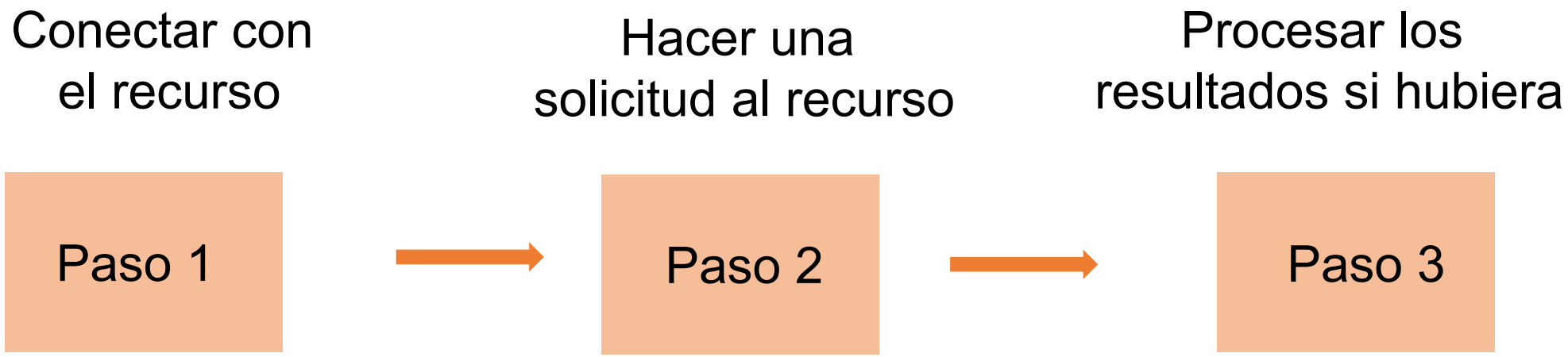
Conectando a una Bases de datos con ADO.NET

- ADO.NET
- Cadena de Conexión a una base de datos
- Objetos ADO.Net
- Objeto Connection
- Objeto Command()
- Objeto DataReader()
- Secuencia para trabajar con una bd
- Ejemplo completo

Bases de datos con ADO.Net

Pasos para recuperar datos de un recurso externo.



Solicitar un recurso

Ejemplo para recuperar recursos de un API

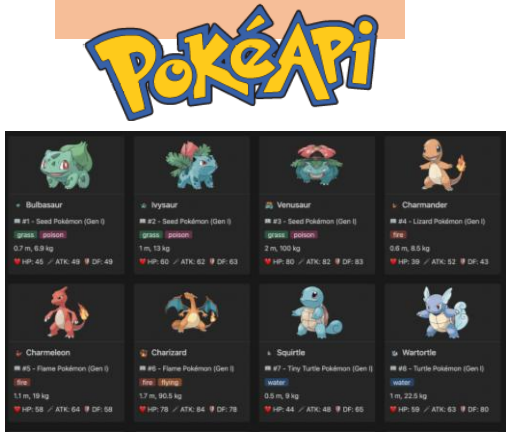
Conectar con el recurso (Base de datos)

Hacer una solicitud al recurso

Procesar los resultados si hubiera

Paso 1

Paso 2



Paso 3

Solicitar un recurso

Ejemplo para recuperar recursos de una base de datos

Conectar con el recurso (Base de datos)

Hacer una solicitud al recurso

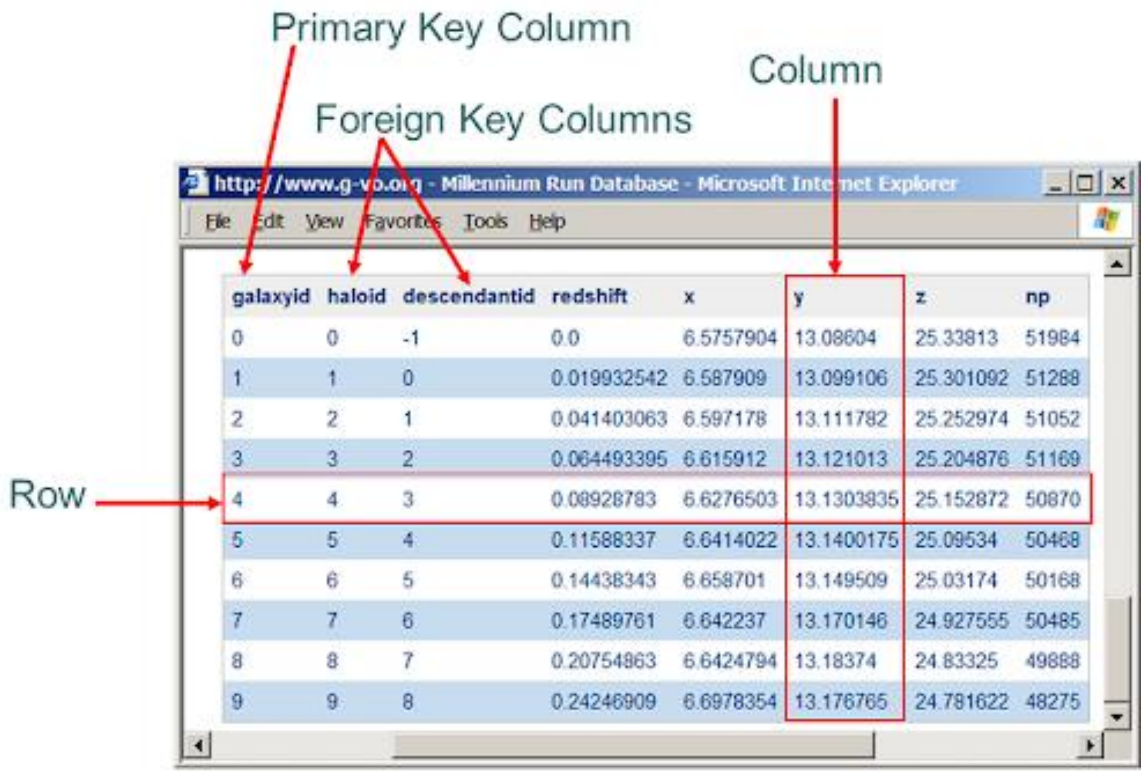
Procesar los resultados si hubiera

Paso 1 → Paso 2 → Paso 3

Bases de datos con ADO.Net

Recordando

Estructura de una base de datos relacional

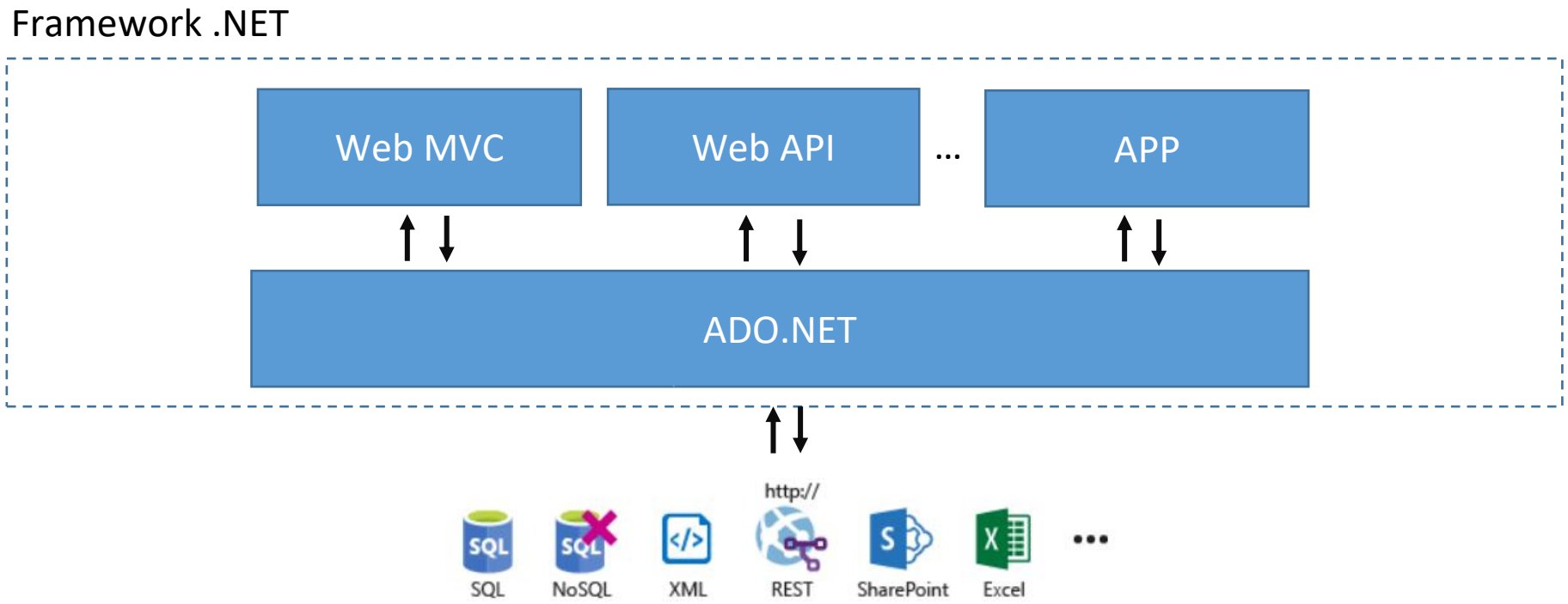


galaxyid	haloid	descendantid	redshift	x	y	z	np
0	0	-1	0.0	6.5757904	13.08604	25.33813	51984
1	1	0	0.019932542	6.587909	13.099106	25.301092	51288
2	2	1	0.041403063	6.597178	13.111782	25.252974	51052
3	3	2	0.064493395	6.615912	13.121013	25.204876	51169
4	4	3	0.08928783	6.6276503	13.1303835	25.152872	50870
5	5	4	0.11588337	6.6414022	13.1400175	25.09534	50468
6	6	5	0.14438343	6.658701	13.149509	25.03174	50168
7	7	6	0.17489761	6.642237	13.170146	24.927555	50485
8	8	7	0.20754863	6.6424794	13.18374	24.83325	49888
9	9	8	0.24246909	6.6978354	13.176765	24.781622	48275

Bases de datos con ADO.Net

Herramientas para trabajar Bases de datos con c#

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para programadores de .NET Framework. ADO.NET ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de datos distribuida

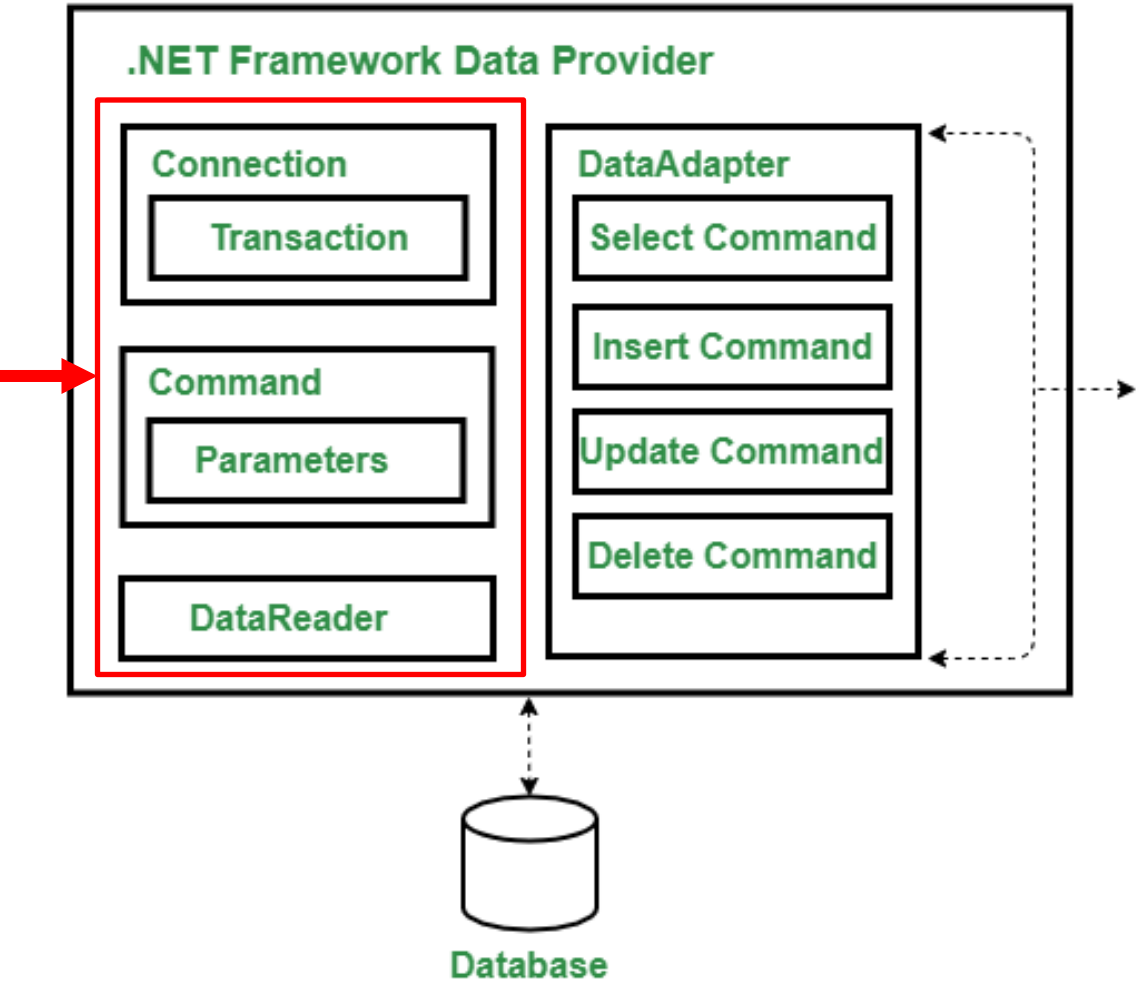


https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/

Bases de datos con ADO.Net

Componentes de ADO Net.

En la materia solo usamos estos tres objetos



Bases de datos con ADO.Net

Librería de nuget

https://www.nuget.org/packages/Microsoft.Data.Sqlite.Core/

Dirección en nuget de la librería para trabajar con SQLite

dotnet add package Microsoft.Data.Sqlite

Comando para incluir SQLite en un proyecto .Net

Bases de datos con ADO.Net

Componentes de ADO Net.

Connection: Permite establecer una conexión con la base de datos.

Command: Permite enviar órdenes SQL para ser ejecutados por la base de datos.

DataReader: Permite leer un flujo de filas de solo avance desde una base de datos.

Conectar con una base de datos

Cadena de conexión

- Una cadena de conexión **contiene información de inicialización** que se transfiere como un parámetro desde un proveedor de datos a un origen de datos.
- Suele ser un conjunto de **claves** y **valores** separados por punto y coma “;”. El conjunto de claves y valores esta conectado por el signo de igual por ejemplo clave1=valor1;clave2=valor2.
- El conjunto de claves y valores disponibles están definidos por el proveedor de la base de datos y muchas veces hay inconsistencias entre las claves de diferentes proveedores de base de datos.

Cadena de conexión para sqlite

```
string CadenaDeConexion  =  "Data Source=Instituto.db;"; // donde Instituto.db es el nombre de la base de datos
```

Conectar con una base de datos

La clase Connection

- Podemos utilizar el objeto Connection para conectar a una fuente de datos específica.
- Establece y gestiona una conexión a una fuente de datos específica.
- método **Open()** que abre la conexión
- método **Close()** que cierra la conexión a la base de datos

```
string connectionString = "Data Source=(Tienda.db); "; // donde tienda.db es el nombre de la base de datos


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

            [….]

        connection.Close();
}
```

Abrir una conexión

Cerrar una conexión

Bases de datos con ADO.Net

La clase Command()

- Permite especificar las órdenes, generalmente en SQL, permite consultar y modificar el contenido de la base de datos: **Select, Insert**, **Delete** y **Update**.
- Recibe como referencia una conexión y un comando SQL
- Devuelve un Objeto DataReader() con los resultados
- Métodos parameters() para pasar parámetros al comando SQL de forma segura.

Solicitudes a una base de datos

La clase Command()

```
string queryString = "SELECT Nombre, FechaNacimiento FROM Empleados;";

var command = new SqlCommand( queryString, connection);
connection.Open();
using(var reader = command.ExecuteReader())
{
        while (reader.Read())
        {[…]} // Resultados obtenidos (1 fila por vez)
}
```

Solicitudes a una base de datos

Posibles forma de ejecutar con Command()

ExecuteReader	Ejecuta comandos que devuelven filas. Se utiliza normalmente con las Instrucciones **SELECT**
ExecuteNonQuery	Ejecuta comandos como las instrucciones **INSERT, DELETE, UPDATE.** command.ExecuteNonQuery();
ExecuteScalar	Recupera un valor único de una base de datos. Int IdBuscado = (Int32)command.ExecuteScalar();

Resultados de una solicitud

La clase DataReader

- Permite realizar lecturas forma eficiente de grandes cantidades de datos que no caben en memoria de una sola vez.
- Provee solo acceso de modo lectura, no puede escribir datos en la base de datos.
- Representación en memoria de una fila (Row) de una Tabla.
- Permite recorrer la Tabla (solo hacia abajo) de fila en fila.
- Muestra los datos siempre y cuando se mantenga conectado a la base de datos. Si la conexión se pierde, los datos también.
- Accedemos a las columnas de una DataReader con corchetes

```
using(SqlDataReader reader = command.ExecuteReader())
{
        while (reader.Read())
        {
                Console.WriteLine(String.Format("{0}, {1}", reader[0], reader[1]));
        }
 }
```

Bases de datos con ADO.Net

Pasos para operar con ADO.NET

- Seteamos el tipo de conexión, especificamos la fuente de datos. En el Objeto **Connection.**
- **Abrimos** la conexión a la fuente de datos en el objeto **Connection**.
    - Ejecutamos un comando SQL en la base de datos usando la clase **Command()**
    - **Si el comando devuelve datos** (Select)
        - Debemos usar command.ExecuteReader() que nos devuelve un objeto DataReader();
        - Leer los datos devueltos por el objeto DataReader();
    - **Si el comando NO devuelve** (Update, Insert, Delete)
        - Se ejecuta con ExecuteNonReader y nos tira una excepción si algo salió mal
- **Cerramos** la conexión después de trabajar con los datos en el objeto **Connection**.

Bases de datos con ADO.Net

Ejemplo completo de ADO Net.

```
private static void GetEmpleados(string connectionString)
{
   string queryString = "SELECT Nombre, IDEmpleado FROM Empleado;";

   using (SqliteConnection connection = new SqliteConnection(connectionString))
   {
      SqliteCommand command = new SqliteCommand(queryString, connection);
      connection.Open();
      using(SqliteDataReader reader = command.ExecuteReader())
      {
         while (reader.Read())
         {
            Console.WriteLine(String.Format("{0}, {1}",reader[“IDEmpleado”], reader[“Nombre”]));
         }
      }
     connection.Close();
   }
}
```

Resultados de una solicitud

La clase DataReader

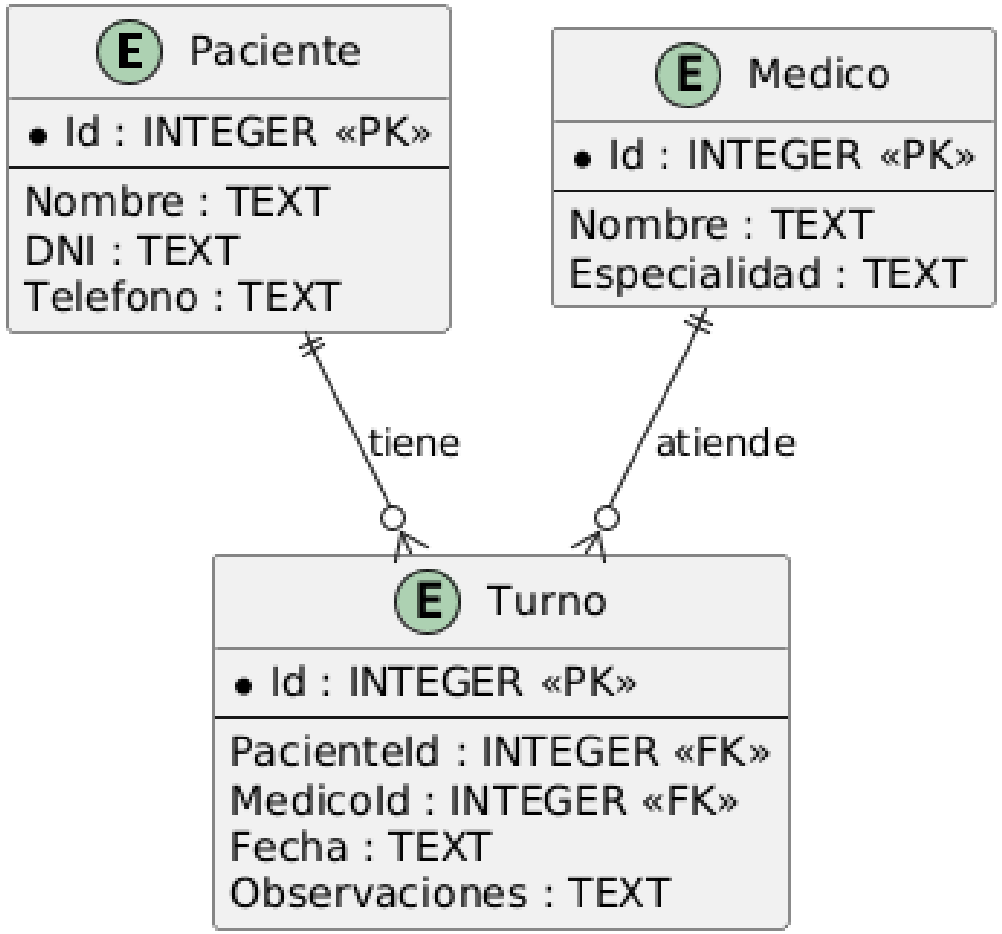
- Permite realizar lecturas forma eficiente de grandes cantidades de datos que no caben en memoria de una sola vez.
- Provee solo acceso de modo lectura, no puede escribir datos en la base de datos.
- Representación en memoria de una fila (Row) de una Tabla.
- Permite recorrer la Tabla (solo hacia abajo) de fila en fila.
- Muestra los datos siempre y cuando se mantenga conectado a la base de datos. Si la conexión se pierde, los datos también.
- Accedemos a las columnas de una DataReader con corchetes

Caso de uso

Bases de datos con ADO.Net

Bases de datos con ADO.Net

Diseño de la base datos




```
CREATE TABLE Paciente (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Nombre TEXT NOT NULL,
    DNI TEXT,
    Telefono TEXT
);

CREATE TABLE Medico (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Nombre TEXT NOT NULL,
    Especialidad TEXT
);

CREATE TABLE Turno (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    PacienteId INTEGER NOT NULL,
    MedicoId INTEGER NOT NULL,
    Fecha TEXT NOT NULL,
    Observaciones TEXT,
    FOREIGN KEY (PacienteId) REFERENCES Paciente(Id),
    FOREIGN KEY (MedicoId) REFERENCES Medico(Id)
);
```

Bases de datos con ADO.Net

Ejemplo de un Select que devuelve un solo objeto

```
string cadenaConexion = "Data Source=clinica.db";

public Paciente ObtenerPacientePorId(int id)
{
    using var conexion = new SQLiteConnection(cadenaConexion);
    conexion.Open();

    string sql = "SELECT Id, Nombre, DNI, Telefono FROM Paciente WHERE Id = @Id";

    using var comando = new SQLiteCommand(sql, conexion);
    comando.Parameters.Add(new SQLiteParameter("@Id", id));

    using var lector = comando.ExecuteReader();

    if (lector.Read()) // si encontró un registro
    {
        var paciente = new Paciente
        {
            Id = Convert.ToInt32(lector["Id"]),
            Nombre = lector["Nombre"].ToString(),
            DNI = lector["DNI"].ToString(),
            Telefono = lector["Telefono"].ToString()
        };
        return paciente;
    }

   return null;
}
```

Bases de datos con ADO.Net

Otro Ejemplo de un Select con un list

```
public List<Paciente> BuscarPacientesPorNombre(string nombreParcial)
{
    var pacientes = new List<Paciente>();

    using var conexion = new SQLiteConnection(cadenaConexion);
    conexion.Open();

    string sql = "SELECT Id, Nombre, DNI, Telefono FROM Paciente WHERE Nombre LIKE @Nombre";
    using var comando = new SQLiteCommand(sql, conexion);

    // Ejemplo: si se pasa “Ana”, busca “%Ana%”
    comando.Parameters.AddWithValue("@Nombre", $"%{nombreParcial}%");

    using var lector = comando.ExecuteReader();

    while (lector.Read())
    {
        var p = new Paciente
        {
            Id = Convert.ToInt32(lector["Id"]),
            Nombre = lector["Nombre"].ToString(),
            DNI = lector["DNI"].ToString(),
            Telefono = lector["Telefono"].ToString()
        };
        pacientes.Add(p);
    }

    return pacientes;
}
```

Bases de datos con ADO.Net

Otro Ejemplo de un Select con un list

```
public List<Turno> ListarTurnos()
{
        using var conexion = new SQLiteConnection(cadenaConexion);
        conexion.Open();

        string sql = @"
            SELECT
                t.Id, t.Fecha, t.Observaciones,
                p.Id AS PacienteId, p.Nombre AS PacienteNombre,
                m.Id AS MedicoId, m.Nombre AS MedicoNombre, m.Especialidad
            FROM Turno t
            INNER JOIN Paciente p ON t.PacienteId = p.Id
            INNER JOIN Medico m ON t.MedicoId = m.Id
            ORDER BY t.Fecha DESC";

        using var comando = new SQLiteCommand(sql, conexion);
        using var lector = comando.ExecuteReader();

        var turnos = mapearTurnos (lector);

        return turnos;
}
```

```
public Turno mapearTurnos(Reader lector)
{
    var turnos = new List<Turno>();
     while (lector.Read())
       {
          var turno = new Turno
          {
             Id = Convert.ToInt32(lector["Id"]),
             Fecha = DateTime.Parse(lector["Fecha"].ToString()),
             Observaciones = lector["Observaciones"].ToString(),
             Paciente = new Paciente
             {
                Id = Convert.ToInt32(lector["PacienteId"]),
                Nombre = lector["PacienteNombre"].ToString()
             },
             Medico = new Medico
             {
                Id = Convert.ToInt32(lector["MedicoId"]),
                Nombre = lector["MedicoNombre"].ToString(),
                Especialidad = lector["Especialidad"].ToString()
             }
          };

          turnos.Add(turno);
       }
}
```

Bases de datos con ADO.Net

Clases modelo de la base de datos

```
public class Paciente
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string DNI { get; set; }
    public string Telefono { get; set; }
}

public class Medico
{
    public int Id { get; set; }
    public string Nombre { get; set; }
    public string Especialidad { get; set; }
}
```

```
public class Turno
{
    public int Id { get; set; }
    public int PacienteId { get; set; }
    public int MedicoId { get; set; }
    public DateTime Fecha { get; set; }
    public string Observaciones { get; set; }

    // Propiedades de navegación (opcional)
    public Paciente Paciente { get; set; }
    public Medico Medico { get; set; }
}
```

Bases de datos con ADO.Net

Ejemplo de un Insert

```
string cadenaConexion = "Data Source=clinica.db";

public void InsertarPaciente(Paciente paciente)
{
        using var conexion = new SQLiteConnection(cadenaConexion);
        conexion.Open();

        string sql = "INSERT INTO Paciente (Nombre, DNI, Telefono)
                        VALUES (@Nombre, @DNI, @Telefono)";

        using var comando = new SQLiteCommand(sql, conexion);

        comando.Parameters.Add(new SQLiteParameter("@Nombre", paciente.Nombre));
        comando.Parameters.Add(new SQLiteParameter("@DNI", paciente.DNI));
        comando.Parameters.Add(new SQLiteParameter("@Telefono", paciente.Telefono));

        comando.ExecuteNonQuery();
}
```

Bases de datos con ADO.Net

Ejemplo de un UPDATE

```
string cadenaConexion = "Data Source=clinica.db";

 public void ActualizarTelefonoPaciente(int id, string nuevoTelefono)
 {
        using var conexion = new SQLiteConnection(cadenaConexion);
        conexion.Open();

        string sql = "UPDATE Paciente SET Telefono = @Telefono WHERE Id = @Id";
        using var comando = new SQLiteCommand(sql, conexion);

        comando.Parameters.Add(new SQLiteParameter("@Telefono", nuevoTelefono));
        comando.Parameters.Add(new SQLiteParameter("@Id", id));

        comando.ExecuteNonQuery();
}
```

Bases de datos con ADO.Net

Ejemplo de un DELETE

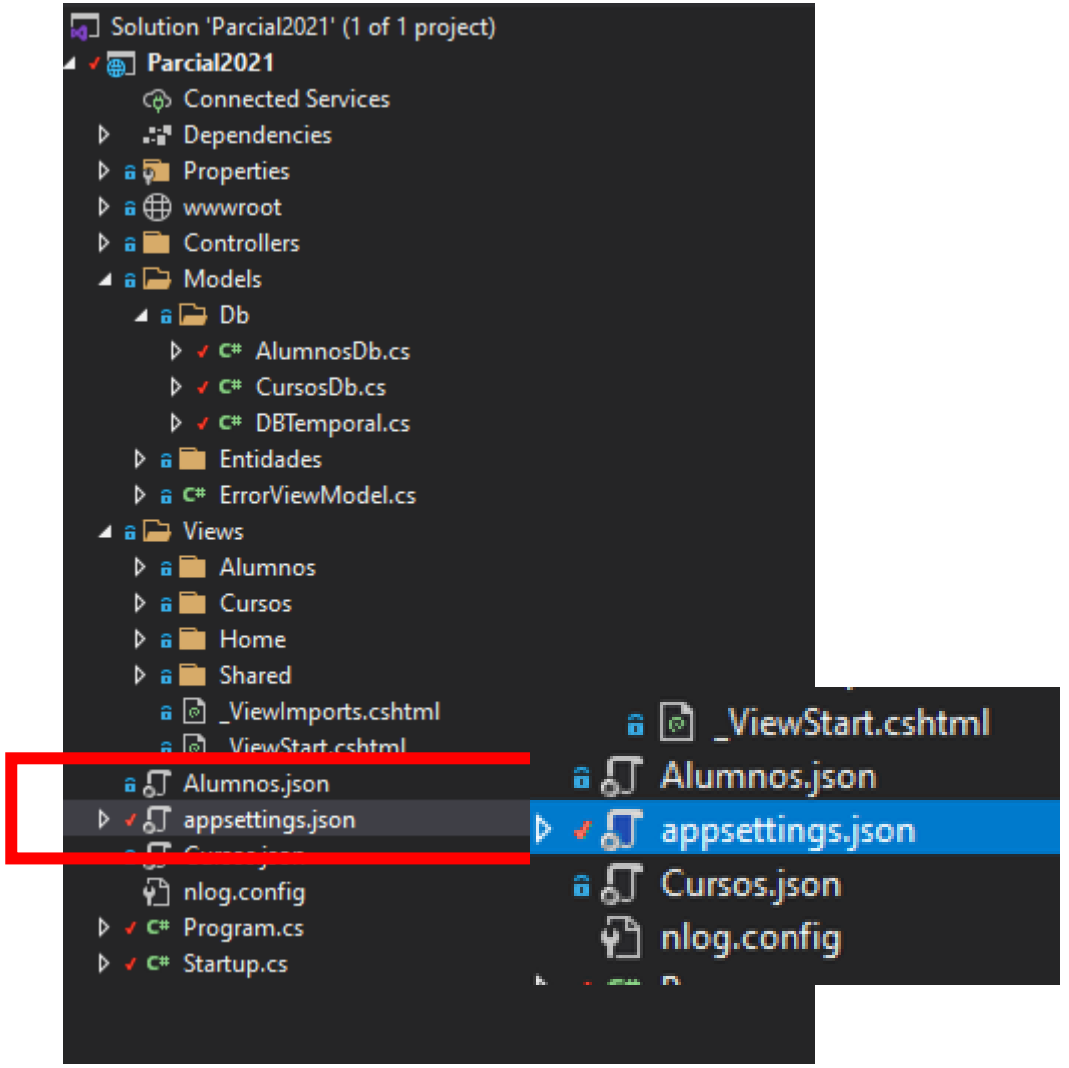
```
string cadenaConexion = "Data Source=clinica.db";

 public void EliminarPaciente(int id)
 {
        using var conexion = new SQLiteConnection(cadenaConexion);
        conexion.Open();

        string sql = "DELETE FROM Paciente WHERE Id = @Id";
        using var comando = new SQLiteCommand(sql, conexion);

        comando.Parameters.Add(new SQLiteParameter("@Id", id));
        comando.ExecuteNonQuery();
}
```

Bases de datos con ADO.Net

Almacenar cadena de conexión

El archivo ***appsettings.json*** es un archivo de configuración de la aplicación que se utiliza para almacenar las opciones de configuración, como las cadenas de conexiones de la base de datos, las variables globales del ámbito de la aplicación, etc.

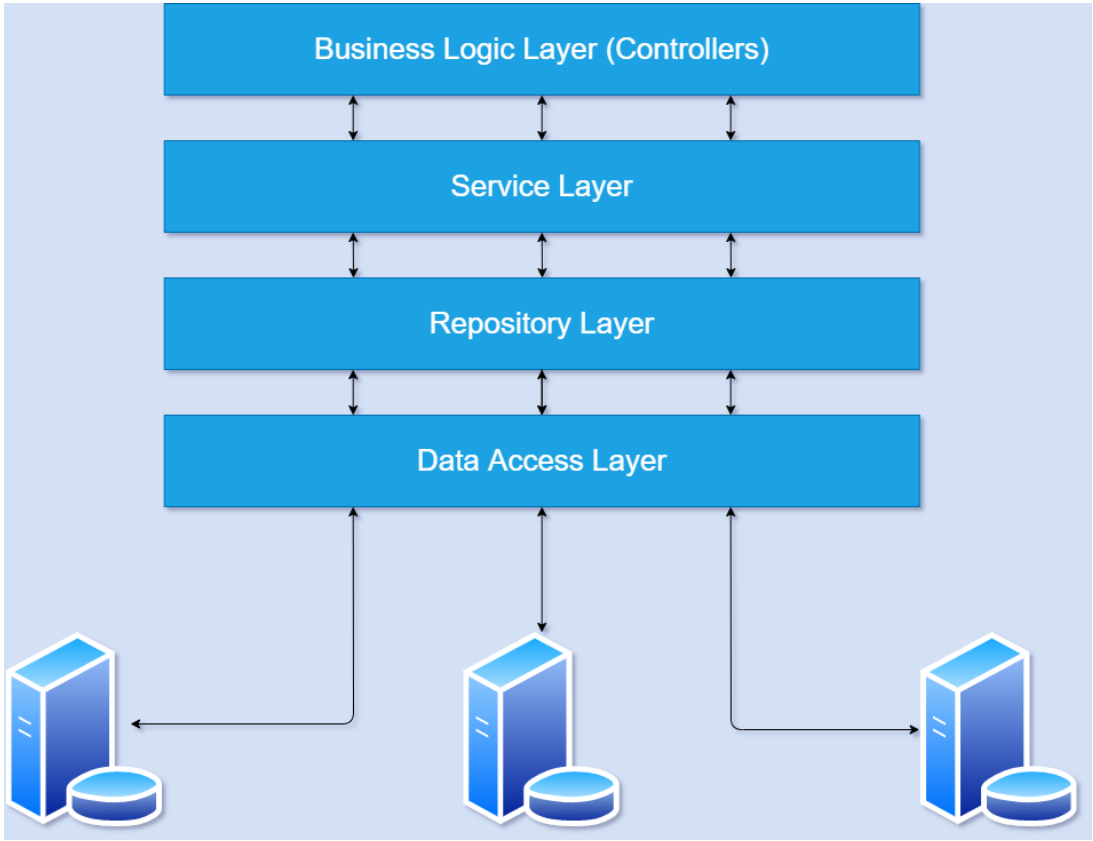
Este archivo se crea con el Template de ASP net core: Empty Project template or Razor Pages or MVC Template or Web API Template.



https://dotnettutorials.net/lesson/asp-net-core-appsettings-json-file/

Bases de datos

Arquitectura Propuesta para acceso a datos



https://www.c-sharpcorner.com/UploadFile/puranindia/ado-net-interview-questions-and-answers/

https://docs.microsoft.com/es-es/dotnet/framework/data/adonet/