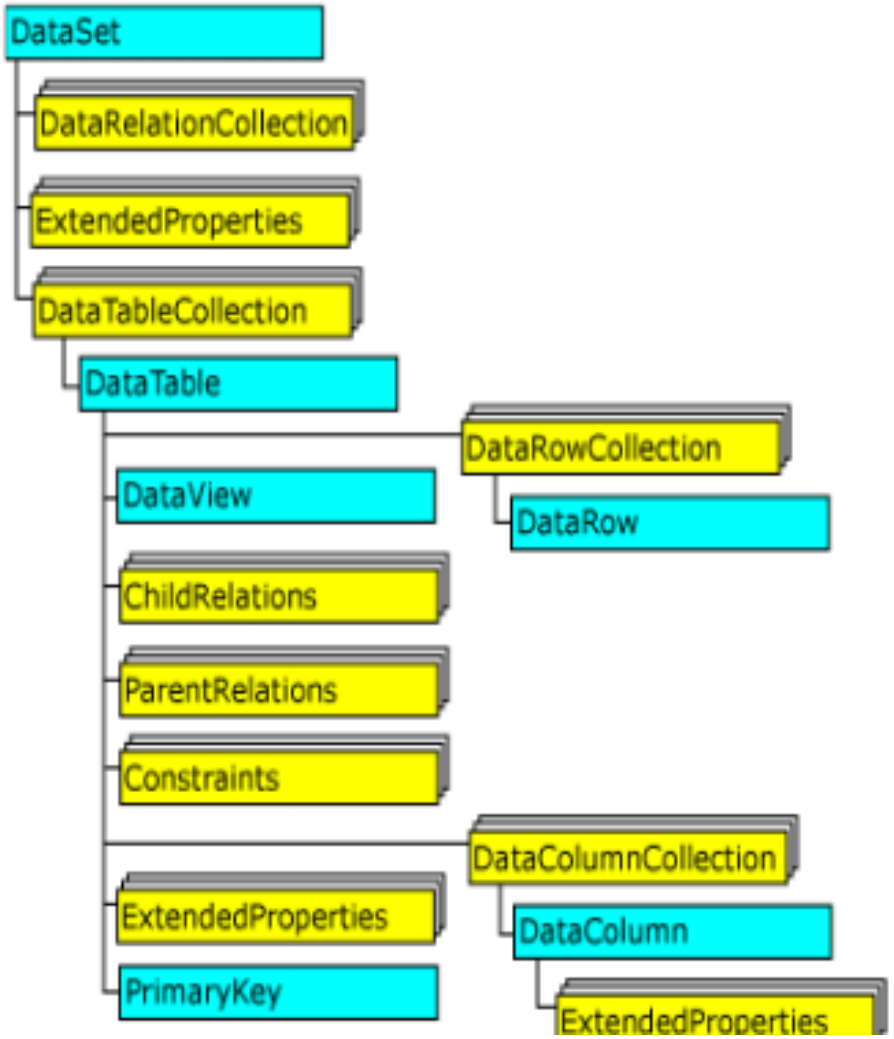
- **DataSet**
  - Representación en memoria de datos de múltiples fuentes.
  - Es independiente de la fuente de datos.
  - Permite acceso y manejo más flexible de la data que el DataReader.
  - Pueden modificar la data a diferencia de los DataReaders (read-only).
  - Pueden guardar los cambios en la BD, mediante el DataAdapter.
  - Contiene múltiples objetos (DataTable, DataColumn, DataView, etc.)
  - Utilizan XML para su persistencia
  - Diseñado para escenarios desconectados.
  - Es la base sobre la cual se construyen los DataSets tipados.



ADO.Net

Microsoft Visual Studio.net

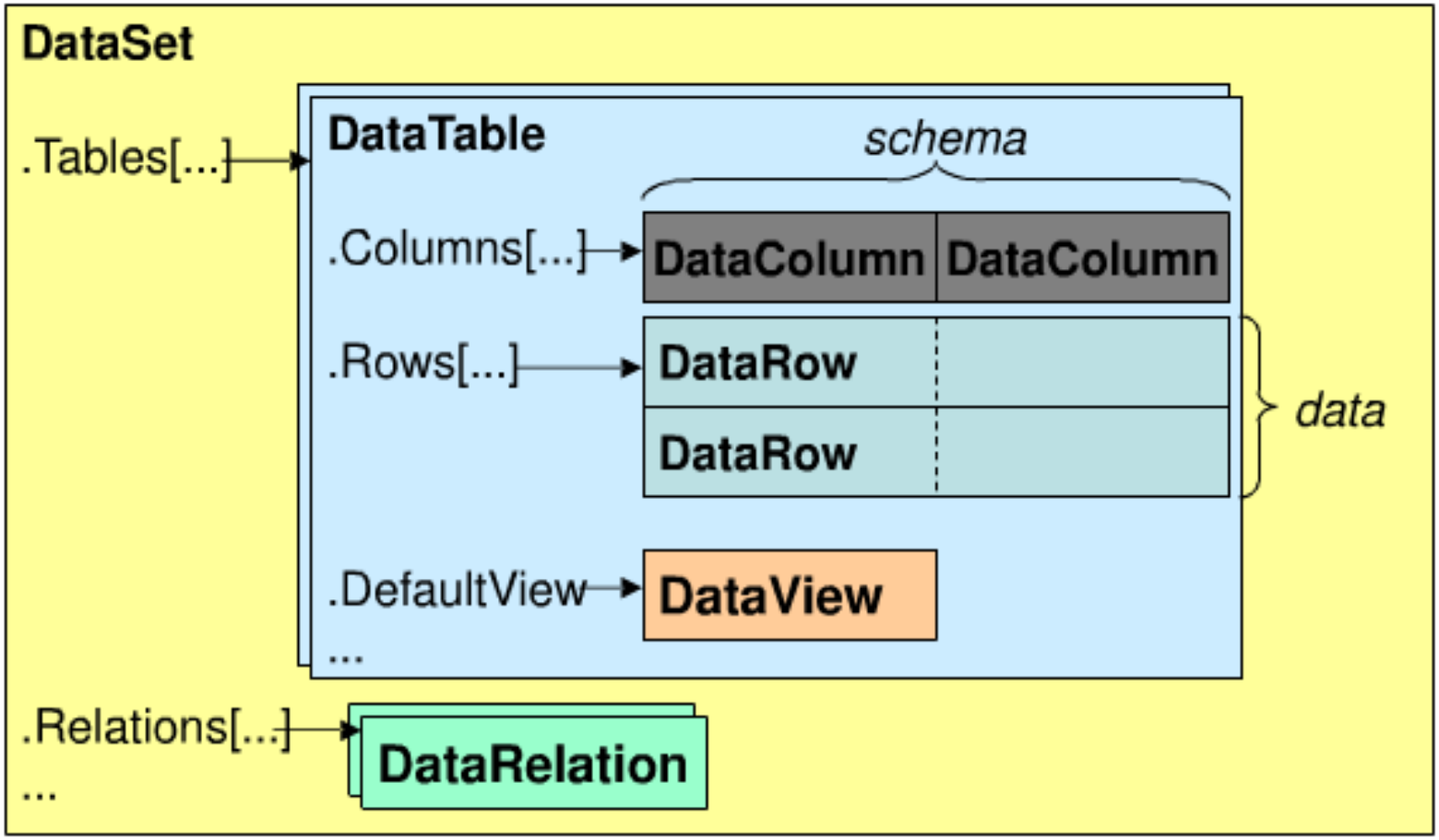
- DataSet
  - Modelo de objetos



ADO.Net

Microsoft Visual Studio.net

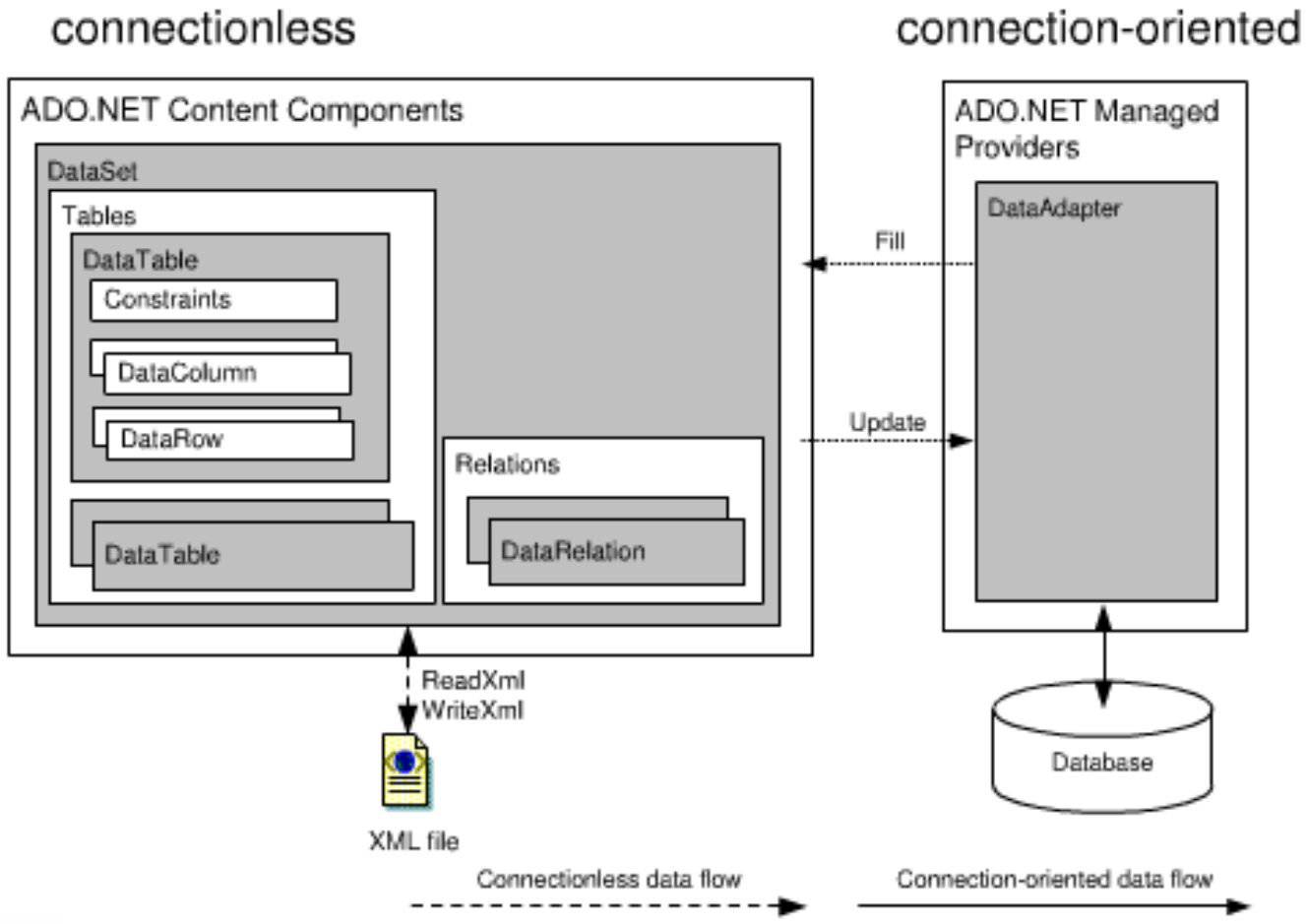
- Dataset (esquema)



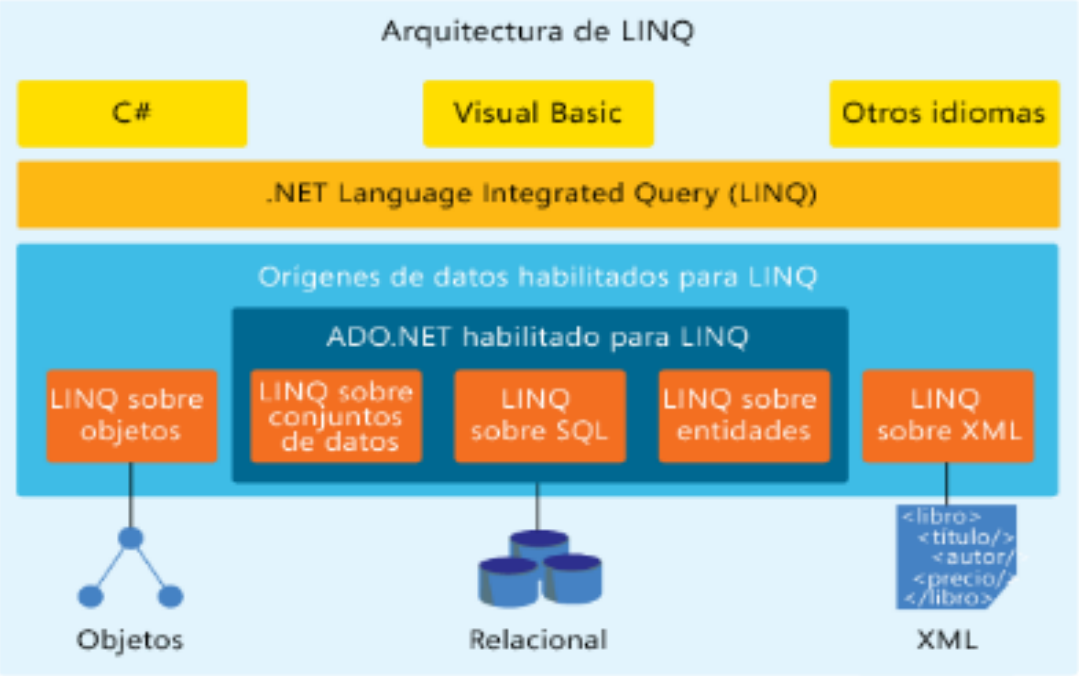
ADO.Net

Microsoft Visual Studio .net

- DataSet - DataAdapter



- LINQ (Net Framework 3.5)

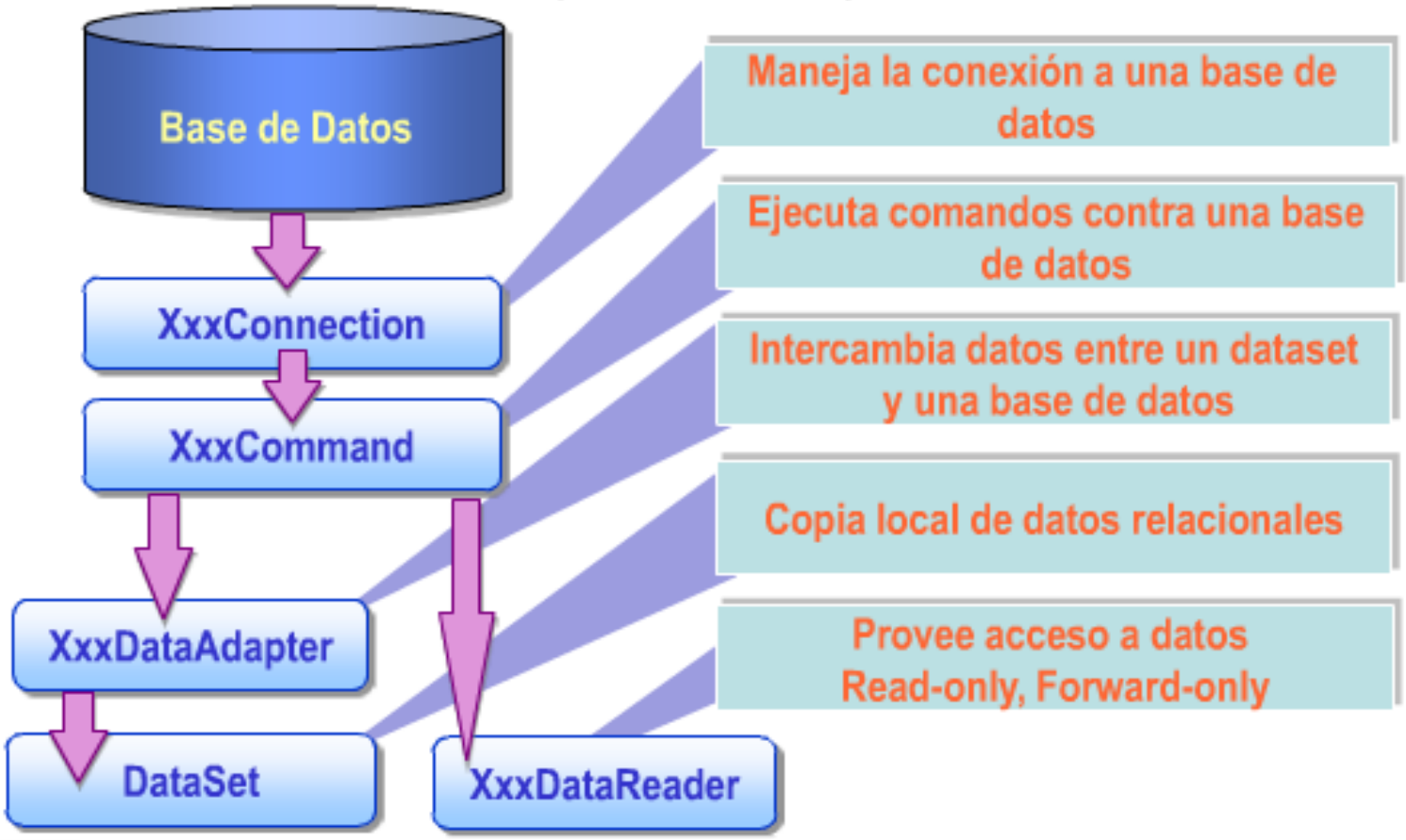



```
var overdrawnQuery = from account in db.Accounts
                     where account.Balance < 0
                     select new { account.Name, account.Address };
```

ADO.Net

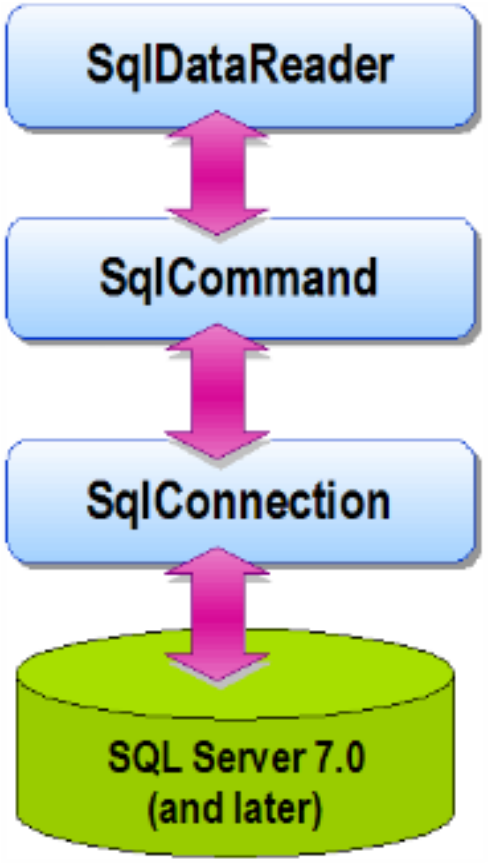
Microsoft Visual Studio .net

- Clases Comunes (Revisión)



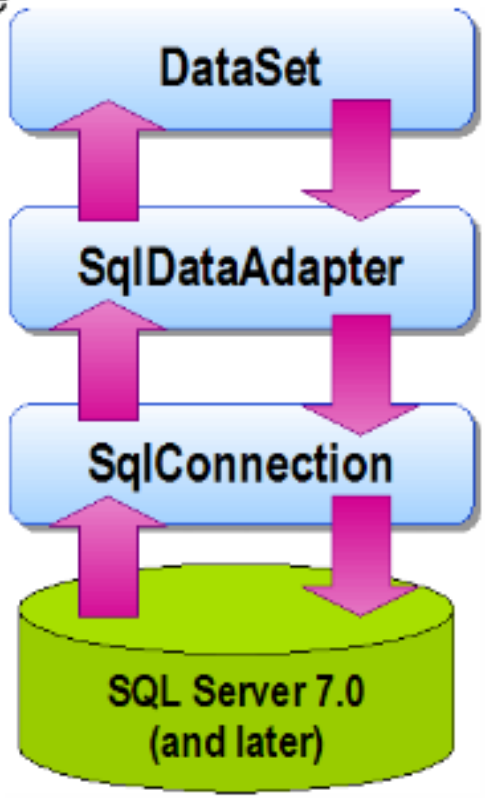
Acceso a Datos – Escenario Conectado

- En un escenario conectado, los recursos se mantienen en el servidor hasta que la conexión se cierra.
- Pasos:
  1. Abrir Conexión
  2. Ejecutar Comando
  3. Procesar Filas en DataReader
  4. Cerrar Reader
  5. Cerrar Conexión



Acceso a Datos – Escenario Desconectado

- En un escenario desconectado, los recursos no se mantienen en el servidor mientras los datos se procesan.
- Pasos:
  1. Abrir Conexión
  2. Llenar DataSet mediante DataAdapter
  3. Cerrar Conexión
  4. Procesar DataSet
  5. Abrir Conexión
  6. Actualizar fuente de datos mediante DataAdapter
  7. Cerrar Conexión



ADO.Net

Microsoft Visual Studio .net

- Ejemplos de ADO.Net
  - Objeto Connection
  - Objeto Command
    - ExecuteReader()
    - ExecuteScalar()
    - ExecuteXmlReader()
    - ExecuteNonQuery()
    - Comandos Paramétricos
  - Objeto DataReader
  - Objeto DataAdapter
  - Objeto DataSet - DataAdapter

- Ejemplo del Objeto Connection

```
string connectionString = ConfigurationManager.ConnectionStrings["AdventureWorks"].ConnectionString;
```

```
string connectionString = "Data Source=(local); Initial Catalog=Adventureworks; " +
                          "Integrated Security=SSPI; Persist Security Info=False";

SqlConnection connection = new SqlConnection(connectionString);
connection.Open();

using (connection)
{
    // Use the database connection.
}
```

```
SqlConnection Cn = new SqlConnection("Data Source=localhost;Integrated Security=SSPI;Initial
Catalog=Adventureworks");

Cn.Open();

// Operaciones de Acceso a Datos

Cn.Close();
```

ADO.Net

Microsoft Visual Studio.net

- Ejemplo de objeto Connection utilizando ConnectionStringBuilder

```
using System;
using System.Data.SqlClient;

namespace ConnectionTest
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnectionStringBuilder connBuilder = new SqlConnectionStringBuilder();
            connBuilder.InitialCatalog = "TestDB";
            connBuilder.DataSource = @".\SQLExpress";
            connBuilder.IntegratedSecurity = true;

            SqlConnection conn = new SqlConnection(connBuilder.ConnectionString);

            conn.Open();
            Console.WriteLine("Connected to SQL Server v" + conn.ServerVersion);
            conn.Close();
            Console.ReadLine();
        }
    }
}
```

- Ejemplo de ExecuteReader() 1/2

```
// Middle-tier class, which contains a business method that returns a data reader.
public class MyProductsDAC
{
    public static SqlDataReader GetProductReader()
    {
        SqlConnection connection = new SqlConnection(aConnectionString);
        connection.Open();

        string sql = "SELECT Name, ListPrice FROM Production.Product";
        SqlDataReader reader;

        using (SqlCommand command = new SqlCommand(sql, connection))
        {
            command.CommandType = CommandType.Text;
            reader = command.ExecuteReader(CommandBehavior.SingleResult |
                    CommandBehavior.CloseConnection);

        } // Dispose command object.

        return reader;
    }
}
```

- Ejemplo de ExecuteReader() 2/2

```
// User-interface component, which calls the middle-tier business method.
public class MyUserInterfaceComponent
{
    public void MyMethod()
    {
        using (SqlDataReader reader = MyProductsDAC.GetProductReader())
        {
            while (reader.Read())
            {
                Console.WriteLine("Product name: {0}", reader.GetString(0));
                Console.WriteLine("Product price: {0}", reader.GetDecimal(1));
            }
        } // Dispose data reader, which also closes the connection.
    }
}
```

- Ejemplos de ExecuteScalar()

```
SqlConnection connection = new SqlConnection(aConnectionString);
SqlCommand command = new SqlCommand( "SELECT COUNT(*) FROM Production.Product", connection);
command.CommandType = CommandType.Text;

connection.Open();
int count = (int)command.ExecuteScalar();
connection.Close();
```

```
using (SqlConnection connection = new SqlConnection(aConnectionString))
{
   connection.Open();

   using (SqlCommand command = new SqlCommand(
         "SELECT AVG(ListPrice) FROM Production.Product",
         connection))
   {
      command.CommandType = CommandType.Text;
      decimal count = (decimal) command.ExecuteScalar();

      // Use query result.

   } // Dispose command object.
} // Dispose (close) connection object.
```

- Ejemplo de ExecuteXmlReader()

```
using (SqlConnection connection = new SqlConnection(aConnectionString))
{
connection.Open();

    using ( SqlCommand command = new SqlCommand("SELECT Name, ListPrice FROM
                Production.Product FOR XML AUTO", connection) )
    {
        command.CommandType = CommandType.Text;

        XmlReader reader = command.ExecuteXmlReader();

        while (reader.Read())
        {
            // Process XML node.
        }

        reader.Close();

    } // Dispose command object.
} // Dispose (close) connection object.
```

- Ejemplo de ExecuteNonQuery() 1/2

```
public static void ExecuteNonQueryExample(IDbConnection con)
    {
        // Create and configure a new command.
        IDbCommand com = con.CreateCommand();
        com.CommandType = CommandType.Text;
        com.CommandText = "UPDATE Employees SET Title = 'Sales Director' WHERE EmployeeId = '5'";

        // Execute the command and process the result.
        int result = com.ExecuteNonQuery();

        // Check the result
        if (result == 1)
        {
            Console.WriteLine("Employee title updated.");
        }
        else
        {
            Console.WriteLine("Employee title not updated.");
        }
    }
```

- Ejemplo de ExecuteNonQuery() 2/2

```
using (SqlConnection connection = new SqlConnection(aConnectionString))
{
    connection.Open();

    using (SqlCommand command = new SqlCommand("uspUpdatePrice", connection))
    {
        command.CommandType = CommandType.StoredProcedure;

        command.Parameters.Add("@prmProductID", SqlDbType.Int).Value = aProductID;
        command.Parameters.Add("@prmAmount", SqlDbType.Money).Value = anAmount;

        int rowsAffected = command.ExecuteNonQuery();

        if (rowsAffected == 1)
            Console.WriteLine("The product has been updated successfully.");
        else
            Console.WriteLine("The product has not been updated.");

    } // Dispose command object.
} // Dispose (close) connection object.
```

- Ejemplo de Comandos Paramétricos 1/2

```
string sql = "SELECT Name, ListPrice  FROM Production.Product  WHERE Name LIKE @NamePart";

using (SqlConnection connection = new SqlConnection(aConnectionString))
{
    connection.Open();

    using (SqlCommand command = new SqlCommand(sql, connection))
    {
        command.CommandType = CommandType.Text;
        command.Parameters.Add("@NamePart", SqlDbType.NVarChar, 50).Value = aName;
        using (SqlDataReader reader = command.ExecuteReader())
        {
            while (reader.Read())
            {
                Console.WriteLine("Product name: {0}", reader.GetString(0));
                Console.WriteLine("Product price: {0}", reader.GetDecimal(1));
            }
        } // Dispose data reader.
    } // Dispose command object.
} // Dispose (close) connection object.
```

• Ejemplo de Comandos Paramétricos 2/2

```
using (SqlConnection connection = new SqlConnection(aConnectionString))
{
   connection.Open();

   using (SqlCommand command = new SqlCommand("uspGetProductsAbovePrice",  connection))
   {
      command.CommandType = CommandType.StoredProcedure;

      command.Parameters.Add("@prmPrice", SqlDbType.Money).Value = aPrice;
      command.Parameters.Add("@prmAverage", SqlDbType.Money).Direction = ParameterDirection.Output;
      command.Parameters.Add("@RETURN_VALUE", SqlDbType.Int).Direction = ParameterDirection.ReturnValue;

      using (SqlDataReader reader = command.ExecuteReader())
      {
         while (reader.Read())
         {
            Console.WriteLine("Product name: {0}", reader.GetString(0));
            Console.WriteLine("Product price: {0}", reader.GetDecimal(1));
         }

      } // Dispose (close) data reader object.

      decimal average = (decimal) command.Parameters["@prmAverage"].Value;
      Console.WriteLine("Average price above threshold: {0}", average);

      int count = (int) command.Parameters["@RETURN_VALUE"].Value;
      Console.WriteLine("Number of products above threshold: {0}", count);

   } // Dispose command object.
} // Dispose (close) connection object.
```

- Ejemplo de DataReader 1/4

```
using System.Data.SqlClient;
class DataReaderExample
{
    public static void Main()
    {
            SqlConnection Cn = new SqlConnection( "Data Source=localhost; Integrated   Security=SSPI; Initial
        Catalog=example");
            SqlCommand Cmd = Cn.CreateCommand();
            Cmd.CommandText = "SELECT Name, Age FROM Employees";
            Cn.Open();

            SqlDataReader Rdr = Cmd.ExecuteReader();
            while (Rdr.Read())
            {
                System.Console.WriteLine( "Name: {0}, Age: {1}", Rdr.GetString(0), Rdr.GetInt32(1));
            }
            Rdr.Close();
            Cn.Close();
    }
}
```

- Ejemplo de DataReader 2/4

```
class Program
{
        static void Main(string[] args)
        {
          SqlConnectionStringBuilder scb = new SqlConnectionStringBuilder();
          scb.DataSource = @".\SQLExpress";
          scb.InitialCatalog = "TestDB";
          scb.IntegratedSecurity = true;
          SqlConnection conn = new SqlConnection(scb.ConnectionString);
          conn.Open();

          SqlCommand cmd = conn.CreateCommand();
          cmd.CommandText = "SELECT * From Users";
          SqlDataReader rdr = cmd.ExecuteReader();

          while (rdr.Read())
          {
            Console.WriteLine(string.Format("User {0}, Full Name {1} {2}", rdr.GetString(rdr.GetOrdinal("UserName")),
              (string)rdr["FirstName"], rdr.GetString(rdr.GetOrdinal("LastName"))));
          }
          rdr.Close();
          conn.Close();
          Console.ReadLine();
        }
}
```

- Ejemplo de DataReader 3/4

```
class Program
{
   static void Main()
   {
      string connectionString = "Data Source=(local);Initial Catalog=Northwind; Integrated
Security=SSPI";
      string queryString = "SELECT CategoryID, CategoryName FROM dbo.Categories;";

      using (SqlConnection connection = new SqlConnection(connectionString))
      {
         SqlCommand command = connection.CreateCommand();
         command.CommandText = queryString;

         try
         {
            connection.Open();

            SqlDataReader reader = command.ExecuteReader();

            while (reader.Read())
            {
               Console.WriteLine("\t{0}\t{1}", reader[0], reader[1]);
            }
            reader.Close();
         }
         catch (Exception ex)
         {
            Console.WriteLine(ex.Message);
         }
      }
   }
}
```

- Ejemplo de DataReader 4/4

```
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();
    }
}
```

ADO.Net

Microsoft Visual Studio.net

- Ejemplo de DataAdapter/DataSet: Llenado de un DataSet con una Tabla

```
private void FillUserTable()
{
   DataSet myData = new DataSet();

   string connectionString = "Data Source=(local);Initial Catalog=Test;Integrated
Security=SSPI;";

   using (SqlConnection testConnection = new SqlConnection(connectionString))
   {
      SqlCommand testCommand = testConnection.CreateCommand();
      testCommand.CommandText = "Select * from userTable";

      SqlDataAdapter dataAdapter = new SqlDataAdapter(testCommand);

      dataAdapter.Fill(myData, "UserTable");

   } // testConnection.Dispose called automatically.
}
```

ADO.Net

Microsoft Visual Studio.net

- Ejemplo de DataAdapter/DataSet: Llenado de un DataSet de 2 Tablas

```
private void buttonFillData_Click(object sender, EventArgs e)
{
    string connectionString =
        "Data Source=(local);Initial Catalog=Test;Integrated Security=SSPI;";

    DataSet userData = new DataSet();

    using (SqlConnection testConnection = new SqlConnection(connectionString))
    {
        SqlCommand testCommand = testConnection.CreateCommand();

        testCommand.CommandText =
            "Select FirstName, LastName from userTable;" +
            " Select PermissionType from PermissionsTable";

        SqlDataAdapter dataAdapter = new SqlDataAdapter(testCommand);

        dataAdapter.Fill(userData);

    } // testConnection.Dispose called automatically.
}
```

- Ejemplo de DataAdapter/DataSet: Llenado de un DataSet de 2 Tablas mediante un Stored Procedure

```
Create Procedure GetMultipleResults
As
Begin
   Select FirstName, LastName from userTable;
   Select PermissionType from PermissionsTable;
End
```